LAPORAN PRAKTIK INDUSTRI

PENGEMBANGAN SOFTWARE WEB API PADA STARTUP DIGITAL KIPAPOS (POINT OF SALE)

PT KIPA TEKNOLOGI INDONESIA

Ruko De Castello, Jl Simpang Sulfat Selatan Kav 1 No 4, Pandanwangi, Kec. Blimbing, Kota Malang 65124

(2 Januari 2020 – 2 Februari 2020)

Disusun Oleh :

Alfarady Raja Ghanie Hamid Jauhar

NIM : 17050974023

PRODI S1 PENDIDIKAN TEKNOLOGI INFORMASI

JURUSAN TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS NEGERI SURABAYA

2020

LEMBAR PENGESAHAN

Laporan Praktik Industri/Praktik Kerja Lapangan :

Judul	:	Pengembangan Software Web *API* pada Startup Digital KipaPOS (*Point of Sale*)
Nama Industri	:	PT Kipa Teknologi Indonesia
Alamat Industri	:	Ruko De Castello, Jl Simpang Sulfat Selatan Kav 1 No 4, Pandanwangi, Kec. Blimbing, Kota Malang 65124
No.Telepon	:	082268888859

Yang dilaksanakan oleh mahasiswa Fakultas Teknik (FT) Universitas Negeri Surabaya :

Nama	:	Alfarady Raja Ghanie Hamid Jauhar
NIM	:	17050974023
Program Studi	:	S1 Pendidikan Teknologi Informasi
Jurusan	:	Teknik Informatika

Telah diseminarkan/diuji dan dinyatakan lulus.

Surabaya, 30 Januari 2020

Menyetujui	Mengetahui/Menyetujui
Penguji	Pembimbing Industri
……...	Mochamad Nor Fadillah S.Kom
NIDN.	Pembimbing Industri

Mengesahkan	Menyetujui
Wakil Dekan I	Pembimbing/Penguji,
Fakultas Teknik Unesa,	
Drs. Edy Sulistiyo, M.Pd.	Dodik Arwin D, S.ST., S.T., MT.
NIP. 196404201991031005	NIP. 197801082000121001

KATA PENGANTAR

Puji syukur kehadirat Allah SWT atas hidayahnya sehingga penulis dapat menyelesaikan Laporan Praktik Industri yang berjudul **"Pengembangan Software Web *API* pada Startup Digital KipaPOS (*Point of Sale*)"** tepat pada waktunya dan dilakukan dengan baik. Maksud dari penyusunan Laporan Praktik Industri ini yaitu untuk memenuhi mata kuliah Praktik Industri yang dilaksanakan di PT Kipa Teknologi Indonesia. Keberhasilan Praktik Industri ini tidak lepas dari bimbingan dan bantuan dari beberapa pihak. Untuk itu tidak lupa ucapkan terima kasih ditujukan kepada :

1. Bapak Drs. Edy Sulistyo,M.Pd. selaku Pembantu Dekan I Fakultas Teknik Universitas Negeri Surabaya.
2. Bapak I Kadek Dwi Nuryana, S.T., M.Kom. selaku Ketua Jurusan Teknik Informatika.
3. Bapak Bambang Sudjatmiko, S.Pd., M.T. selaku Koordinator Praktik Industri.
4. Bapak Dodik Arwin Dermawan, SST., ST., MT. selaku Pembimbing Praktik Industri.
5. Bapak Mochamad Nor Fadillah S.Kom selaku Pembimbing Praktik Industri di PT Kipa Teknologi Indonesia.

Penulis menyadari bahwa penyusunan Laporan Praktik Industri ini masih jauh dari kesempurnaan, oleh karena itu penulis mengharapkan saran dan kritik yang membangun dari Bapak/Ibu Dosen dan pembaca sekalian. Semoga laporan ini mendatangkan manfaat bagi pembaca. Terima Kasih.

Surabaya, 03 Maret 2020

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

Halaman ini sengaja dikosongkan

BAB I
PENDAHULUAN

A. Latar Belakang

Teknologi informasi adalah suatu teknologi yang dapat digunakan untuk mengolah suatu data atau informasi, yang didalamnya dapat memproses menyusun, menyimpan, bahkan memanipulasi data. Teknologi informasi saat ini berkembang dengan pesat. Sistem – sistem yang dijalankan di perusahaan secara manual mulai berubah dan dilakukan secara terkomputerisasi. Peranan informasi yang semakin mempengaruhi terhadap perkembangan kinerja manusia khususnya dalam bidang penganalisaan data, lebih-lebih komputer pada saat ini sudah bukan lagi hal yang asing di tengah-tengah masyarakat. Komputer dan Mobile merupakan sarana untuk mengelola data atau informasi yang lebih akurat, dan daya tahannya untuk melakukan pemerosesan data dalam jumlah besar dan bisa dipertanggungjawabkan.

Indonesia hingga akhir tahun 2018 lalu, jumlah usaha mikro di Indonesia mencapai 58,91 juta dan usaha kecil 59,260. Adapun jumlah usaha menengah mencapai 4,987 (Candra, 2019) dan semakin banyak generasi baru yang ingin menjadi wirausaha. Namun tantangan wirausaha selain pengembangan produk, juga dalam pengelolaan penjualan, keuangan, karyawan, permodalan, dan lainnya. Teknologi dapat membantu wirausaha mempermudah mengelola bisnis seperti pengelolaan penjualan, laporan laba rugi, laporan neraca keuangan, pengelolaan hak akses pegawai, pencatatan biaya pengeluaran ataupun pemasukan, hingga peringatan stok barang.

Tujuan dibuatnya Aplikasi Point of Sale (KipaPOS) berbasis Web API yang nantinya akan di Implementasikan pada Mobile App adalah untuk mengatasi masalah yang ada pada UMKM saat ini. Aplikasi yang dihasilkan diharapkan mampu memudahkan wirausaha mulai dari pengelolaan penjualan, pelanggan, pembayaran, inventori, keuangan, karyawan, sampai dengan pembiayaan usaha. Aplikasi ini diharapkan dapat membuat bisnis jadi lebih mudah, semakin rendah biaya, penjualan dan pelayanan meningkat, dan pengelolaan bisnis dapat dilakukan secara online darimana saja.

B. Tujuan PI

Tujuan dari kegiatan PI (Praktik Industri) di PT Kipa Teknologi Indonesia ini adalah sebagai berikut.

1. Tujuan Umum

Adapun tujuan umum yang ingin dicapai, antara lain:

a. Untuk memenuhi syarat kelulusan mata kuliah Praktik Industri.
b. Untuk membiasakan diri terhadap situasi dan kondisi di dunia kerja yang sebenarnya terutama berkenaan dengan sikap disiplin kerja dan profesionalitas.

2. Tujuan Khusus

Adapun tujuan khusus yang ingin dicapai, yaitu untuk membuat Aplikasi *Point of Sale* Berbasis Web *API* di PT Kipa Teknologi Indonesia.

C. Manfaat PI

Manfaat kegiatan PI (Praktik Industri) di PT Kipa Teknologi Indonesia ini adalah sebagai berikut.

1. **Bagi Mahasiswa**
   a. Mahasiswa memahami struktur organisasi di perusahaan dan dapat menerapkannya secara profesional sesuai dengan kebutuhan di perusahaan.
   b. Memiliki pembelajaran tentang perusahaan sehingga mampu menciptakan mental berwirausaha untuk menciptakan lapangan pekerjaan dan mampu bersaing di pasar global.
   c. Mendapatkan pengetahuan yang baru mengenai rancangan sistem informasi.

2. **Bagi Perusahaan**
   a. Perusahan akan terbantu dengan adanya tenaga dari mahasiswa – mahasiswa yang melakukan praktek industri
   b. Meningkatkan kinerja Perusahaan dalam mengembangkan startup.

3. **Bagi Universitas**

   Berfungsi sebagai pengenalan antara mahasiswa dengan PT Kipa Teknologi Indonesia dalam meningkatkan kreatifitas pribadi pada khusunya dan kualitas sumber daya manusia pada umumnya. Serta dapat membina kemitraan antara pihak Universitas dan Instansi terkait.

BAB II
KAJIAN PUSTAKA

A. UMKM

Pengertian tentang UMKM sebenarnya sudah dijelaskan secara jelas dalam UU No. 20/2008. Pada undang-undang tersebut, disebutkan bahwa UMKM adalah perusahaan kecil yang dimiliki dan dikelola oleh seseorang atau dimiliki oleh sekelompok kecil orang dengan jumlah kekayaan dan pendapatan tertentu.

UU tersebut juga menjelaskan tentang kriteria UMKM dan usaha besar, yang dibagi berdasarkan aset dan omzet.

Dari tahun ke tahun, sektor UMKM di Indonesia terus mengalami perkembangan. Kembali ke sepuluh tahun lalu, yakni pada 2009, jumlah UMKM adalah 52.764.750 unit dengan pangsa 99,99%. Memasuki tahun 2014-2016, jumlah tersebut meningkat menjadi lebih dari 57,9 juta unit.

Lalu, pada tahun 2017, UMKM di Indonesia diperkirakan sudah mencapai lebih dari 59 juta unit. Tidak mengherankan apabila UMKM menjadi salah satu bagian paling signifikan dalam tulang punggung perekonomian Indonesia dan ASEAN.

Bahkan sebanyak kurang lebih 88,8-99,9% bentuk usaha di ASEAN adalah UMKM, dengan penyerapan tenaga kerja sebanyak 51,7-97,2%.

Positifnya perkembangan sektor UMKM di Indonesia tidak bisa dilepaskan dari dukungan perbankan terkait penyaluran kredit kepada para pelaku UMKM.

B. *Point of Sale*

Point of Sale adalah suatu sistem yang digunakan oleh berbagai macam usaha ritel untuk menyelesaikan transaksi jual beli. *Point of Sale* kini sudah banyak digunakan oleh banyak jenis usaha, seperti restoran, kedai kopi, barbershop atau laundry.

Biasanya sebuah *Point of Sale* terdiri dari seperangkat mesin kasir, lengkap dengan *cash drawer* untuk menyimpan uang dan printer untuk mencetak struk.

Tetapi kini dengan berkembangnya teknologi, penggunaan mesin kasir konvensional sudah mulai digantikan oleh *Point of Sale* dengan sistem dan perangkat pendukung yang lebih modern.

Untuk perangkatnya saja, kini sudah banyak penyedia *Point of Sale* di Indonesia yang menggunakan tablet pintar berbasis Android atau iOS, ada juga yang menggunakan perangkat khusus POS yang biasa disebut terminal.

Sementara untuk konektivitas sudah banyak yang meninggalkan penggunaan kabel, contohnya untuk mencetak struk saja kini sudah bisa dihubungkan dengan printer *bluetooth* secara *wireless*.

Dengan inovasi teknologi yang ada saat ini, POS *System* kini sudah tidak hanya digunakan untuk memproses transaksi jual beli saja. Berikut beberapa manfaat POS *System* untuk berbagai macam jenis usaha:

1. Mencatat data transaksi secara lengkap dan memprosesnya menjadi laporan yang mudah dianalisa.
2. Jika sudah terintegrasi dengan sistem stok barang (*inventory*), sehingga mengantisipasi kekurangan bahan baku dan ketersediaan produk.
3. Memberikan laporan penjualan usaha Anda secara langsung atau online.
4. Segala macam bentuk perubahan terhadap data menu seperti harga dapat dilakukan dengan cepat.
5. Mencetak struk pembelian bagi pelanggan, sehingga Anda tidak perlu lagi menggunakan kertas nota/kwitansi secara manual.
6. Dengan kalkulasi otomatis, kasir tidak perlu repot menghitung kembalian dan mempercepat proses transaksi, sehingga dapat mengurangi antrian.
7. Menerima berbagai metode pembayaran baik itu tunai, non tunai, kartu debit dan kartu kredit, dll.

C. RESTful API

REST (*Representional State Transfer*) adalah suatu arsitektur metode komunikasi yang menggunakan protokol HTTP untuk pertukaran data dan metode ini sering diterapkan dalam pengembangan aplikasi. Dimana tujuannya adalah untuk menjadikan sistem yang memiliki performa yang baik, cepat dan mudah untuk di kembangkan

(*scale*) terutama dalam pertukaran dan komunikasi data.

Sebuah *REST client* akan membuat *HTTP request* ke server melalui sebuah *global ID* atau *URIs* dan server akan merespon dengan mengirimkan balik sebuah HTTP *response* sesuai yang diminta client.

Komponen HTTP *Request*:

1. HTTP *method* seperti *GET, POST, PUT, DELETE* dll sesuai dengan tugasnya masing-masing
2. URI/Routes untuk mengetahui lokasi data di server
3. HTTP *Version*, seperti HTTP v1.1
4. *Request Header*, berisi metadata seperti *Authorization*, tipe client dan lain
5. *Request Body*, data yang diberikan *client* ke server seperi *URI params*

Komponen HTTP *Response*:

1. *Response Code*, status server terhadap *request* yang diminta seperti 200, 401, 404 dan lainnya.
2. HTTP *Version*
3. *Response Header* yang berisi meta data seperti *contect type, cache tag* dan yang lainnya.

4. *Repsonse Body*, data/*resource* yang diberikan oleh server baik itu berupa *text*, *json* ataupun *xml*

D. Lumen

Lumen adalah *Micro Framework* yang diciptakan pengembang *Laravel* untuk mengakomodasi kebutuhan developer yang ingin membuat aplikasi dalam skala lebih kecil dari *Laravel*. Karena banyak *library* yang dihilangkan dalam *bundle source code*, *Lumen* bisa dijadikan *framework* untuk membuat *REST API*.

Fitur Unggulan

Lumen sudah *dibundle* dengan *Eloquent ORM* agar proses pengqueryan lebih mudah dan tidak memakan waktu karena kita tidak perlu lagi mengetik *query* yang panjang dan sederet fitur lain seperti:

1. *Caching*

   Dalam lamannya menyebutkan *Laravel* menyediakan *API* terpadu untuk berbagai sistem *caching*. Konfigurasi *cache* terletak di file .env. Dalam file ini Anda dapat menentukan *driver cache* mana yang ingin Anda gunakan secara default di seluruh aplikasi Anda. *Laravel* mendukung *backend caching* yang populer seperti *Memcached* dan *Redis* di luar kotak.

2. *Queues*

   Dalam lamannya menyebutkan Layanan antrian *Lumen* menyediakan *API* terpadu di berbagai ujung belakang antrian yang berbeda. Antrian memungkinkan Anda menunda

pemrosesan tugas yang memakan waktu, seperti melakukan tugas di server jauh, sampai waktu yang secara drastis mempercepat permintaan web ke aplikasi Anda.

3. *Validation*

Dalam lamannya menyebutkan *Lumen* menyediakan beberapa pendekatan berbeda untuk memvalidasi data masuk aplikasi Anda. Secara default, kelas pengendali dasar *Lumen* menggunakan fitur *ProvidesConvenienceMethods* yang menyediakan metode yang mudah digunakan untuk memvalidasi permintaan HTTP masuk dengan berbagai aturan validasi yang kuat.

4. *Routing*

Pengaturan terhadap URL yang terdiri dari beberapa *method* dan bisa menyematkan/mengambil parameter pada URL sesuai dengan aturan penulisannya.

5. *Middleware*

Dalam lamannya menyebutkan *Middleware* HTTP menyediakan mekanisme yang mudah digunakan untuk memfilter permintaan HTTP yang masuk ke aplikasi Anda. Misalnya, *Lumen* menyertakan *middleware* yang memverifikasi pengguna aplikasi Anda telah diautentikasi. Jika pengguna tidak diautentikasi, *middleware* akan mengalihkan pengguna ke layar masuk. Namun, jika pengguna diautentikasi, *middleware* akan mengizinkan permintaan untuk melangkah lebih jauh ke aplikasi.

E. JSON

JSON singkatan untuk *JavaScript Object Notation* adalah sebuah format untuk berbagi data. Seperti dapat kita lihat dari namanya, *JSON* diturunkan dari bahasa pemrograman J*avaScript*, akan tetapi format ini tersedia bagi banyak bahasa lain termasuk *Python, Ruby, PHP*, dan *Java*. *JSON* biasanya dilafalkan seperti nama "*Jason*."

JSON menggunakan ekstensi *.json* saat ia berdiri sendiri. Saat didefinisikan di dalam format file lain (seperti di dalam *.html*), ia dapat tampil didalam tanda petik sebagai *JSON* string, atau ia dapat dimasukkan kedalam sebuah variabel. Format ini sangat mudah untuk ditransfer antar server web dengan klien atau browser.

Karena sangat mudah dibaca dan ringan, *JSON* memberikan alternatif lebih baik dari *XML* dan membutuhkan *formatting* yang tidak banyak. Panduan ini akan membantu pembaca untuk memahami apa itu *JSON*, bagaimana menggunakan data di file *JSON*, serta struktur dan sintaks dari format ini.

Sintaks dan Struktur

Sebuah objek *JSON* adalah format data *key-value* yang biasanya di *render* di dalam kurung kurawal. Saat kita bekerja dengan *JSON*, kita akan sering melihat objek *JSON* disimpan di dalam sebuah

file *.json*, tapi mereka juga dapat disimpan sebagai objek *JSON* atau string di dalam sebuah program.

F. JWT

JSON Web Token atau lebih dikenal dengan *JWT* yang mana *JWT* ini adalah sebuah token berbentuk string panjang yang sangat random yang gunanya sendiri untuk melakukan sistem Autentikasi dan Pertukaran Informasi. Umumnya untuk melakukan login tidak seperti pada aplikasi website biasa dimana kita menggunakan *session* untuk mengingat siapa yang sedang Login. Tapi didalam *API* sendiri kita menggunakan konsep *JWT* atau dibacanya sebagai "jot" . Website resminya ada di *jwt.io*.

Cara Kerja *JWT*

Dimana *JWT* atau Token ini seperti password jadi ketika users berhasil melakukan Login maka server akan memberikan sebuah Token. Nanti Token tersebut akan disimpan oleh *users* pada *Local Storage* atau *Cookies Browser* dan bila users ingin mengakses halaman halaman tertentu maka harus menyertakan token tersebut. Untuk itu *users* akan mengirim balik token yang dikasih diawal tadi sebagai bukti bila *user* ini, sudah melakukan login. Sekarang kita akan lihat struktur dasarnya Tokennya dimana terdiri dari tiga bagian yaitu yang pertama *header* lalu kedua bagian *payload*-nya atau datanya dan yang ketiga adalah bagian *verify signature*.

Komponen *JWT*:

1. *Header* – Terdiri dari Algoritma HS256 yang kita gunakan dan tipenya jwt sebagai defaultnya.

2. *Payload* – Sebagai infomasi atau data yang ingin kita kirimkan untuk *users* misalnya _id *user*-nya atau tanggal *expired*-nya dan lain lain.
3. *Verify Signature* – hasil dari *Hash* atau gabungan dari isi *encode Header* dan *Payload*-nya lalu ditambahkan kode *secret*-nya.

G. Git

Git adalah *version control system* yang digunakan para developer untuk mengembangkan software secara bersama-bersama. Fungsi utama *git* yaitu mengatur versi dari *source code* program anda dengan mengasih tanda baris dan *code* mana yang ditambah atau diganti.

Git digunakan untuk memudahkan programmer untuk mengetahui perubahan *source code*-nya daripada harus membuat file baru seperti Program.java, ProgramRevisi.java, ProgramRevisi2.java, ProgramFix.java. Selain itu, dengan *git* kita tak perlu khawatir *code* yang kita kerjakan bentrok, karena setiap developer bias membuat *branch* sebagai *workspace*-nya. Fitur yang tak kalah keren lagi, pada *git* kita bisa memberi komentar pada *source code* yang telah ditambah/diubah, hal ini mempermudah developer lain untuk tahu kendala apa yang dialami developer lain.

Untuk mengetahui bagaimana menggunakan *git*, berikut perintah-perintah dasar *git*:

1. *Git init* : untuk membuat *repository* pada file lokal yang nantinya ada folder .git

2. *Git status* : untuk mengetahui status dari *repository* lokal
3. *Git add* : menambahkan file baru pada *repository* yang dipilih
4. *Git commit* : untuk menyimpan perubahan yang dilakukan, tetapi tidak ada perubahan pada *remote repository*.
5. *Git push* : untuk mengirimkan perubahan file setelah di commit ke *remote repository*.
6. *Git branch* : melihat seluruh *branch* yang ada pada *repository*
7. *Git checkout* : menukar *branch* yang aktif dengan *branch* yang dipilih
8. *Git merge* : untuk menggabungkan *branch* yang aktif dan *branch* yang dipilih
9. *Git clone* : membuat Salinan *repository* lokal

Contoh dari software *version control system* adalah *github, bitbucket, snowy evening*, dan masih banyak lagi. Jika anda sebagai developer belum mengetahui fitur git ini, maka anda wajib mencoba dan memakainya. Karena banyak manfaat yang akan didapat dengan git ini.

BAB III
PELAKSANAAN PI DAN PEMBAHASAN

1. Profil Perusahaan

1. Organisasi dan Manajemen Industri

a. Gambaran Umum Perusahaan

PT Kipa Teknologi Indonesia adalah Startup Digital yang terbentuk pada Kuartal ke 3 pada tahun 2019 sebagai anak perusahaan dari *Software House* PT Dedayu Opus Maxima. PT Kipa Teknologi Indonesia diresmikan pada tanggal 14 Februari 2020.

PT Kipa Teknologi Indonesia memiliki produk utamanya yakni KipaPOS. KipaPOS adalah sebuah aplikasi *Point of Sale* berbasis Web dan *Mobile* yang ditujukan untuk mengatasi masalah yang ada pada UMKM di Indonesia.

PT Kipa Teknologi Indonesia termasuk Startup Digital yang masih baru lahir dengan hanya memiliki karyawan sejumlah 13 pada masa pengembangan. Masa pengembangan KipaPOS sendiri dimulai sejak bulan Oktober pada tahun 2019 hingga awal Februari 2020 sebagai MVP (*Minimum Viable Product*) dari KipaPOS.

b. Logo PT Kipa Teknologi Indonesia

Berikut adalah logo dari PT Kipa Teknologi Indonesia.

Gambar 3. 1 Logo PT Kipa Teknologi Indonesia.

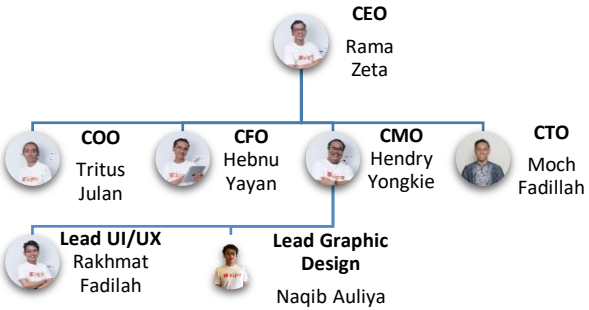
c. Visi dan Misi PT Kipa Teknologi Indonesia

1) **Visi**
Visi dari Dinas Komunikasi Dan Informatika Pemerintah Provinsi Jawa Timur adalah "Membantu UMKM di indonesia untuk naik level dalam mengembangkan usahanya"

2) **Misi**
   a. Mengembangkan startup point of sale untuk membantu UMKM bertahan di era digital.
   b. Memberikan edukasi kepada UMKM tentang teknologi point of sale dalam mengelola usaha.
   c. Memberikan solusi atas permasalahan yang terjadi di UMKM dengan teknologi terbaru.

d. Struktur Organisasi Perusahaan

Struktur organisasi menunjukkan suatu hubungan yang saling terkait antara satu dengan yang lain, dalam setiap bagian mempunyai wewenang dan tanggung jawab masing – masing. Berikut adalah struktur organisasi dalam PT Kipa Teknologi Indonesia.



Gambar 3. 2 Organization Chart PT Kipa Teknologi Indonesia

2. Deskripsi Pelaksanaan Kegiatan PI

a. Tempat dan Waktu Pelaksanaan

Dalam pelaksanaan PI mahasiswa ditempatkan di bagian Pengembangan Produk Startup Digital. Adapun deskripsi pelaksanaan kegiatan PI adalah sebagai berikut.

- **Nama Instansi:**
  PT Kipa Teknologi Indonesia

- **Alamat Instansi** :
  Jl. Simpang Sulfat Selatan Kav 1 No 4 Malang, Jawa Timur Indonesia

- **Kontak Instansi :**
  Telp. : 082268888859
  Web : kipa.id
  E-mail : hai@kipapos.com

Sedangkan, waktu pelaksanaan kegiatan praktik industrinya adalah sebagai berikut.

- Hari Kerja : Senin - Jumat
- Jam Kerja : Pukul 09:00 - 19:00 WIB

Kegiatan PI ini berlangsung selama kurang lebih 1 bulan.

b. Kegiatan Praktik Industri

Tabel 3. 1 Jadwal Kegiatan Praktik Industri

No	Tanggal	Uraian Kegiatan
1	Kamis, 2 Januari 2020	- Perkenalan lingkungan kerja - Pembagian Bidang Pekerjaan
2	Jumat, 3 Januari 2020	- Pemberian Akses *Repository* - Penjelasan Standar Kerja Tim dengan menggunakan *Git*
3	Sabtu, 4 Januari 2020	LIBUR
4	Minggu, 5 Januari 2020	LIBUR
5	Senin, 6 Januari 2020	- Mempelajari Konsep dan Algoritma POS - Mempelajari Konsep MVP yang sudah ditetapkan oleh KipaPOS - Mencari referensi dari software POS yang ada di Indonesia
6	Selasa, 7 Januari 2020	- Mencari referensi *request* dan *response API* - Merancang standart *request* dan *response API*
7	Rabu, 8 Januari 2020	- Membuat *API Authentication User* menggunakan *JWT* sebagai *authenticator*-nya - Membuat *API Util* untuk list Provinsi dan Kota seluruh Indonesia

8	Kamis, 9 Januari 2020	- Merancang *API CRUD Profile* - Merancang *API CRUD* Bisnis *Setting*
9	Jumat, 10 Januari 2020	- Merancang *API CRUD* Merek / *Brands*
10	Sabtu, 11 Januari 2020	LIBUR
11	Minggu, 12 Januari 2020	LIBUR
12	Senin, 13 Januari 2020	- Merancang *API CRUD* Satuan / *Unit*
13	Selasa, 14 Januari 2020	- Merancang *API CRUD* Kategori dan Sub Kategori
14	Rabu, 15 Januari 2020	- Merancang *API CRUD* Supplier - Merancang *API CRUD* Pelanggan
15	Kamis, 16 Januari 2020	- Merancang *API CRUD* Karyawan / *Employee*
16	Jumat, 17 Januari 2020	- Merancang *API CRUD* Grup Pelanggan
17	Sabtu, 18 Januari 2020	LIBUR

18	Minggu, 19 Januari 2020	LIBUR
19	Senin, 20 Januari 2020	- Merancang *API CRUD* Grup Harga Jual
20	Selasa, 21 Januari 2020	- Merancang *API CRUD* Outlet
21	Rabu, 22 Januari 2020	- Merancang *API CRUD* Biaya / Pengeluaran
22	Kamis, 23 Januari 2020	- Merancang *API CRUD Invoice Layout*
23	Jumat, 24 Januari 2020	- Merancang *API CRUD* Akun Keuangan
24	Sabtu, 25 Januari 2020	LIBUR
25	Minggu, 26 Januari 2020	LIBUR
26	Senin, 27 Januari 2020	- Merancang *API CRUD* Produk - Merancang *API CRUD Opening Stock*
27	Selasa, 28 Januari 2020	- Merancang *API* Kasir Penjualan Produk Fisik - Merancang *API* Retur Penjualan

28	Rabu, 29 Januari 2020	- Merancang *API* Kasir Penjualan Produk Digital Prabayar
29	Kamis, 30 Januari 2020	- Merancang *API* Kasir Penjualan Produk Digital Pascabayar - Mengintegrasikan *Callback* Server Pulsa ke sistem KipaPOS
30	Jumat, 31 Januari 2020	- Merancang *API* Laporan - *Final Testing* Aplikasi - *Live Bug Fixing*
31	Sabtu, 1 Februari 2020	- Pemberian Kenang Kenangan - Permintaan Nilai Praktik Perpisahan

3. Faktor – Faktor Pendukung dan Penghambat

a. Faktor Pendukung

Selama melaksanakan Praktik Industri di PT Kipa Teknologi Indonesia, ada beberapa faktor pendukung yang membantu kelancaran dalam melakukan kegiatan praktik sebagai berikut:

1) Ruangan yang bersih dan cukup nyaman untuk melakukan pekerjaan
2) Terdapat mess/penginapan untuk karyawan
3) Adanya kerjasama yang baik dengan pembimbing praktik industri
4) Karyawan dan staf di PT Kipa Teknologi Indonesia sangat ramah dan terbuka kepada mahasiswa Praktik Industri sehingga menciptakan suasana yang nyaman

b. Faktor Penghambat

Selama melaksanakan Praktik Industri di PT Kipa Teknologi Indonesia, ada beberapa faktor

yang menghambat pelaksanaan Praktik Industri, diantaranya adalah sebagai berikut:

1) Terbatasnya waktu yang dipergunakan dalam menempuh Praktik Industri sehingga pengetahuan yang diperoleh masih kurang
2) Kurangnya wawasan tentang *Point of Sale* yang mengakibatkan beberapa kendala ketika proses pengembangan aplikasi
3) Kampus dengan tempat Praktik Industri berbeda kota sehingga dibutuhkan waktu 2 jam untuk menempuh perjalanan
4) Karena keterbatasan waktu sehingga membutuhkan ekstra *effort* untuk menyelesaikan *task* yang diberikan oleh pembimbing hingga terkadang harus lembur

4. Pembahasan

1. Analisis Sistem

a. Identifikasi Masalah

Berdasarkan latar belakang yang telah diuraikan sebelumnya maka permasalahan yang dapat dirumuskan, yaitu bagaimana merancang aplikasi Web *API* KipaPOS (*Point of Sale*) untuk memenuhi MVP (*Most Viable Product*) perusahaan dan memenuhi kebutuhan pasar.

b. Analisa Kebutuhan

Berikut adalah data dan aplikasi yang dibutuhkan dalam perancangan aplikasi Web *API* KipaPOS.

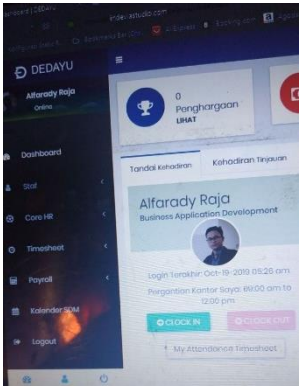
1) *Database* KipaPOS Web
2) *Framework Lumen-Laravel*
3) *VSCode*
4) *Insomnia*

c. Prosedur Kerja Perusahaan

PT Kipa Teknologi Indonesia memiliki prosedur kerja yang sudah ditetapkan dari awal masa pengembangan.

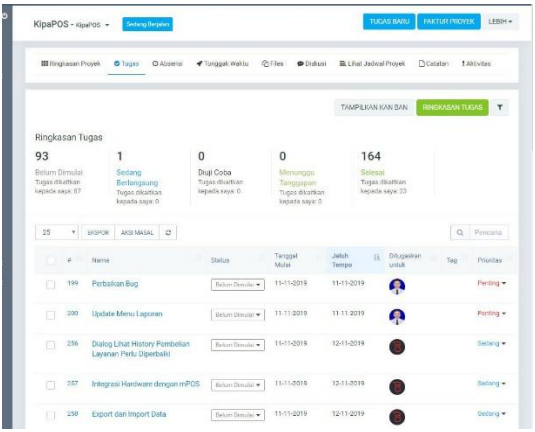
a. Software Management Project

PT Kipa Teknologi Indonesia awalnya memiliki sebuah software sendiri bernama "Projects" (Gambar 3.3, 3.4, 3.5) untuk mengatur *task* para karyawan sebelum berubah menggunakan *Jira Software* (Gambar 3.6, 3.7).



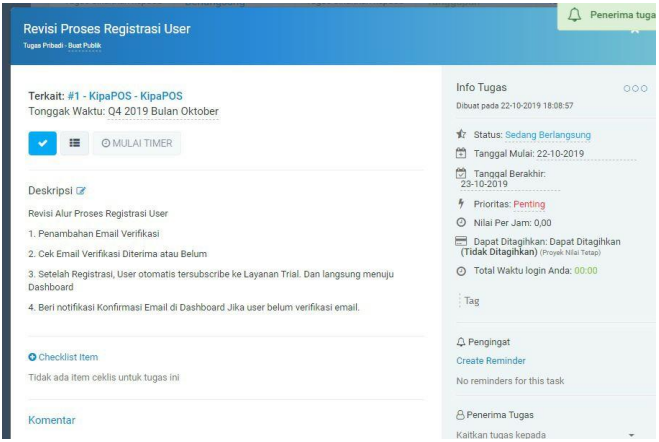
Gambar 3. 3 Tampilan awal Projects.

Tampilan awal saat login ke website Projects milik PT Kipa Teknologi Indonesia sebelum menggunakan *Jira Software.*



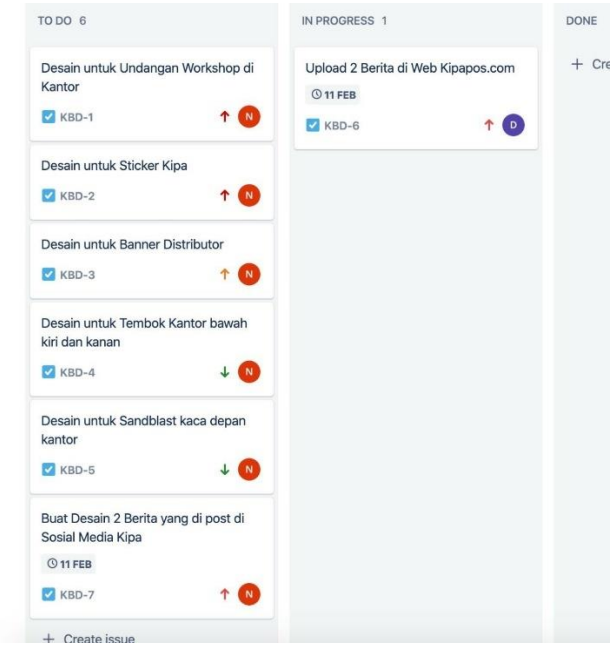
Gambar 3. 4 Tampilan Daftar Task Projects.

Task dari *lead* akan ditampilkan seperti gambar 3.4 dan bisa juga dibuat tampilan Kanban. Ketika mulai pengerjaan, mahasiswa wajib merubah status Belum Mulai jadi Sedang Berlangsung.



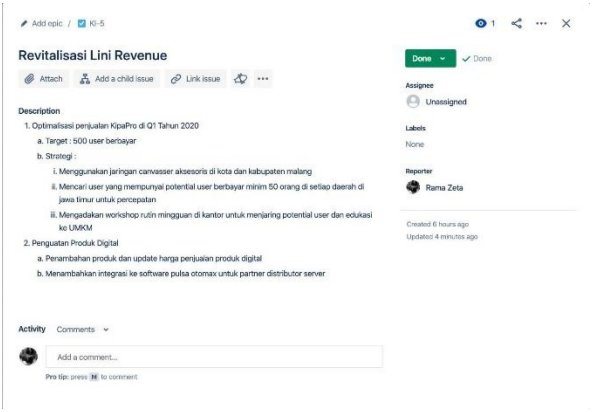
Gambar 3. 5 Tampilan Detail Task Projects.

Detail *task* akan berisikan Status, Tgl Mulai, Tgl Berakhir, Prioritas, dan ada Deskripsi.



Gambar 3. 6 Tampilan Daftar Task Jira.

Tampilan ini (Gambar 3.6) adalah daftar *task* mahasiswa ketika menggunakan *Jira Software*. Berbentuk Kanban secara default dan untuk mengganti statusnya kita dapat langsung *Drag and Drop*.



Gambar 3. 7 Tampilan Task Jira.

Detail *task* pad *Jira Software* berisikan kode, judul, dan deskripsi. Kode tersebut dapat terhubung dengan *git commit message*.

b. Alur Penyelesaian Permasalahan

PT Kipa Teknologi Indonesia memiliki cara dalam menyelesaikan sebuah permasalahan untuk dijadikan sebuah *task* hingga diimplementasikan pada produk.

1) Permasalahan akan dibahas oleh para *Chief*
2) CTO akan membuat *task* untuk diimplementasikan nantinya dan diserahkan kepada *Lead*
3) *Lead* akan membagikan *task* kepada masing masing *stakeholder*
4) *Team UI/UX* akan membuat desain dan alur yang nantinya akan divalidasi oleh *Lead UI/UX*
5) Setelah desain valid, *Team UI/UX* menyerahkan desain ke team *Mobile Developer*

6) *Mobile Developer* akan menentukan data apa saja yang diperlukan dan memintanya ke bagian Web *API Developer*
7) Web *API Developer* akan menyelesaikan tugasnya dan divalidasi oleh *Lead Software Engineer* / CTO lalu akan diserahkan Kembali ke *Mobile Developer*
8) *Mobile Developer* akan menyelesaikan tugasnya dan akan divalidasi juga oleh *Lead Software Engineer* / CTO

2. *Software Testing Level*

PT Kipa Teknologi Indonesia menerapkan *Software Testing* secara terstruktur dan ada tingkatannya sebagai berikut:

1) Unit Testing

Laravel memiliki teknologi untuk melakukan *Unit Testing* dan dijalankan oleh *developer.*

2) Integration Testing

Tahap ini dilakukan oleh *developer* sendiri dengan manual.

3) System Testing

Tahap ini dilakukan oleh *Lead Software Engineer*/CTO.

4) Acceptance Testing

Tahap ini memiliki 2 tahap lagi, yang pertama dilakukan oleh karyawan yang tidak campur tangan dengan proses pengembangan, kedua dilakukan oleh pengguna.

BAB IV
KESIMPULAN DAN SARAN

A. Kesimpulan

Setelah melakukan Praktik Industri di PT Kipa Teknologi Indonesia yang dilaksanakan pada 2 Januari sampai 2 Februari 2020, mahasiswa mendapatkan banyak ilmu dan pengetahuan yang sebelumnya tidak pernah didapatkan waktu perkuliahan.

Di tempat industri mahasiswa mendapatkan tugas sebagai Web Developer yang bertanggung jawab pada Web *API*. Web *API* nantinya akan di implementasikan oleh pihak *Mobile Developer* untuk dijadikan aplikasi *mobile*.

B. Saran

Aplikasi ini telah berhasil dibuat untuk memenuhi MVP (*Minimum Viable Product*) dari perusahaan, tentu saja aplikasi ini masih banyak kekurangan untuk menyelesaikan masalah yang ada di UMKM Indonesia. Proses Pengembangan Startup Digital tidak akan pernah berhenti seiring kebutuhan pasar, tidak menutup kemungkinan setelah mahasiswa selesai melakukan Praktik Industri aplikasi ini sudah banyak sekali pengembangan terbaru. Alangkah baiknya proses pengembangan ini diiringi oleh proses pemasaran.

DAFTAR PUSTAKA

Candra, Y. 2019. *Mengapa Masih Banyak UMKM Indonesia yang Belum "Go Digital"?*. https://ekonomi.kompas.com/read/2019/02/12/152246426/mengapa-masih-banyak-umkm-indonesia-yang-belum-go-digital (diakses tanggal 17 Mei 2020).

GoBiz., 2019. *Apa itu Point of Sale?*. https://gobiz.co.id/pusat-pengetahuan/apa-itu-point-of-sale-pos/ (diakses tanggal 8 Februari 2020).

Kiddy., 2019. *Apa itu Restful API?*. https://medium.com/@kiddy.xyz/restful-api-apaan-tuh-dbcfa434761e (diakses tanggal 8 Februari 2020).

Fauzi, Rizky., 2017. *Pengenalan Lumen Framework, Micro Framework Berbasis PHP*. https://www.codepolitan.com/pengenalan-lumen-framework-micro-framework-berbasis-php-59f19fe6ea010 (diakses tanggal 8 Februari 2020).

Santoso, Bagus Aji., 2017. *Mengenal Format JSON*. https://www.codepolitan.com/mengenal-format-json-59e8152dd0e51 (diakses tanggal 8 Februari 2020).

Salma, Alfiana Irsyada., 2017. *Mengenal Konsep JSON Web Token (JWT)*. https://www.dumetschool.com/blog/mengenal-konsep-json-web-token-jwt (diakses tanggal 8 Februari 2020).

IdCloudhost., 2016. *Pengertian dan Manfaat GIT bagi Developer.* https://idcloudhost.com/pengertian-dan-manfaat-git-bagi-developer/ (diakses tanggal 8 Februari 2020).

LAMPIRAN

Lampiran 1. Rekapitulasi Kegiatan PI

Rekapitulasi Kegiatan
Praktik Industri

Nama : Alfarady Raja Ghanie Hamid Jauhar
Prodi : SI – Pend. Teknologi Informasi
Tempat Praktik : PT Kipa Teknologi Indonesia
Bidang Pekerjaan : Web Developer
Catatan Kegiatan : Minggu ke I/II/III/IV

No	Tanggal	Uraian Kegiatan	Paraf Pembimbing Praktik Industri
1	Kamis, 2 Januari 2020	- Perkenalan lingkungan kerja - Pembagian Bidang Pekerjaan	
2	Jumat, 3 Januari 2020	- Pemberian Akses Repository - Penjelasan Standar Kerja Tim dengan menggunakan Git	
3	Sabtu, 4 Januari 2020	LIBUR	
4	Minggu, 5 Januari 2020	LIBUR	
5	Senin, 6 Januari 2020	- Mempelajari Konsep dan Algoritma POS - Mempelajari Konsep MVP yang sudah ditetapkan oleh KipaPOS - Mencari referensi dari software POS yang ada di Indonesia	
6	Selasa, 7 Januari 2020	- Mencari referensi request dan response API - Merancang standart request dan response API	
7	Rabu, 8 Januari 2020	- Membuat API Authentication User menggunakan JWT sebagai authenticator-nya - Membuat API Util untuk list Provinsi dan Kota seluruh Indonesia	
8	Kamis, 9 Januari 2020	- Merancang API CRUD Profile - Merancang API CRUD Bisnis Setting	

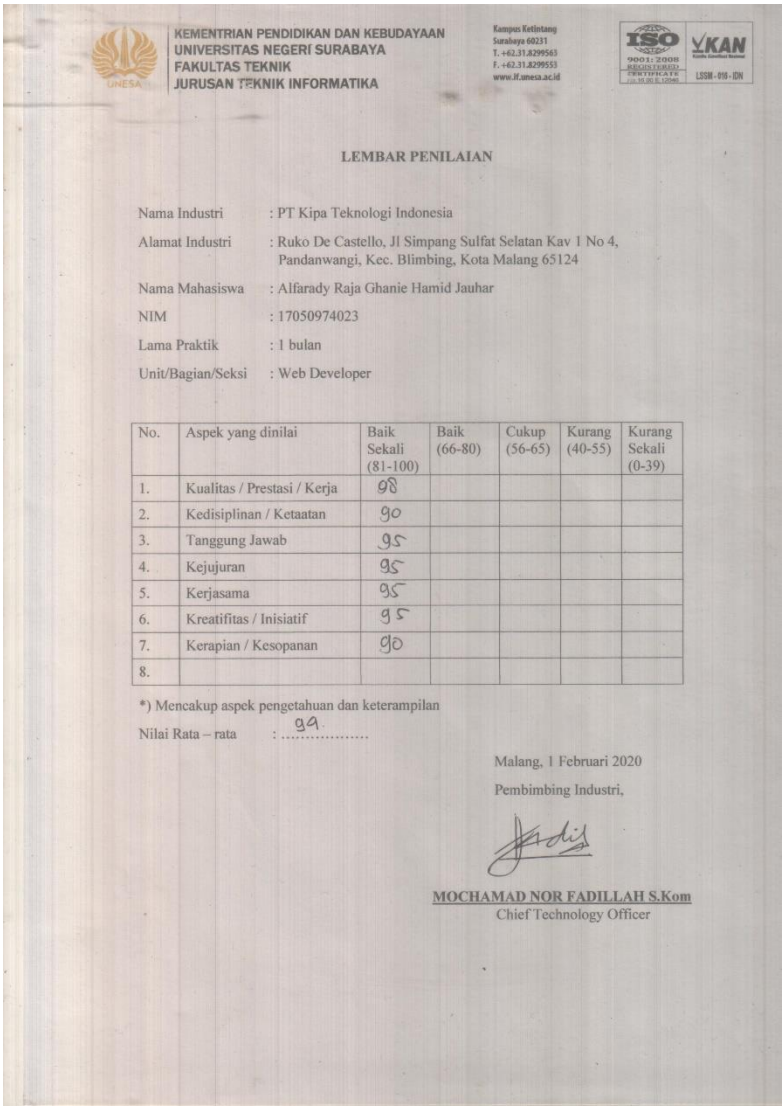
9	Jumat, 10 Januari 2020	- Merancang API CRUD Merek / Brands	
10	Sabtu, 11 Januari 2020	LIBUR	
11	Minggu, 12 Januari 2020	LIBUR	
12	Senin, 13 Januari 2020	- Merancang API CRUD Satuan / Unit	
13	Selasa, 14 Januari 2020	- Merancang API CRUD Kategori dan Sub Kategori	
14	Rabu, 15 Januari 2020	- Merancang API CRUD Supplier - Merancang API CRUD Pelanggan	
15	Kamis, 16 Januari 2020	- Merancang API CRUD Karyawan / Employee	
16	Jumat, 17 Januari 2020	- Merancang API CRUD Grup Pelanggan	
17	Sabtu, 18 Januari 2020	LIBUR	
18	Minggu, 19 Januari 2020	LIBUR	
19	Senin, 20 Januari 2020	- Merancang API CRUD Grup Harga Jual	
20	Selasa, 21 Januari 2020	- Merancang API CRUD Outlet	
21	Rabu, 22 Januari 2020	- Merancang API CRUD Biaya / Pengeluaran	
22	Kamis, 23 Januari 2020	- Merancang API CRUD Invoice Layout	

23	Jumat, 24 Januari 2020	- Merancang API CRUD Akun Keuangan	
24	Sabtu, 25 Januari 2020	LIBUR	
25	Minggu, 26 Januari 2020	LIBUR	
26	Senin, 27 Januari 2020	- Merancang API CRUD Produk - Merancang API CRUD Opening Stok	
27	Selasa, 28 Januari 2020	- Merancang API Kasir Penjualan Produk Fisik - Merancang API Retur Penjualan	
28	Rabu, 29 Januari 2020	- Merancang API Kasir Penjualan Produk Digital Prabayar	
29	Kamis, 30 Januari 2020	- Merancang API Kasir Penjualan Produk Digital Pascabayar - Mengintegrasikan Callback Server Pulsa ke sistem KipaPOS	
30	Jumat, 31 Januari 2020	- Merancang API Laporan - Final Testing Aplikasi - Live Bug Fixing	
31	Sabtu, 1 Februari 2020	- Pemberian Kenang Kenangan - Permintaan Nilai Praktik Perpisahan	

Pembimbing Industri,

MOCHAMAD NOR FADILLAH S.Kom
Chief Technology Officer

Lampiran 2. Format NPI (Nilai Praktik Industri)



KEMENTRIAN PENDIDIKAN DAN KEBUDAYAAN
UNIVERSITAS NEGERI SURABAYA
FAKULTAS TEKNIK
JURUSAN TEKNIK INFORMATIKA

Kampus Ketintang
Surabaya 60231
T. +62.31.8299563
F. +62.31.8299553
www.if.unesa.ac.id

ISO 9001:2008 REGISTERED CERTIFICATE
KAN
LSSN - 016 - IDN

LEMBAR PENILAIAN

Nama Industri : PT Kipa Teknologi Indonesia

Alamat Industri : Ruko De Castello, Jl Simpang Sulfat Selatan Kav 1 No 4, Pandanwangi, Kec. Blimbing, Kota Malang 65124

Nama Mahasiswa : Alfarady Raja Ghanie Hamid Jauhar

NIM : 17050974023

Lama Praktik : 1 bulan

Unit/Bagian/Seksi : Web Developer

No.	Aspek yang dinilai	Baik Sekali (81-100)	Baik (66-80)	Cukup (56-65)	Kurang (40-55)	Kurang Sekali (0-39)
1.	Kualitas / Prestasi / Kerja	98				
2.	Kedisiplinan / Ketaatan	90				
3.	Tanggung Jawab	95				
4.	Kejujuran	95				
5.	Kerjasama	95				
6.	Kreatifitas / Inisiatif	95				
7.	Kerapian / Kesopanan	90				
8.						

*) Mencakup aspek pengetahuan dan keterampilan

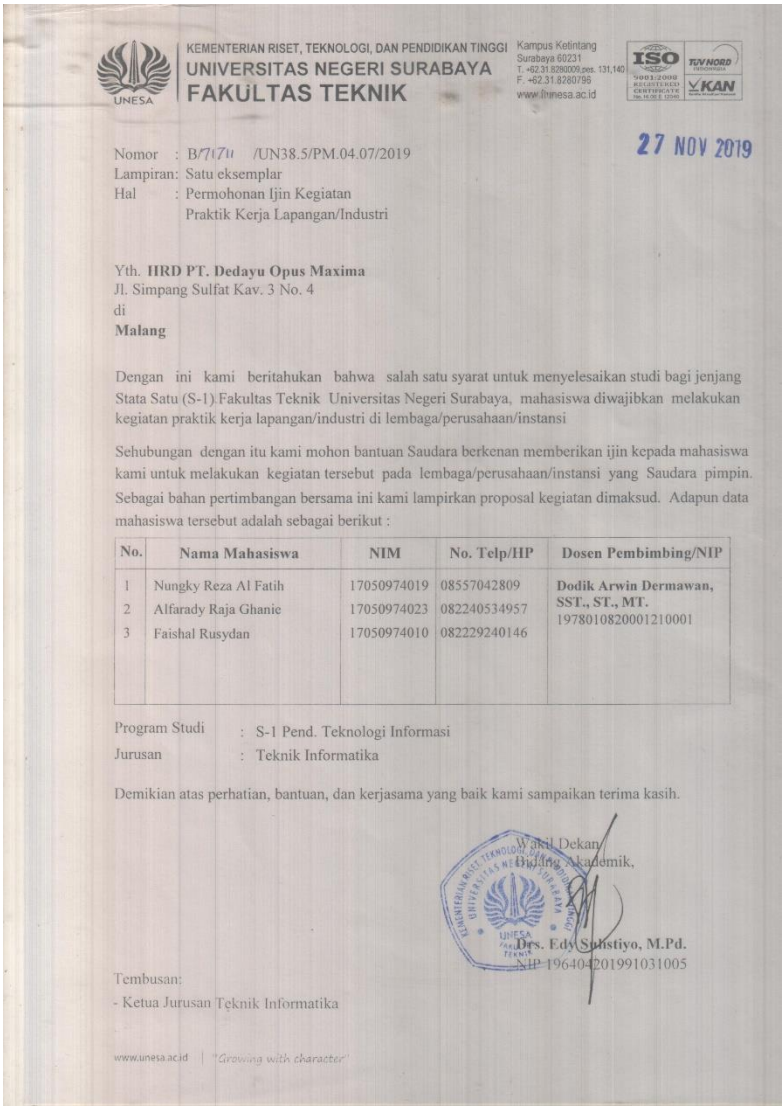
Nilai Rata – rata : 94

Malang, 1 Februari 2020

Pembimbing Industri,

MOCHAMAD NOR FADILLAH S.Kom
Chief Technology Officer

Lampiran 3. Surat Permohonan Izin Praktik Industri



KEMENTERIAN RISET, TEKNOLOGI, DAN PENDIDIKAN TINGGI
UNIVERSITAS NEGERI SURABAYA
FAKULTAS TEKNIK

Kampus Ketintang
Surabaya 60231
T. +62.31.8280009,pes. 131,140
F. +62.31.8280796
www.ftunesa.ac.id

27 NOV 2019

Nomor : B/7171/UN38.5/PM.04.07/2019
Lampiran: Satu eksemplar
Hal : Permohonan Ijin Kegiatan Praktik Kerja Lapangan/Industri

Yth. **HRD PT. Dedayu Opus Maxima**
Jl. Simpang Sulfat Kav. 3 No. 4
di
Malang

Dengan ini kami beritahukan bahwa salah satu syarat untuk menyelesaikan studi bagi jenjang Stata Satu (S-1) Fakultas Teknik Universitas Negeri Surabaya, mahasiswa diwajibkan melakukan kegiatan praktik kerja lapangan/industri di lembaga/perusahaan/instansi

Sehubungan dengan itu kami mohon bantuan Saudara berkenan memberikan ijin kepada mahasiswa kami untuk melakukan kegiatan tersebut pada lembaga/perusahaan/instansi yang Saudara pimpin. Sebagai bahan pertimbangan bersama ini kami lampirkan proposal kegiatan dimaksud. Adapun data mahasiswa tersebut adalah sebagai berikut :

No.	Nama Mahasiswa	NIM	No. Telp/HP	Dosen Pembimbing/NIP
1	Nungky Reza Al Fatih	17050974019	08557042809	**Dodik Arwin Dermawan, SST., ST., MT.** 197801082000121000 1
2	Alfarady Raja Ghanie	17050974023	082240534957	
3	Faishal Rusydan	17050974010	082229240146	

Program Studi : S-1 Pend. Teknologi Informasi
Jurusan : Teknik Informatika

Demikian atas perhatian, bantuan, dan kerjasama yang baik kami sampaikan terima kasih.

Wakil Dekan
Bidang Akademik,

Drs. Edy Sulistiyo, M.Pd.
NIP 196404201991031005

Tembusan:
- Ketua Jurusan Teknik Informatika

www.unesa.ac.id | "Growing with character"

Lampiran 4. Surat Balasan

Malang, 22 Mei 2020

Nomor : 026/SKet/HRD/V/2020
Lampiran : -
Perihal : Permohonan Praktek Kerja Lapanagan (PKL)

Kepada
Yth. : Bapak Dekan Fakultas Teknik
Universitas Negeri Surabaya

Sehubungan dengan Surat Permohonan Praktek Kerja Lapangan (PKL) Nomor : B/71711/UN38.5/PM.04.07/2019 yang diajukan kepada kami oleh mahasiswa bapak atas nama :

1. Nama : Nungky Reza Al Fatih
   NIM : 17050974019
   Jurusan/Program Studi : Teknik Informatika Universitas Negeri Surabaya

2. Nama : Faishal Rusydan
   NIM : 17050974010
   Jurusan/Program Studi : Teknik Informatika Universitas Negeri Surabaya

3. Nama : Alfarady Raja Ghanie Hamid Jauhar
   NIM : 17050974023
   Jurusan/Program Studi : Teknik Informatika Universitas Negeri Surabaya

Dengan ini kami memberikan ijin kepada ketiga mahasiswa tersebut diatas untuk melakukan kegiatan Praktek Kerja Lapangan (PKL) dan kegiatan-kegiatan lain yang berhubungan dengan kegiatan tersebut diatas.

Demikian Surat Balasan ini dibuat untuk dipergunakan sebagaimana mestinya.

CEO PT. Kipa Teknologi Indonesia

Chandra Ramazeta P.

Lampiran 5. Format NLP (Nilai Laporan PI)

Lampiran 6. Lembar Konsultasi Bimbingan PI

DEPARTEMEN PENDIDIKAN NASIONAL
UNIVERSITAS NEGERI SURABAYA
FAKULTAS TEKNIK
Kampus Ketintang
Jl.Ketintang, Surabaya 60231
Telp (031) 8280009 psw 500,510
Fax (031) 8280796
www.ftunesa.ac.id

LEMBAR KONSULTASI BIMBINGAN PRAKTIK INDUSTRI

Nama	: Alfarady Raja Ghanie Hamid Jauhar
NIM	: 17050974023
Program Studi	: S1 Pend. Teknologi Informasi
Jurusan/Angkatan	: Teknik Informatika / 2017
Fakultas	: Teknik
Judul Laporan	: PENGEMBANGAN SOFTWARE WEB API PADA STARTUP DIGITAL KIPAPOS (POINT OF SALE)
Pembimbing	: Dodik Arwin Dermawan, SST., ST., MT.

No.	Hari/Tanggal	BAB	Catatan Bimbingan

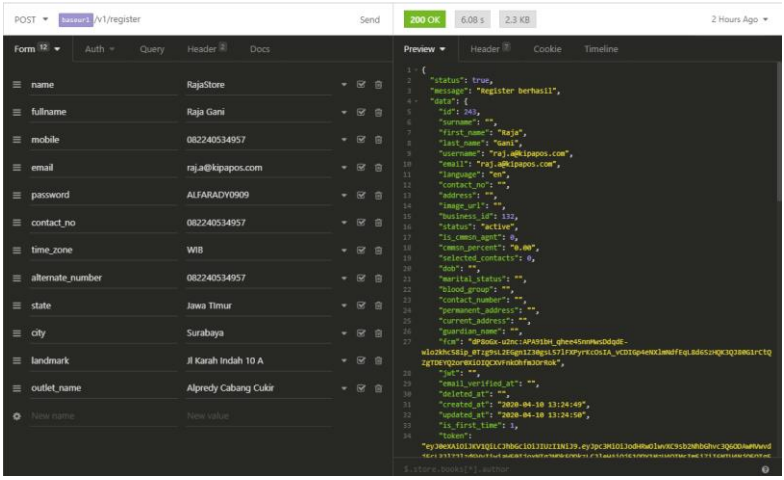
Surabaya, 11 April 2020

Pembimbing PI

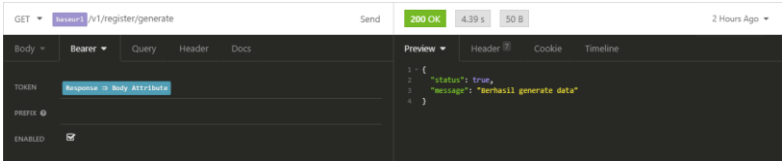
Lampiran 7. Dokumentasi Kegiatan PI

Lampiran 8. Dokumentasi API

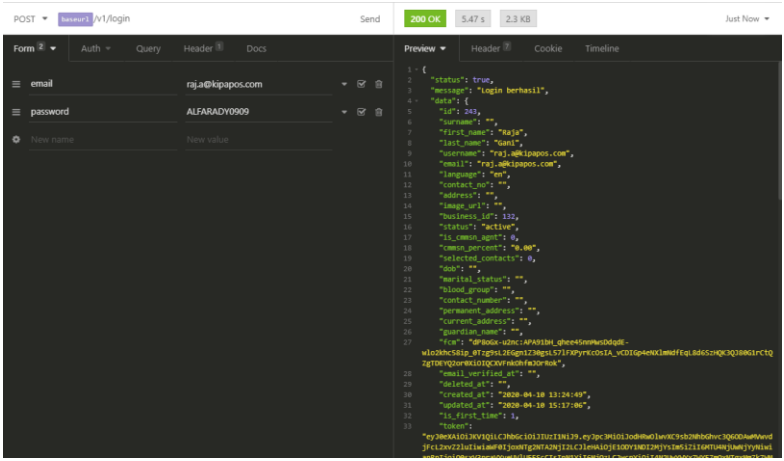
1. *API Authentication*
   a. *Register*



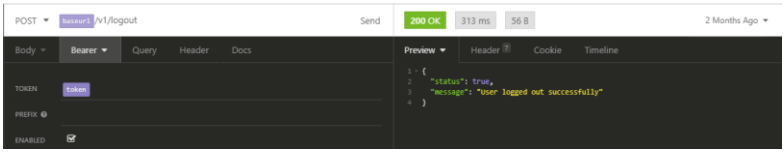
   b. *Generate Data*



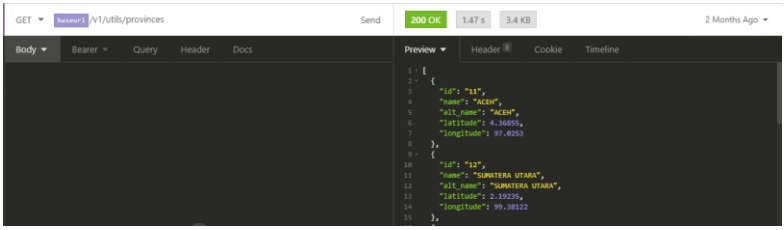
   c. *Login*



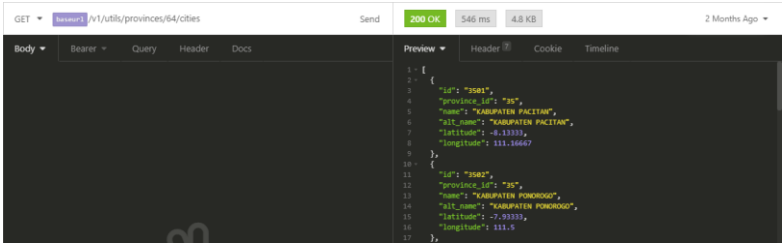
d. Logout



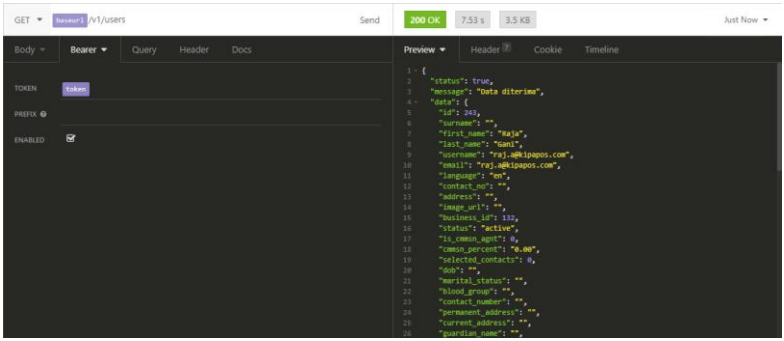
2. *API Utility*
   a. Provinsi



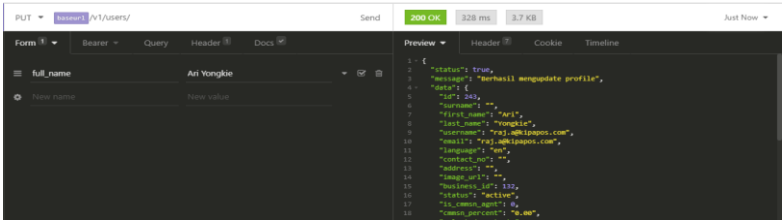
b. Kota



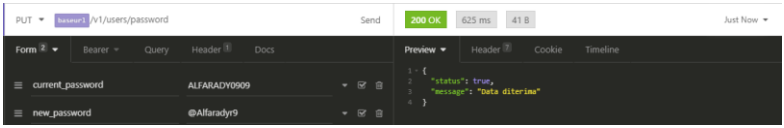
3. *API Profile*
   a. *Profile*



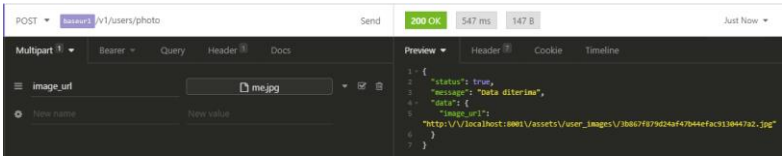
   b. *Update Profile*



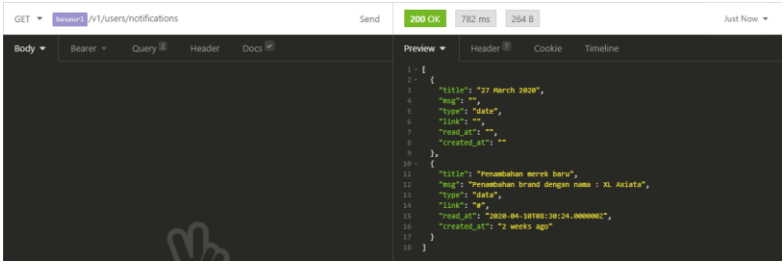
   c. *Update Password*



d. Update Photo Profile

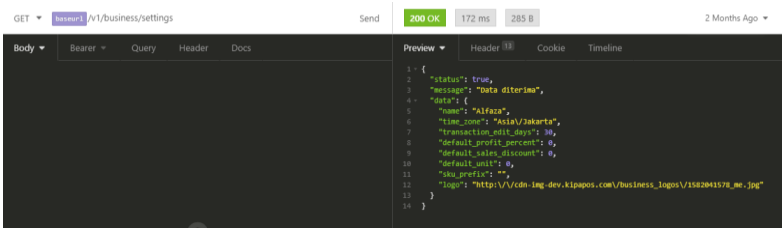


e. Notification

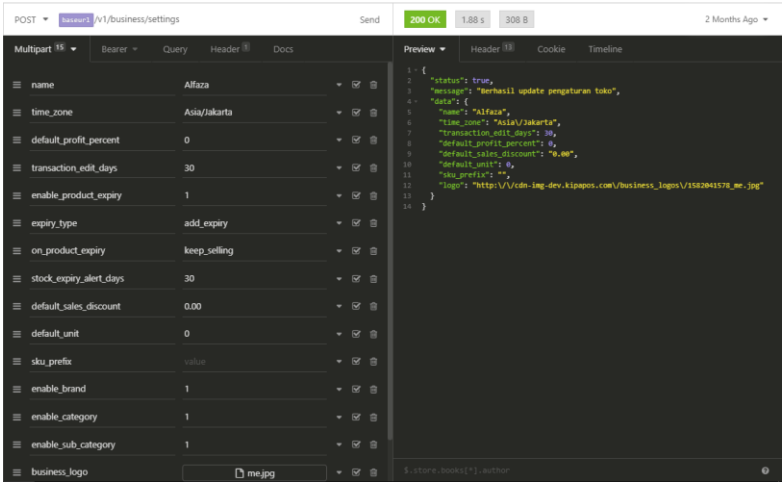


4. API Business Settings

a. Get Current Settings

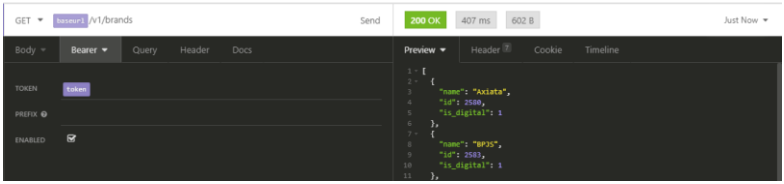


b. Update Settings

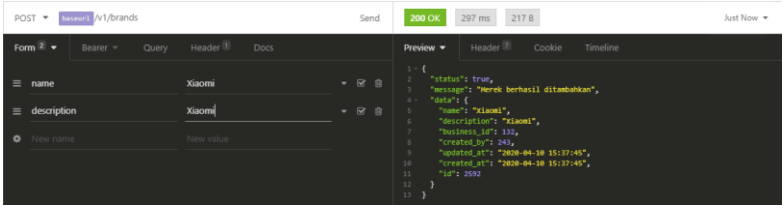


5. *API* Merek

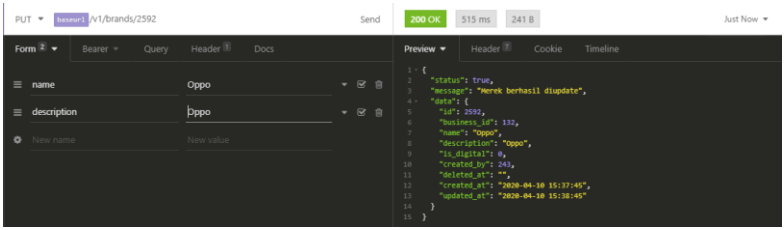
   a. *Get* Merek



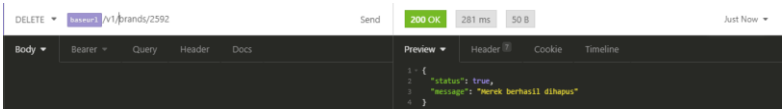
   b. *Create* Merek



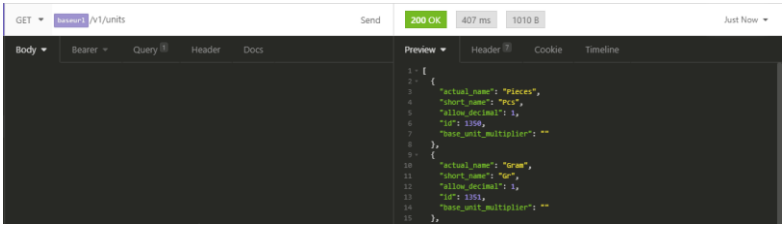
   c. *Update* Merek



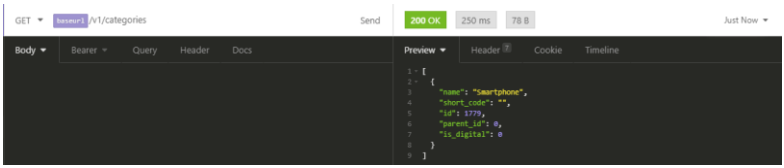
d. Delete Merek



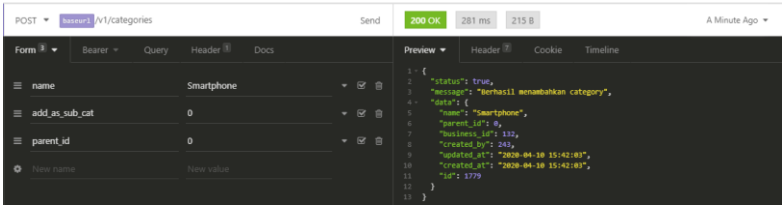
6. *API* Satuan
   a. *Get* Satuan



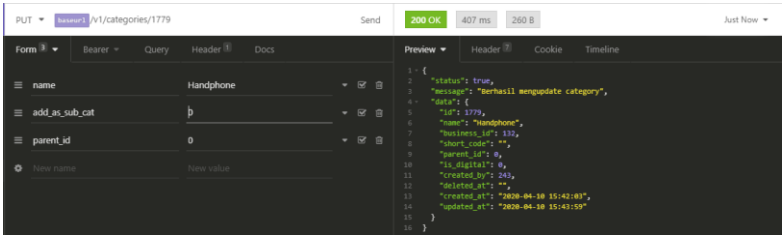
7. *API* Kategori & *Sub* Kategori
   a. *Get* Kategori & *Sub* Kategori



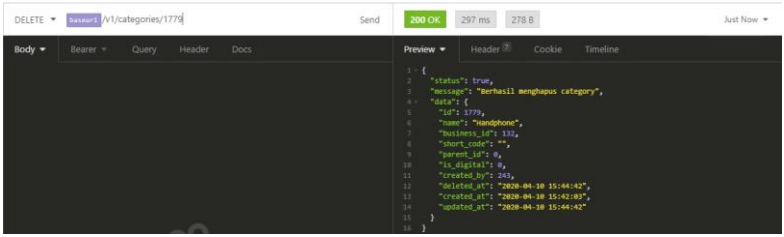
   b. *Create* Kategori & *Sub* Kategori



c. Update Kategori & *Sub* Kategori

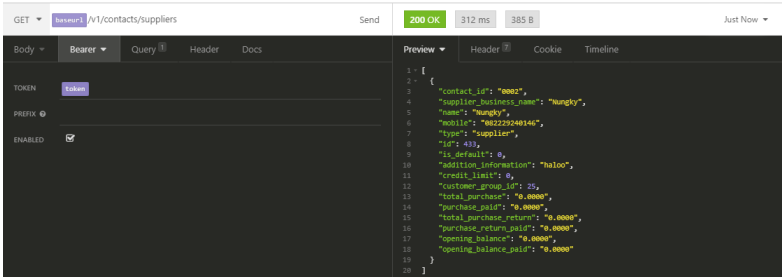


d. Delete Kategori & *Sub* Kategori

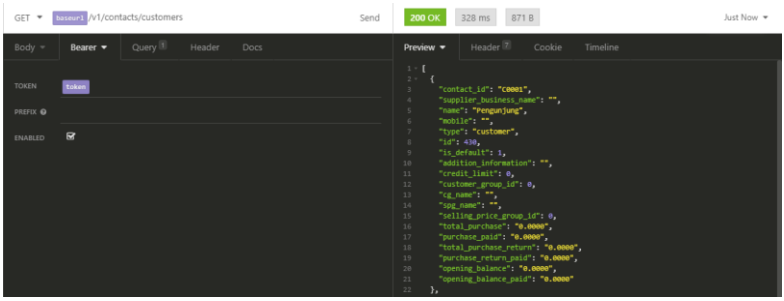


8. API Contacts

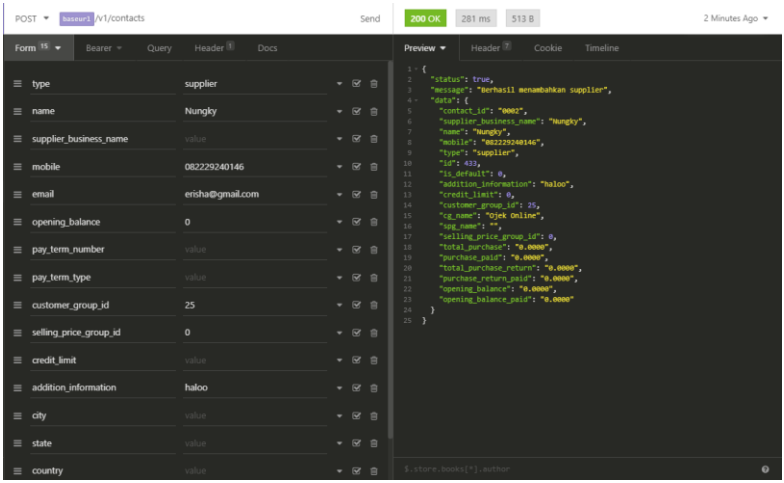
a. Get Supplier



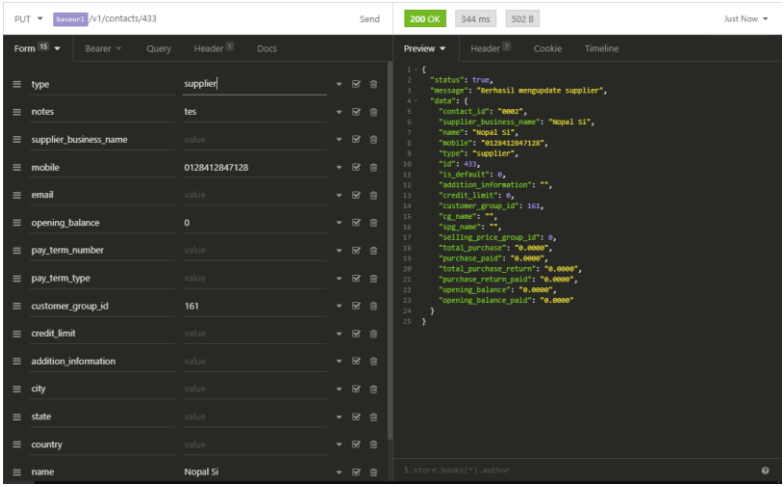
b. Get Customer



c. Create Contact (Supplier/Customer)



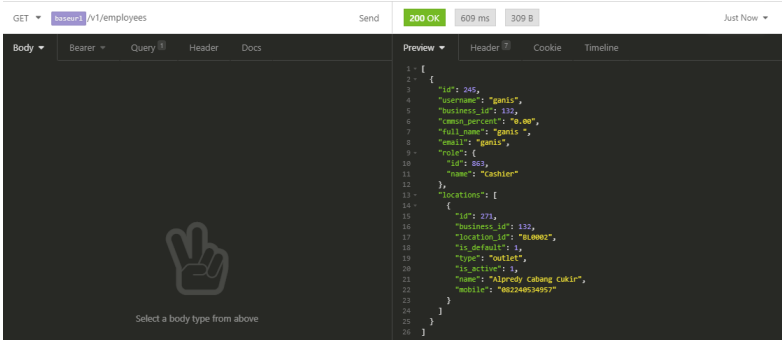
d. Update Contact (Supplier/Customer)



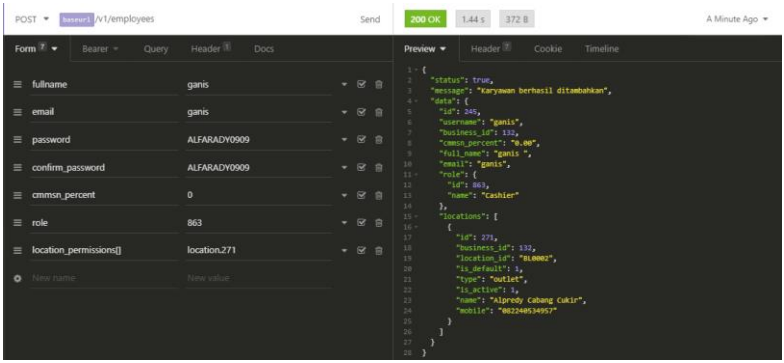
e. Delete Contact (Supplier/Customer)

9. API Employee

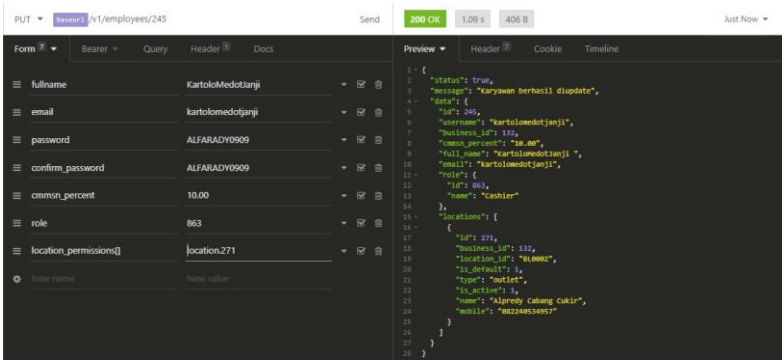
a. Get Employee



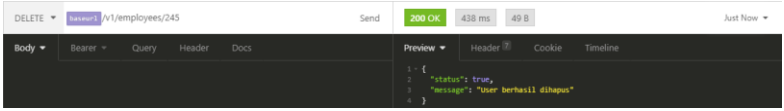
b. Create Employee



c. Update Employee

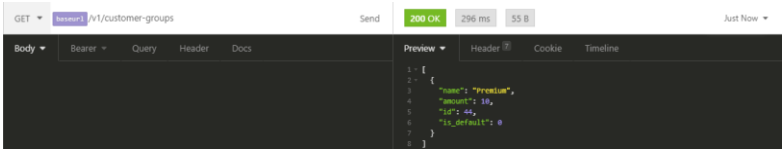


d. Delete Employee

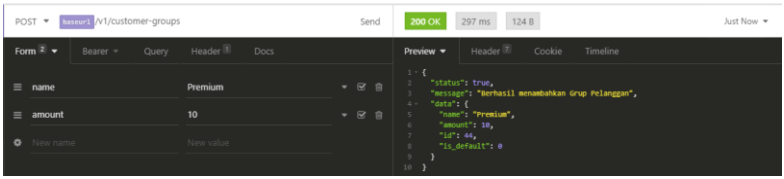


10. API Customer Group

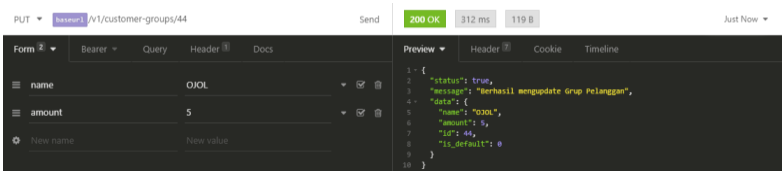
a. Get Customer Group



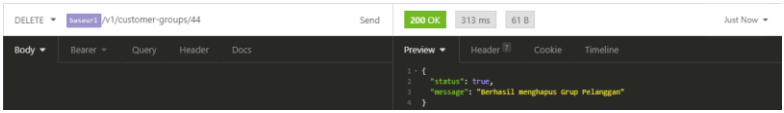
b. *Create Customer Group*



c. *Update Customer Group*

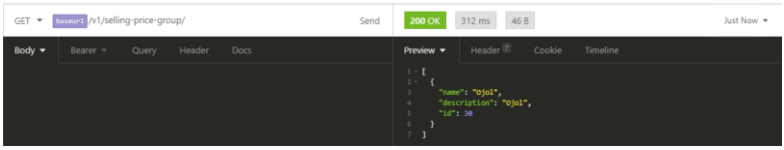


d. *Delete Customer Group*

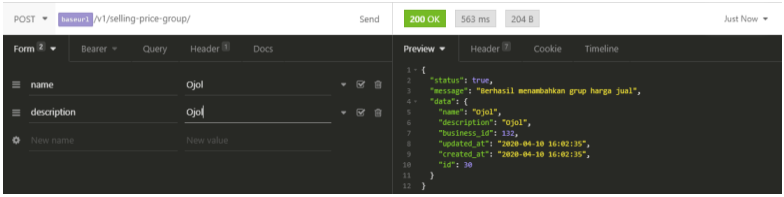


11. *API* Grup Harga Jual

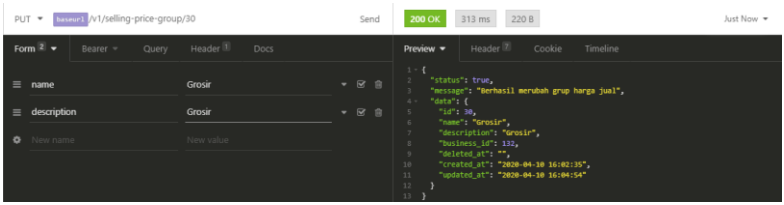
a. *Get* Grup Harga Jual



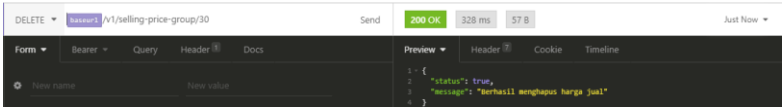
b. *Create* Grup Harga Jual



c. *Update* Grup Harga Jual

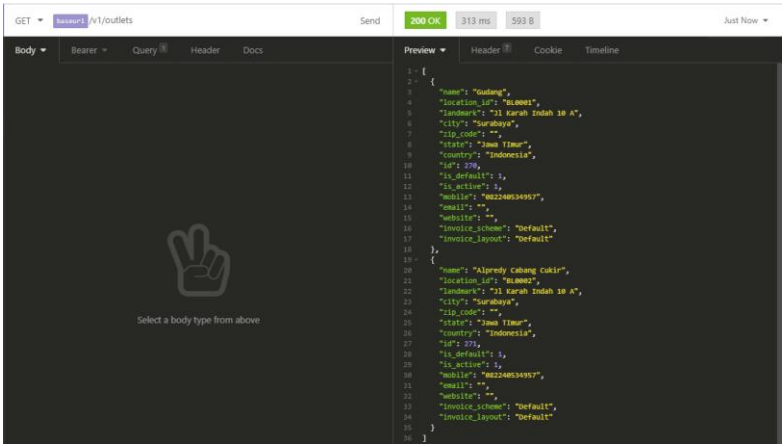


d. *Delete* Grup Harga Jual

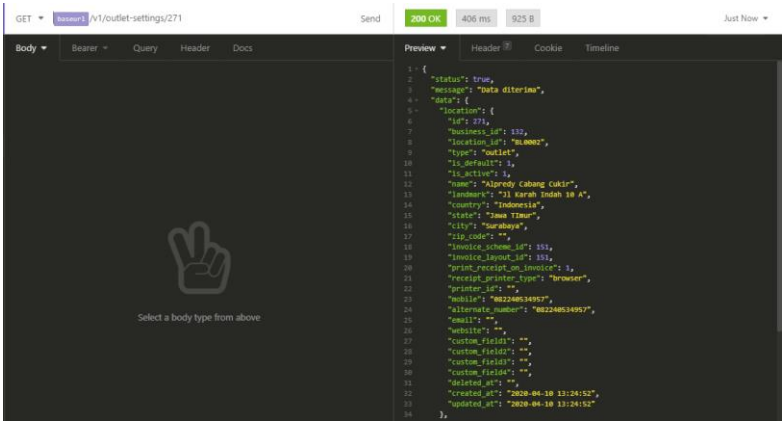


12. *API* Outlet

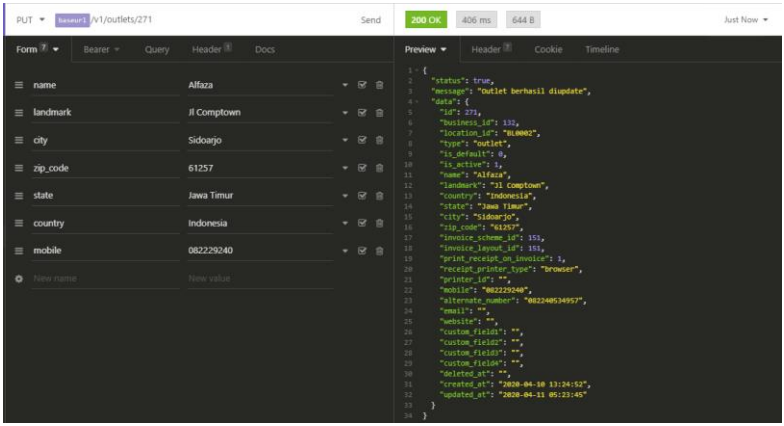
a. *Get* Outlet



b. *Get* Outlet *Settings*

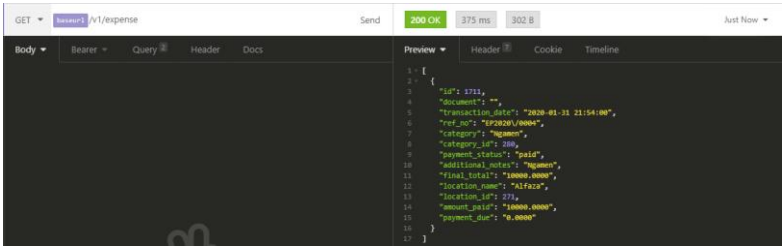


c. *Update* Outlet

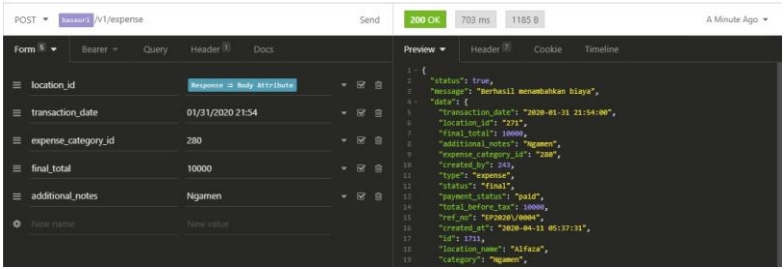


13. API Biaya/Pengeluaran

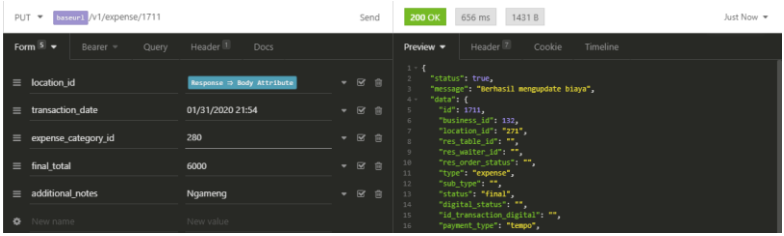
a. *Get* Biaya



b. *Create* Biaya



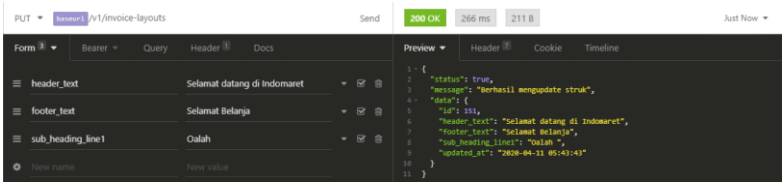
c. *Update* Biaya



d. *Delete* Biaya

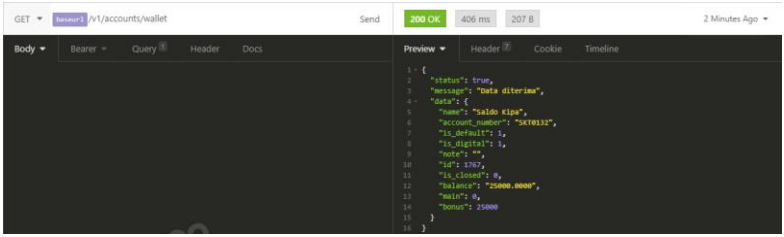
14. API Invoice Layout

a. *Update Invoice Layout*

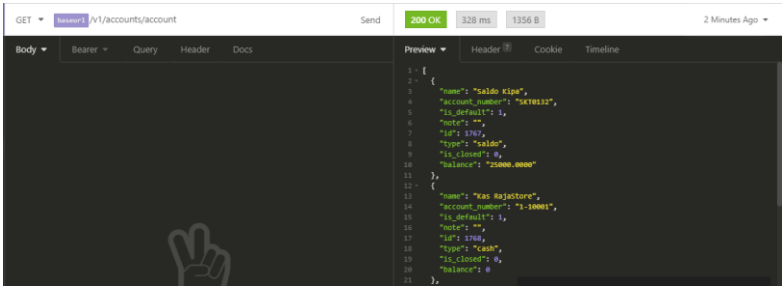


15. API Akun Keuangan

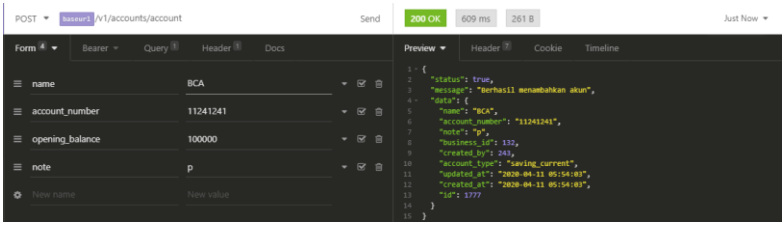
a. *Get* Saldo Kipa



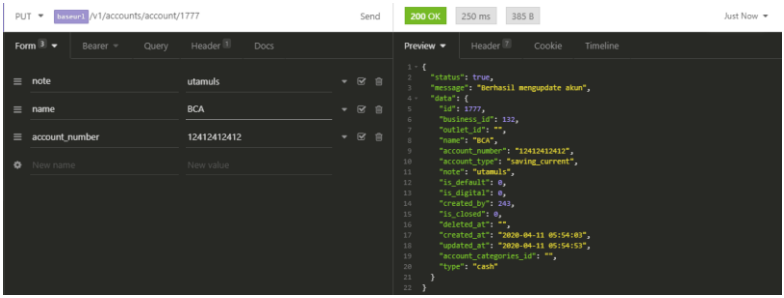
b. *Get* Akun Keuangan



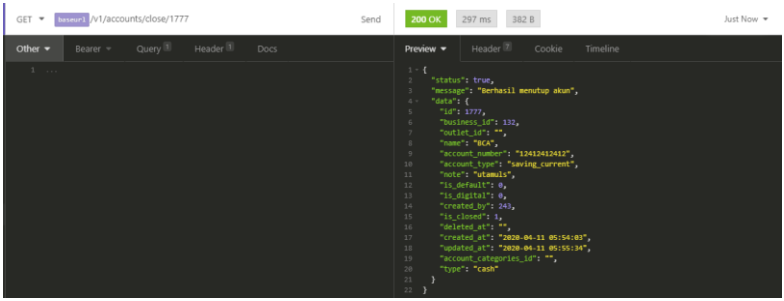
c. *Create* Akun Keuangan



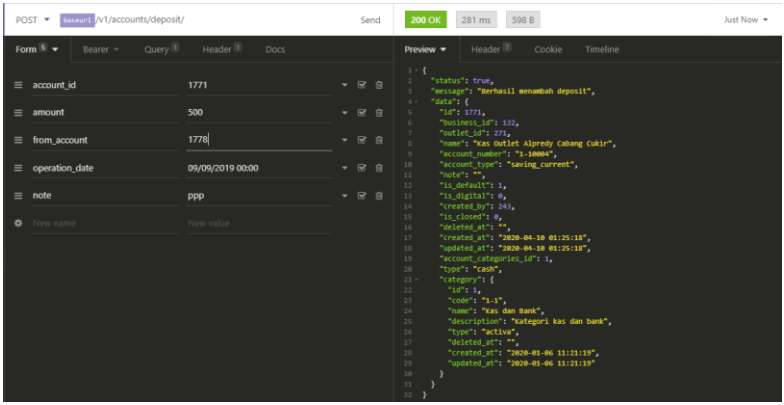
d. *Update* Akun Keuangan



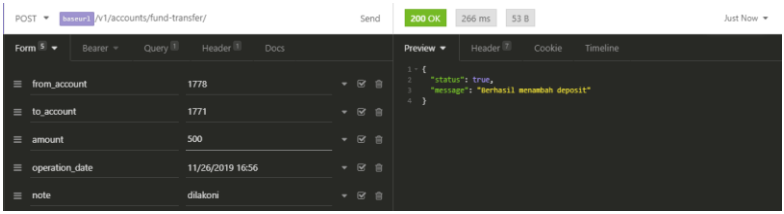
e. *Close* Akun Keuangan



f. *Deposit* Akun Keuangan

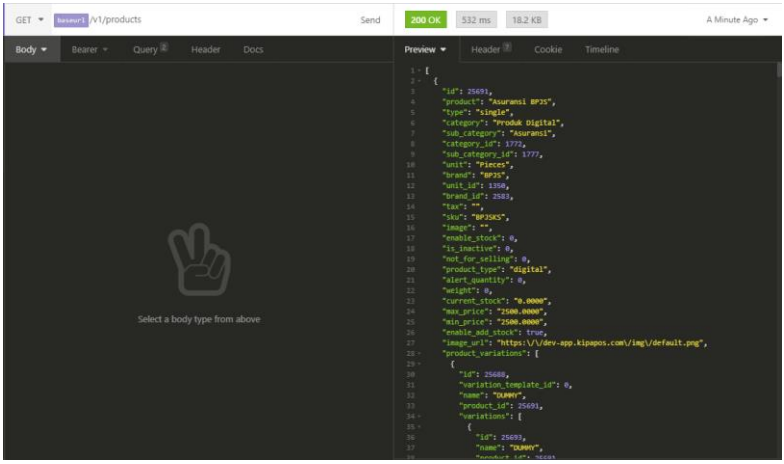


g. *Transfer* Akun Keuangan

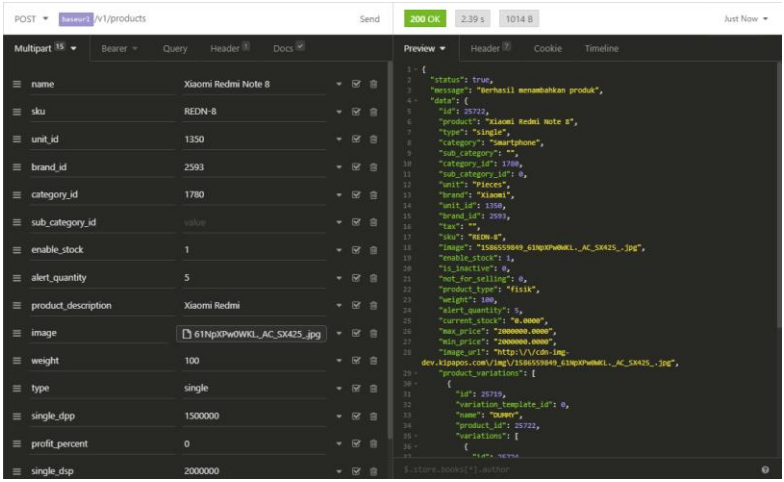


16. API Produk

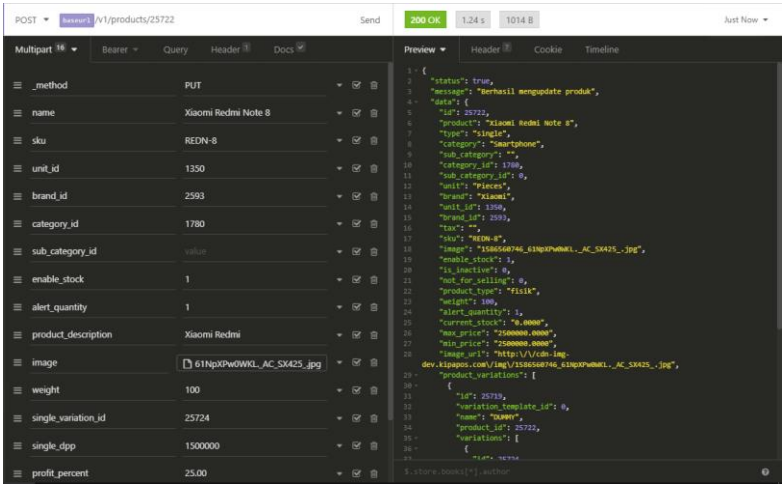
a. *Get* Produk



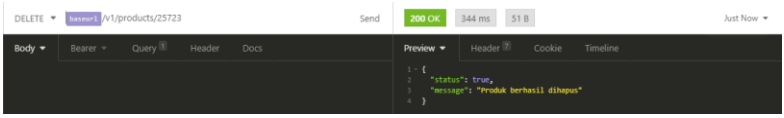
b. *Create* Produk



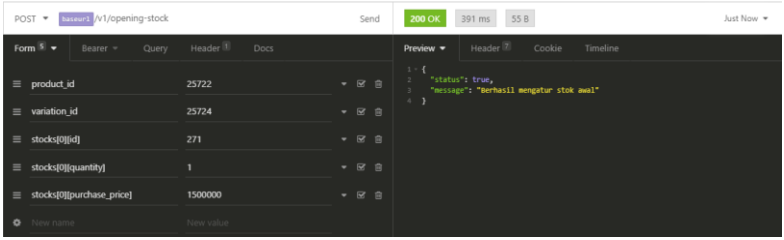
c. *Update* Produk



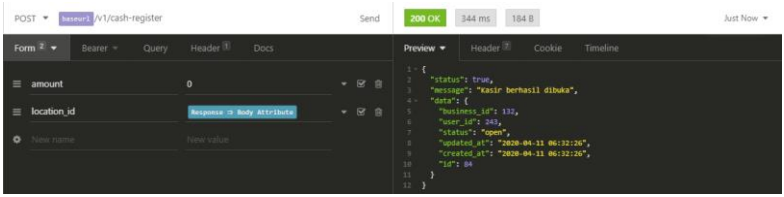
d. *Update* Produk



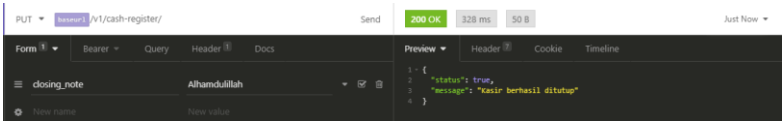
17. *API Opening Stock*
    a. *Create Opening Stock*



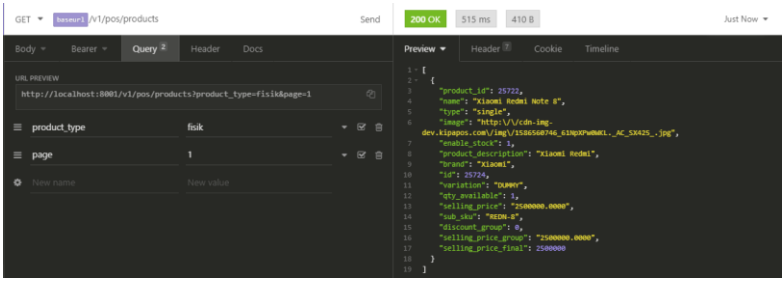
18. *API Cash Register*
    a. *Open Cash Register*



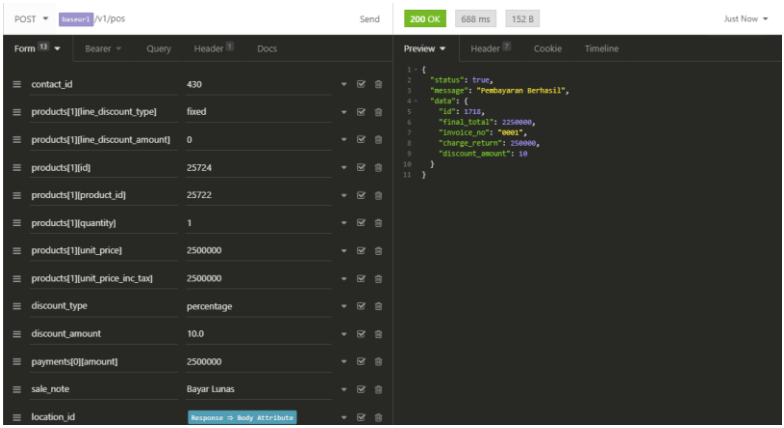
    b. *Close Cash Register*



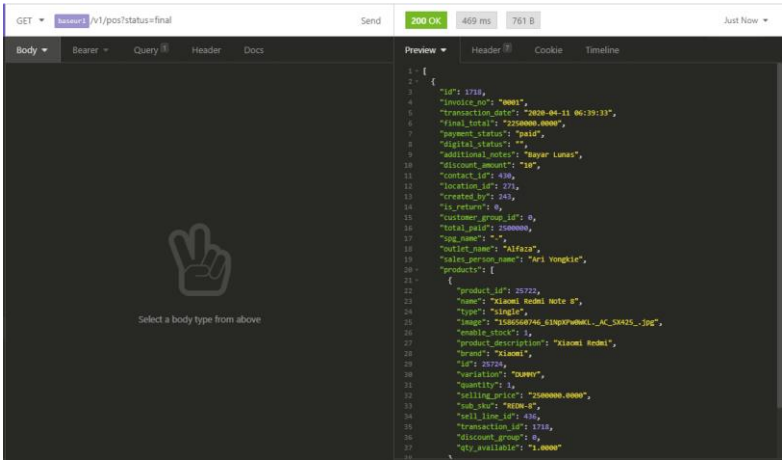
19. *API* Kasir Penjualan Produk Fisik
    a. *Get Product* Fisik



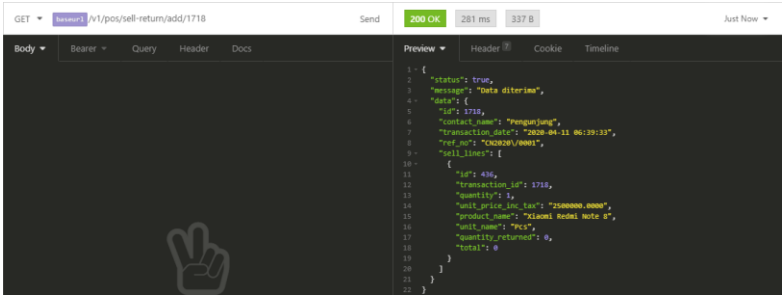
b. Penjualan Produk Fisik



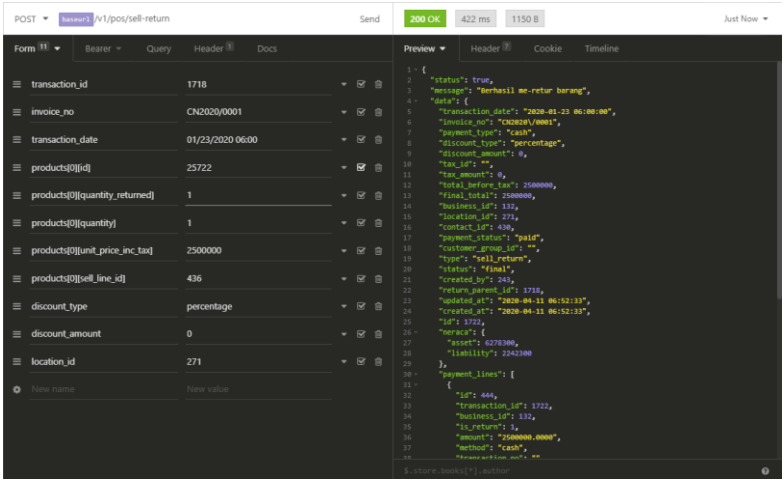
c. Detail Kasir



d. Retur *Form*

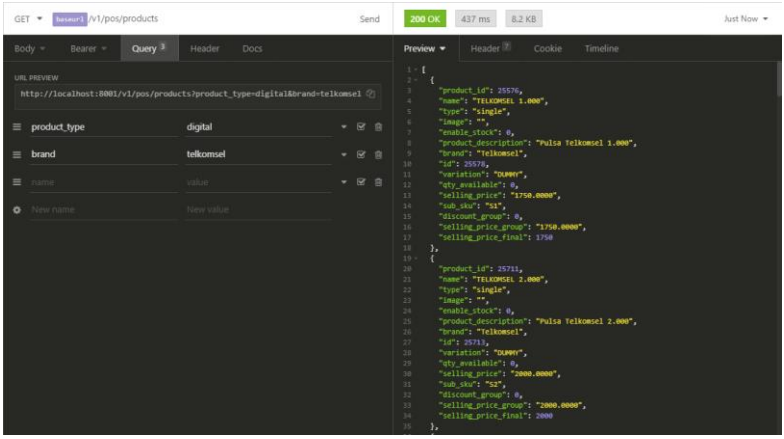


e. Tambah Retur Penjualan

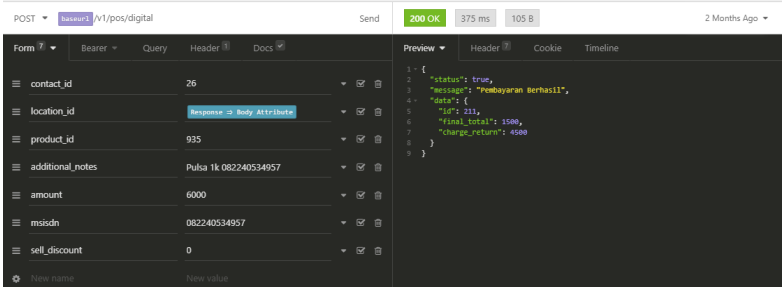


20. API Kasir Penjualan Produk Digital

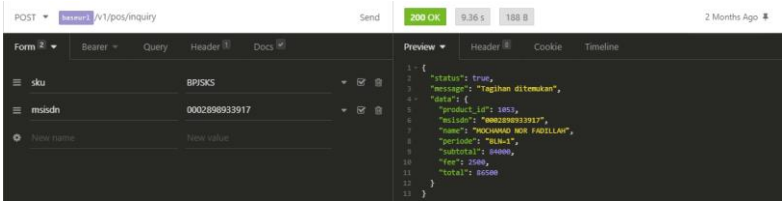
a. *Get Product* Digital



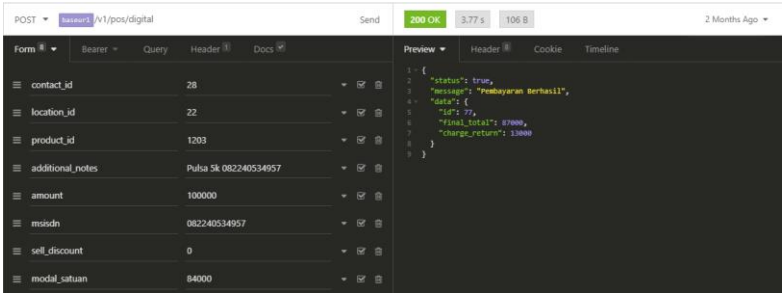
b. Penjualan Digital Prabayar



c. *Inquiry* Digital Pascabayar

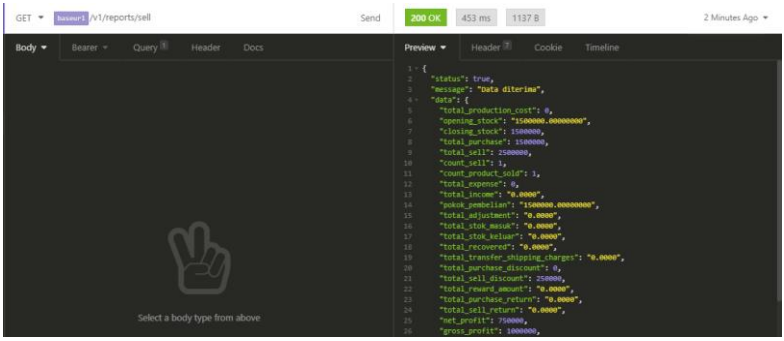


d. Penjualan Digital Pascabayar

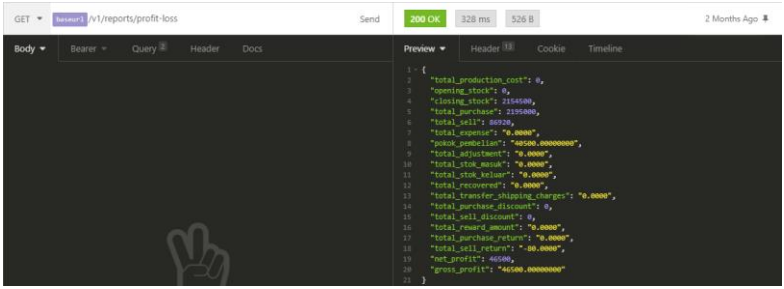


21. API Laporan

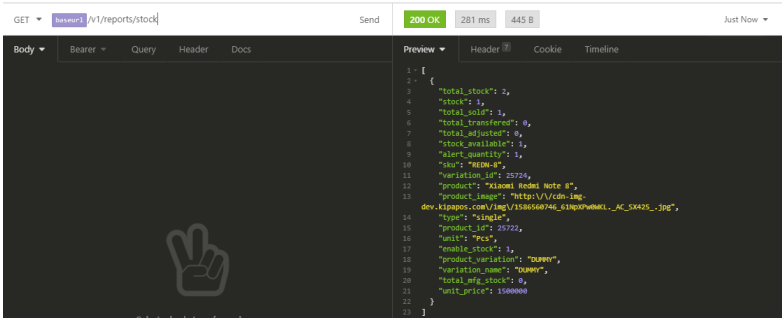
a. Penjualan



b. Laba Rugi



c. Stok Barang


GET
baseurl
/v1/reports/stock
Send
200 OK
281 ms
445 B
Just Now
Body
Bearer
Query
Header
Docs
Preview
Header 7
Cookie
Timeline
[
{
"total_stock": 2,
"stock": 1,
"total_sold": 1,
"total_transfered": 0,
"total_adjusted": 0,
"stock_available": 1,
"alert_quantity": 1,
"sku": "REDM-8",
"variation_id": 25724,
"product": "Xiaomi Redmi Note 8",
"product_image": "http:\/\/cdn-img-dev.kipapos.com\/img\/v1586560746_61NpXPwNWKL._AC_SX425_.jpg",
"type": "single",
"product_id": 25722,
"unit": "Pcs",
"enable_stock": 1,
"product_variation": "DUMMY",
"variation_name": "DUMMY",
"total_mfg_stock": 0,
"unit_price": 1500000
}
]

Lampiran 9. Dokumentasi *Source Code*

a. *Auth Controller*

```php
<?php

namespace App\Http\Controllers;

use App\Account;
use App\AccountTransaction;
use App\Business;
use App\Currency;
use App\System;
use App\TaxRate;
use App\Unit;
use App\User;
use App\Utils\BusinessUtil;
use App\Utils\GenerateDataUtil as GenerateDataUtil;
use App\Utils\DigitalUtil as DigitalUtil;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\DB;
use Carbon\Carbon;
use Illuminate\Auth\Passwords\PasswordBrokerManager;
use Illuminate\Support\Facades\Password;

use Spatie\Permission\Models\Permission;
use Spatie\Permission\Models\Role;

class AuthController extends Controller
{
    /**
     * All Utils instance.
     *
     */
    protected $businessUtil;

    /**
     * Constructor
     *
     * @param ProductUtils $product
     * @return void
     */
    public function __construct(BusinessUtil $businessUtil)
    {
        $this->businessUtil = $businessUtil;
    }

    /**
     * Get a JWT via given credentials.
     *
     * @param  Request  $request
     * @return Response
     */
    public function login(Request $request)
    {
          //validate incoming request
        $isValid = $this->validator($request, [
            'email' => 'required|string',
            'password' => 'required|string',
        ]);
        if(!$isValid) return $this->respondFailed("Data tidak valid", 400);

        $credentials = $request->only(['email', 'password']);

        Auth::factory()->setTTL(600); // 10 H
        if (! $token = Auth::attempt($credentials)) {
            return $this->respondFailed("Email atau password salah", 401);
        }

        $user = \App\User::where('email', $request->input('email'))->first()->makeHidden([
            'custom_field_1',
            'custom_field_2',
            'custom_field_3',
            'custom_field_4',
            'bank_details',
            'id_proof_name',
```

```
            'id_proof_number',
            'fb_link',
            'twitter_link',
            'social_media_1',
            'social_media_2',
            'permissions',
            'roles',
            'jwt'
        ]);

        if(!in_array($user->getRoleNameAttribute(), ['Admin', 'Cashier', 'Kasir', 'Owner'])) {
            return $this->respondFailed("Hanya user admin ataupun kasir yang diperbolehkan
login");
        }


        if($request->header('FCM'))
            $user->update(['fcm' => $request->header('FCM')]);

        if($user->getPermittedLocations(false, true) == null && count($user-
>getPermittedLocations(false,true)) <= 0) {
            return $this->respondFailed("User tidak terkait pada outlet, silahkan hubungi
pemilik");
        }

        $user->update(['jwt' => $token]);
        $user->token = $token;
        $user->role = $user->getRoleNameAttribute();
        $user->permitted_location = $user->getPermittedLocations(false, true);
        $permissions = [];
        foreach ($user->getPermissionsViaRoles() as $key => $value) {
            $permissions[] = $value->name;
        }
        $user->permission = $permissions;

        $business = Business::findOrFail($user->business_id);
        $user->is_generated = (boolean)$business->is_generated;

        return $this->respondSuccess($user, "Login berhasil");
    }

    public function logout(Request $request) {
        try {
            Auth::invalidate(Auth::getToken());

            return response()->json([
                'status' => true,
                'message' => 'User logged out successfully'
            ]);
        } catch (\Exception $exception) {
            return response()->json([
                'status' => false,
                'message' => 'Sorry, the user cannot be logged out'
            ], 500);
        }
    }

    // 1. Send reset password email
    public function generateResetToken(Request $request)
    {
        $client = new \GuzzleHttp\Client(['timeout' => 60]);
        $result = $client->request('POST', env('MAIN_URL').'/api/forgot/password', [
            'form_params' => [
                'email' => $request->input('email')
            ],
        ]);

        if($result->getStatusCode() == 201 || $result->getStatusCode() == 200){
            $body = json_decode($result->getBody());
            if($body->status) {
                return $this->respondSuccess(null, 'Reset password konfirmasi telah dikirim');
            } else {
                return $this->respondFailed('Kesalahan terjadi, tidak bisa mengirim link
konfirmasi');
            }
        } else {
            return $this->respondFailed('Kesalahan terjadi, tidak bisa mengirim link
konfirmasi');
        }
    }
```

```
    // 2. Reset Password
    public function resetPassword(Request $request)
    {
        // Check input is valid
        $rules = [
            'token'    => 'required',
            'email' => 'required|email'
        ];
        $this->validate($request, $rules);

        $request['password'] = $this->getRandomWord();
        $request['password_confirmation'] = $request['password'];

        // Reset the password
        $response = $this->broker()->reset(
        $this->credentials($request),
            function ($user, $password) {
                $user->password = app('hash')->make($password);
                $user->save();
            }
        );

        return $response == Password::PASSWORD_RESET
            ? $this->respondSuccess(['new_password' => $request['password']], "Password berhasil
diganti")
            : $this->respondSuccess("Terjadi kesalahan, tidak dapat mengganti password");
    }

    public function getRandomWord($len = 10) {
        $word = array_merge(range('a', 'z'), range('A', 'Z'));
        shuffle($word);
        return substr(implode($word), 0, $len);
    }

    /**
     * Get the password reset credentials from the request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array
     */
    protected function credentials(Request $request)
    {
        return $request->only('email', 'password', 'token');
    }

    /**
     * Get the broker to be used during password reset.
     *
     * @return \Illuminate\Contracts\Auth\PasswordBroker
     */
    public function broker()
    {
        $passwordBrokerManager = new PasswordBrokerManager(app());
        return $passwordBrokerManager->broker();
    }

    public function postRegister(Request $request)
    {
        try {
            $fullname = explode(" ", $request->input('fullname'));
            $request->merge([
                "currency_id" => "54",
                "start_date" => date("d/m/Y"),
                "website" => "",
                "country" => "Indonesia",
                "zip_code" => "",
                "tax_label_1" => "",
                "tax_number_1" => "",
                "tax_label_2" => "",
                "tax_number_2" => "",
                "fy_start_month" => "1",
                "date_format" => "m/d/Y",
                "accounting_method" => "avco",
                "surname" => "",
                "first_name" => $fullname[0],
                "last_name" => count($fullname) > 1 ? $fullname[1] : ''
            ]);
            $validator = $this->validate($request,
                [
                'name' => 'required|max:255',
                'currency_id' => 'required|numeric',
```

```
                'country' => 'required|max:255',
                'time_zone' => 'required|max:255',
                'email' => 'sometimes|nullable|email|unique:users|max:255',
                'first_name' => 'required|max:255',
                'password' => 'required|min:4|max:255',
                'fy_start_month' => 'required',
                'accounting_method' => 'required'
                ]
            );

            DB::beginTransaction();

            //Create owner.
            $owner_details = $request->only(['surname', 'first_name', 'last_name', 'email',
'password', 'language', 'contact_no']);
            $user = User::create_user($owner_details);
            // $user->sendEmailVerificationNotification();

            $business_details = $request->only(['name', 'start_date', 'currency_id',
'time_zone']);

            if($business_details['time_zone'] == 'WIB') {
                $business_details['time_zone'] = 'Asia/Jakarta';
            } else if($business_details['time_zone'] == 'WITA') {
                $business_details['time_zone'] = 'Asia/Makassar';
            } else if($business_details['time_zone'] == 'WIT') {
                $business_details['time_zone'] = 'Asia/Jayapura';
            } else {
                $business_details['time_zone'] = 'Asia/Jakarta';
            }

            $business_details['fy_start_month'] = 1;

            $business_location = $request->only(['name', 'country', 'state', 'city', 'zip_code',
'website', 'mobile','landmark', 'alternate_number']);
            $business_location_gudang = $request->only(['name', 'country', 'state', 'city',
'zip_code', 'website', 'mobile', 'landmark', 'alternate_number']);

            if(!empty($business_location)) {
                $business_location['name'] = $request->input('outlet_name');
            }
            // change value name = Gudang
            if (!empty($business_location_gudang)) {
                $business_location_gudang['name'] = 'Gudang';
                $business_location_gudang['type'] = 'gudang';
                $business_location_gudang['is_default'] = 1;
            }

            $business_location['is_default'] = 1;

            //Create the business
            $business_details['owner_id'] = $user->id;
            if (!empty($business_details['start_date'])) {
                $business_details['start_date'] = Carbon::createFromFormat('d/m/Y',
$business_details['start_date'])->toDateString();
            }

            //upload logo
            $logo_name = $this->businessUtil->uploadFileNew($request, 'business_logo',
'business_logos');
            if (!empty($logo_name)) {
                $business_details['logo'] = $logo_name;
            }

            $business_details['pos_settings'] = json_encode($this->businessUtil-
>defaultPosSettings());

            // get referal code
            if ($request->referral != null) {
                $businessReferral = Business::where('referal_code', $request->referral)->first();
                if ($businessReferral) {
                    $business_details['referal_id'] = $businessReferral->id;
                }
            }

            $business = $this->businessUtil->createNewBusiness($business_details);

            //Update user with business id
            $user->business_id = $business->id;
            if($request->header('FCM')){
                $user->fcm = $request->header('FCM');
```

```
            }
            $user->save();

            // set is reseller
            $business->referal_code = "KIPAREF" . $business->id;
            $business->tipe_referal = 'reseller';
            $business->save();

            $this->businessUtil->newBusinessDefaultResources($business->id, $user->id);
            $new_location_gudang = $this->businessUtil->addLocation($business->id,
$business_location_gudang);
            $new_location = $this->businessUtil->addLocation($business->id, $business_location);

            //create new permission with the new location
            Permission::create(['guard_name' => 'web', 'name' => 'location.' . $new_location->id
]);
            Permission::create(['guard_name' => 'web', 'name' => 'location.' .
$new_location_gudang->id ]);

            DB::commit();

            //Process payment information if superadmin is installed & package information is
present
            $package_id = 1;
            if (!empty($package_id)) {
                $package = \App\Superadmin\Package::find($package_id);
                if (!empty($package)) {
                    Auth::factory()->setTTL(600); // 12 H
                    $token = Auth::attempt(['email' => $user->email, 'password' =>
$owner_details['password']]);
                    $user->update(['jwt' => $token]);
                    // return $this->businessUtil->pay($package_id, $token, 1);
                    $user = Auth::user()->makeHidden([
                        'custom_field_1',
                        'custom_field_2',
                        'custom_field_3',
                        'custom_field_4',
                        'bank_details',
                        'id_proof_name',
                        'id_proof_number',
                        'fb_link',
                        'twitter_link',
                        'social_media_1',
                        'social_media_2',
                        'permissions',
                        'roles'
                    ]);
                    $user->token = $token;

                    $user->role = $user->getRoleNameAttribute();
                    $user->permitted_location = $user->getPermittedLocations(false, true);

                    return $this->respondSuccess($user, "Register berhasil");
                    // return redirect('home');
                }
            }

            return $this->respondSuccess($user, "Register berhasil");
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            $message = 'Gagal mendaftarkan user';
            if (isset($e->response)) {
                if(array_key_exists('email', $e->response->original)){
                    $message = 'Email sudah terdaftar';
                }
            }

            return $this->respondFailed($message, 409);
        }
    }

    public function postRegisterKipatronic(Request $request)
    {
        $digitalUtil = new DigitalUtil('http://128.199.220.53:8080/apps/v8/users/register_kipa');

        $input = $request->only(['name', 'address', 'phone', 'email', 'province', 'city']);

        $uuid = $this->generateRandomString(16);
```

```
        $pin = $this->generateRandomString(6, true);
        $name = $input['name'];
        $address = $input['address'];
        $phone = $input['phone'];
        $email = $input['email'];
        $province = $input['province'];
        $city = $input['city'];

        $doRegister = $digitalUtil->register($uuid, $pin, $name, $address, $phone, $email,
$province, $city, 3634, '61471');

        if($doRegister->success) {
            return $this->respondSuccess($input, $doRegister->msg);
        } else {
            return $this->respondFailed($doRegister->msg);
        }
    }

    public function checkEmail(Request $request)
    {
        try {
            $email = $request->email;
            $user = User::where('email', $email)->get();

            if (count($user) > 0) {
                return $this->respondFailed("Email telah terdaftar", 409);
            } else {
                return $this->respondSuccess(null, "Email belum terdaftar");
            }
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            $message = 'Gagal mendaftarkan user';

            return $this->respondFailed($message, 409);
        }
    }

    private function generateRandomString($length = 10, $num_only = false) {
        if(!$num_only) $characters =
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
        else $characters = '0123456789';

        $charactersLength = strlen($characters);
        $randomString = '';
        for ($i = 0; $i < $length; $i++) {
            $randomString .= $characters[rand(0, $charactersLength - 1)];
        }
        return $randomString;
    }

    public function generateData() {
        try {
            $business_id = Auth::user()->business_id;
            $business = Business::find($business_id);
            if ($business->is_generated) {
                return $this->respondFailed("Data sudah tergenerate");
            }

            DB::beginTransaction();

            GenerateDataUtil::generateData($business_id, Auth::user()->id);

            $account = Account::where('type', 'saldo')->where('business_id', $business_id)-
>first();

            AccountTransaction::createAccountTransaction([
                'amount' => '25000',
                'account_id' => $account->id,
                'type' => 'credit',
                'sub_type' => 'bonus',
                'note' => 'Saldo Bonus Awal Registrasi',
                'created_by' => Auth::user()->id
            ]);

            DB::commit();
            // echo "test";

            return $this->respondSuccess(null, "Berhasil generate data");
```

```
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal generate data");
        }
    }

}
```

b. *Util Controller*

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\File;

class UtilController extends Controller
{
    public function getProvince()
    {
        $provinces = json_decode(File::get('data/provinces.json'));

        if($provinces){
            return $this->respondArray($provinces);
        } else {
            return $this->respondFailed("Terjadi kesalahan");
        }
    }

    public function getCity($id)
    {
        $cities = json_decode(File::get('data/regencies.json'));

        if($cities){
            $city = array_filter(
                $cities,
                function ($e) use (&$id) {
                    return $e->province_id == $id;
                }
            );

            $data = [];
            foreach($city as $value) {
                $data[] = $value;
            }

            return $this->respondArray($data);
        } else {
            return $this->respondFailed("Terjadi kesalahan pada server");
        }
    }
}
```

c. *User Controller*

```
<?php

namespace App\Http\Controllers;

use App\Utils\ModuleUtil;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;
use Auth;

class UserController extends Controller
{
    /**
     * All Utils instance.
     *
     */
    protected $moduleUtil;

    /**
     * Constructor
     *
     * @param ProductUtils $product
     * @return void
     */
    public function __construct(ModuleUtil $moduleUtil)
    {
        $this->moduleUtil = $moduleUtil;
    }

    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $user = Auth::user()->makeHidden([
            'custom_field_1',
            'custom_field_2',
            'custom_field_3',
            'custom_field_4',
            'bank_details',
            'id_proof_name',
            'id_proof_number',
            'fb_link',
            'twitter_link',
            'social_media_1',
            'social_media_2',
        ]);

        $referal_code = $user->business->referal_code == NULL ? NULL : $user->business-
>referal_code;
        if($referal_code) {
            $firebase = $this->moduleUtil->generateShortLink($referal_code);
            if($firebase->status) {
                $referal_code = $firebase->shortLink;
            } else {
                $referal_code = $firebase->message;
            }
        } else {
            $referal_code = '';
        }
        $user->referal_url = $referal_code;

        return $this->respondSuccess($user);
    }

    public function getPermissions()
    {
        $rolename = Auth::user()->getRoleNames()[0];
        $business_id = Auth::user()->business_id;
        $role = Role::where('business_id', $business_id)
                    ->where('name', $rolename)
                    ->with(['permissions'])
                    ->first();
        $role_permissions = [];
        foreach ($role->permissions as $role_perm) {
            $role_permissions[] = $role_perm->name;
```

```
        }
        return $this->respondArray($role_permissions);
    }

    public function getNotifications(Request $request)
    {
        $notifications = Auth::user()->notifications()->orderBy('created_at', 'DESC');

        Auth::user()->unreadNotifications->markAsRead();

        $start = 0;
        $limit = $request->input('limit') ? $request->input('limit') : 20;
        $paged = $request->input('page') ? $request->input('page') : 1;

        $start = ($paged - 1) * $limit;

        $notifications = $notifications->offset($start)->limit($limit)->get();

        $filter_type = $request->input('type') ? $request->input('type') : 'all';

        $notifications_data = [];
        $last_date = "";
        foreach($notifications as $key => $notification) {
            $formatedDate = date('dmy', strtotime($notification->created_at));
            if($last_date != $formatedDate) {
                $last_date = $formatedDate;
                $notifications_data[] = [
                    'title' => date('d F Y', strtotime($notification->created_at)),
                    'msg' => "",
                    'type' => "date",
                    'link' => "",
                    'read_at' => "",
                    'created_at' => ""
                ];
            }
            $data = $notification->data;
            $data_type = $data['type'];
            if (in_array($notification->type,
[\App\Notifications\RecurringInvoiceNotification::class])) {
                $title = '';
                $msg = '';
                $type = '';
                $link = '';
                if ($notification->type ==
                    \App\Notifications\RecurringInvoiceNotification::class) {
                    $title = !empty($data['invoice_status']) && $data['invoice_status'] ==
'draft' ?
                    "Tidak dapat membuat faktur"
                    : "Faktur Baru berhasil dibuat";
                    $msg = !empty($data['invoice_status']) && $data['invoice_status'] == 'draft'
?
                    "Tidak dapat membuat faktur untuk No. Berlangganan .:
".!empty($data['subscription_no']) ? $data['subscription_no'] : ''.". Stok wajib tidak tersedia
untuk produk ".$data['out_of_stock_product']
                    : "Faktur Baru dibuat untuk Berlangganan no.:
"+!empty($data['subscription_no']) ? $data['subscription_no'] : ''+", No. Faktur .:
"+!empty($data['invoice_no']) ? $data['invoice_no'] : '';
                    $type = "RecurringInvoiceNotification";
                    $link = '#';
                }

                if($filter_type == 'all') {
                    $notifications_data[] = [
                        'title' => $title,
                        'msg' => $msg,
                        'type' => $data_type,
                        'link' => $link,
                        'read_at' => $notification->read_at,
                        'created_at' => $notification->created_at->diffForHumans()
                    ];
                } else {
                    continue;
                }
            } elseif(in_array($notification->type,
[\App\Notifications\ProductNotification::class, \App\Notifications\MasterDataNotification::class,
\App\Notifications\TransactionNotification::class])) {
                $title = '';
                $msg = '';
                $type = '';
                $link = '';
                if ($notification->type ==
```

```
                    \App\Notifications\ProductNotification::class) {
                    $product_detail = json_decode($data['product']);
                    $data_type = 'data';
                    if ($data['type'] == 'add') {
                        $title = "Penambahan produk baru";
                        $msg = "Penambahan produk dengan nama : $product_detail->name";
                        $link = env('APP_URL') . '/products/notification/'.$notification-
>id.'/generate';
                    } else if ($data['type'] == 'edit') {
                        $title = "Perubahan produk baru";
                        $msg = "Produk dengan nama : $product_detail->name, sku: $product_detail-
>sku telah diubah";
                        $link = '#';
                    }
                    $type = 'ProductNotification';
                } else if ($notification->type ==
                    \App\Notifications\MasterDataNotification::class) {
                    $title = '';
                    $type = $data['type'];
                    $link = '#';
                    $name = '';
                    $detail = json_decode($data['detail'], TRUE);
                    $data_type = 'data';

                    if ($type == 'unit') {
                        $title = 'Penambahan satuan baru';
                        $name = $detail['actual_name'];
                    } else if ($type == 'brand') {
                        $title = 'Penambahan merek baru';
                        $name = $detail['name'];
                    } else if ($type == 'category') {
                        $title = 'Penambahan category baru';
                        $name = $detail['name'];
                    } else if ($type == 'contact') {
                        $title = 'Penambahan kontak baru';
                        $name = $detail['name'];
                    } else if ($type == 'account') {
                        $title = 'Penambahan akun keuangan baru';
                        $name = $detail['name'];
                    } else if ($type == 'variation') {
                        $title = 'Penambahan produk variation baru';
                        $name = $detail['name'];
                    }

                    if($data['action'] == 'add') {
                        $msg = "Penambahan $type dengan nama : $name";
                    } else if($data['action'] == 'edit') {
                        $msg = "Data $type dengan nama : $name telah diubah";
                    }
                } else if ($notification->type ==
                    \App\Notifications\TransactionNotification::class) {
                    $title = '';
                    $icon_class = 'ic-notif-update';
                    $type = $data['type'];
                    $link = '#';
                    $name = '';
                    $detail = json_decode($data['detail'], TRUE);

                    if ($type == 'stock') {
                        $title = 'Tambahkan stok produk';
                        $data_type = 'stock';
                        if(array_key_exists('product', $detail)) {
                            if((int)$detail['details']['qty_available'] > 0)
                                $msg = 'Stok '.$detail['product']['name'].' tersisa
'.(int)$detail['details']['qty_available'].' di outlet '. $detail['location']['name'];
                            else
                                $msg = 'Stok '.$detail['product']['name'].' habis di outlet '.
$detail['location']['name'];
                        } else {
                            $msg = 'Peringatan Stok ' . $detail['name'];
                        }
                        $icon_class = 'ic-notif-stok';
                    } else if ($type == 'sales_due') {
                        $title = 'Piutang Jatuh Tempo';
                        $data_type = 'due';
                        $msg = 'Piutang Jatuh Tempo dari ' . $detail['contact']['name'];
                        $icon_class = 'ic-notif-utang';
                    } else if ($type == 'purchase_due') {
                        $title = 'Hutang Jatuh Tempo';
                        $data_type = 'due';
                        $msg = 'Hutang Jatuh Tempo dari ' . $detail['contact']['name'];
```

```
                    $icon_class = 'ic-notif-utang';
                } else if($type == 'subscription_due') {
                    $title = 'Layanan Segera Habis';
                    $data_type = 'subscription';
                    $msg = 'Paket Layanan Akan Segera Habis';
                    $icon_class = 'ic-notif-layanan';
                }
            }

            if($filter_type == 'all' || $filter_type == $data_type) {
                $notifications_data[] = [
                    'title' => $title,
                    'msg' => $msg,
                    'type' => $data_type,
                    'link' => $link,
                    'read_at' => $notification->read_at,
                    'created_at' => $notification->created_at->diffForHumans()
                ];
            } else {
                continue;
            }
        }
    }
    return $this->respondArray($notifications_data);
}

public function updatePassword(Request $request)
{
    $isValid = $this->validator($request, [
        'current_password' => 'required|string',
        'new_password' => 'required|string',
    ]);
    if(!$isValid) return $this->respondFailed("Bad Request", 400);

    $user = Auth::user();

    if (Hash::check($request->input('current_password'), $user->password)) {
        $user->password = Hash::make($request->input('new_password'));
        $user->save();

        return $this->respondSuccess();
    } else {
        return $this->respondFailed("The password you entered is incorrect");
    }
}

public function updateProfile(Request $request)
{
    try {
        $fullname = $request->input('full_name');
        if($fullname) {
            $fullname = explode(' ', $fullname);
        }

        $input = $request->only(['surname', 'language', 'marital_status',
            'blood_group', 'contact_no', 'fb_link', 'twitter_link', 'social_media_1',
            'social_media_2', 'permanent_address', 'current_address',
            'guardian_name', 'custom_field_1', 'custom_field_2',
            'custom_field_3', 'custom_field_4', 'id_proof_name', 'id_proof_number']);

        $input['first_name'] = $fullname[0];
        if (count($fullname) > 1) {
            unset($fullname[0]);
            $input['last_name'] = implode(" ", $fullname);
        } else {
            $input['last_name'] = '';
        }

        if (!empty($request->input('dob'))) {
            $input['dob'] = $this->moduleUtil->uf_date($request->input('dob'));
        }
        if (!empty($request->input('bank_details'))) {
            $input['bank_details'] = json_encode($request->input('bank_details'));
        }

        $user = Auth::user();
        $user->update($input);

        return $this->respondSuccess($user, "Berhasil mengupdate profile");
    } catch (\Exception $e) {
```

```
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal mengupdate profile");
        }
    }

    public function updatePhotoProfile(Request $request) {
        $isValid = $this->validator($request, [
            'image_url' => 'required|image|mimes:jpeg,png,jpg,gif,svg|max:2000',
        ]);
        if(!$isValid) return $this->respondFailed("Bad Request", 400);

        if ($request->hasFile('image_url')) {
            $image = $request->file('image_url');
            $name = md5(Auth::id().time()).'.'.$image->getClientOriginalExtension();
            $destinationPath = base_path().'/public/assets/user_images/';
            $image->move($destinationPath, $name);
            $updateId = Auth::user()->update(['image_url' =>
url('/').'/assets/user_images/'.$name]);
            if($updateId)
                return $this->respondSuccess(['image_url' =>
url('/').'/assets/user_images/'.$name]);
            else
                return $this->respondFailed();
        } else {
            return $this->respondFailed("Bad Request", 400);
        }
    }
}
```

d.*Business Controller*

```
<?php

namespace App\Http\Controllers;

use App\Business;
use App\Currency;
use App\System;
use App\TaxRate;
use App\Unit;
use App\User;
use App\Utils\BusinessUtil;
use Auth;

use App\Utils\ModuleUtil;
use App\Utils\RestaurantUtil;
use Carbon\Carbon;
use DateTimeZone;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\DB;

use Spatie\Permission\Models\Permission;
use App\Rules\Captcha;

class BusinessController extends Controller
{
    protected $businessUtil;
    protected $restaurantUtil;
    protected $moduleUtil;
    protected $mailDrivers;

    /**
     * Constructor
     *
     * @param ProductUtils $product
     * @return void
     */
    public function __construct(BusinessUtil $businessUtil, RestaurantUtil $restaurantUtil,
ModuleUtil $moduleUtil)
    {
        $this->businessUtil = $businessUtil;
        $this->moduleUtil = $moduleUtil;

        $this->avlble_modules = [
            'tables' => [ 'name' => __('restaurant.tables'),
                    'tooltip' => __('restaurant.tooltip_tables')
                ] ,
            'modifiers' => [ 'name' => __('restaurant.modifiers'),
                    'tooltip' => __('restaurant.tooltip_modifiers')
                ],
            'service_staff' => [
                    'name' => __('restaurant.service_staff'),
                    'tooltip' => __('restaurant.tooltip_service_staff')
                ],
            'kitchen' => [
                'name' => __('restaurant.kitchen_for_restaurant')
            ],
            'account' => ['name' => __('lang_v1.account')],
            'subscription' => ['name' => __('lang_v1.enable_subscription')],
            'booking' => ['name' => __('lang_v1.enable_booking')]
        ];

        $this->theme_colors = [
            'blue' => 'Blue',
            'black' => 'Black',
            'purple' => 'Purple',
            'green' => 'Green',
            'red' => 'Red',
            'yellow' => 'Yellow',
            'blue-light' => 'Blue Light',
            'black-light' => 'Black Light',
            'purple-light' => 'Purple Light',
            'green-light' => 'Green Light',
            'red-light' => 'Red Light',
        ];

        $this->mailDrivers = [
                'smtp' => 'SMTP',
```

```
                'sendmail' => 'Sendmail',
                'mailgun' => 'Mailgun',
                'mandrill' => 'Mandrill',
                'ses' => 'SES',
                'sparkpost' => 'Sparkpost'
            ];
    }

    /**
     * Shows registration form
     *
     * @return \Illuminate\Http\Response
     */
    public function getRegister()
    {
        if (!env('ALLOW_REGISTRATION', true)) {
            return redirect('/');
        }

        $currencies = $this->businessUtil->allCurrencies();

        $timezone_list = $this->businessUtil->allTimeZones();

        $months = [];
        for ($i=1; $i<=12; $i++) {
            $months[$i] = __('business.months.' . $i);
        }

        $accounting_methods = $this->businessUtil->allAccountingMethods();
        $package_id = request()->package;

        $system_settings = System::getProperties(['superadmin_enable_register_tc',
'superadmin_register_tc'], true);

        return view('business.register', compact(
            'currencies',
            'timezone_list',
            'months',
            'accounting_methods',
            'package_id',
            'system_settings'
        ));
    }

    /**
     * Handles the registration of a new business and it's owner
     *
     * @return \Illuminate\Http\Response
     */
    public function postRegister(Request $request)
    {
        if (!env('ALLOW_REGISTRATION', true)) {
            return redirect('/');
        }

        try {
            $fullname = explode(" ", $request->input('fullname'));
            $request->merge([
                "currency_id" => "54",
                "start_date" => date("d/m/Y"),
                "website" => "",
                "alternate_number" => "",
                "country" => "Indonesia",
                "state" => "",
                "city" => "",
                "zip_code" => "",
                "landmark" => "",
                "tax_label_1" => "",
                "tax_number_1" => "",
                "tax_label_2" => "",
                "tax_number_2" => "",
                "fy_start_month" => "1",
                "accounting_method" => "avco",
                "surname" => "",
                "time_zone" => "Asia/Jakarta",
                "first_name" => $fullname[0],
                "last_name" => count($fullname) > 1 ? $fullname[1] : ''
            ]);
            $validator = $request->validate(
                [
                'name' => 'required|max:255',
```

```
                'currency_id' => 'required|numeric',
                'country' => 'required|max:255',
                'time_zone' => 'required|max:255',
                'email' => 'sometimes|nullable|email|unique:users|max:255',
                'first_name' => 'required|max:255',
                'password' => 'required|min:4|max:255',
                'fy_start_month' => 'required',
                'accounting_method' => 'required',
                'g-recaptcha-response' => new Captcha()
                ],
                [
                'name.required' => __('validation.required', ['attribute' =>
__('business.business_name')]),
                'name.currency_id' => __('validation.required', ['attribute' =>
__('business.currency')]),
                'country.required' => __('validation.required', ['attribute' =>
__('business.country')]),
                'time_zone.required' => __('validation.required', ['attribute' =>
__('business.time_zone')]),
                'email.email' => __('validation.email', ['attribute' => __('business.email')]),
                'email.email' => __('validation.unique', ['attribute' => __('business.email')]),
                'first_name.required' => __('validation.required', ['attribute' =>
                    __('business.first_name')]),
                'username.min' => __('validation.min', ['attribute' => __('business.username')]),
                'password.required' => __('validation.required', ['attribute' =>
__('business.username')]),
                'password.min' => __('validation.min', ['attribute' => __('business.username')]),
                'fy_start_month.required' => __('validation.required', ['attribute' =>
__('business.fy_start_month')]),
                'accounting_method.required' => __('validation.required', ['attribute' =>
__('business.accounting_method')]),
                'g-recaptcha-response.required' => __('validation.required', ['attribute' =>
__('business.g-recaptcha-response')]),
                ]
            );

            DB::beginTransaction();

            //Create owner.
            $owner_details = $request->only(['surname', 'first_name', 'last_name', 'email',
'password', 'language']);
            $user = User::create_user($owner_details);
            // $user->sendEmailVerificationNotification();

            $business_details = $request->only(['name', 'start_date', 'currency_id',
'time_zone']);
            $business_details['fy_start_month'] = 1;

            $business_location = $request->only(['name', 'country', 'state', 'city', 'zip_code',
'landmark', 'website', 'mobile', 'alternate_number']);
            $business_location_gudang = $request->only(['name', 'country', 'state', 'city',
'zip_code', 'landmark', 'website', 'mobile', 'alternate_number']);

            // change value name = Gudang
            if (!empty($business_location_gudang)) {
                $business_location_gudang['name'] = 'Gudang';
            }

            //Create the business
            $business_details['owner_id'] = $user->id;
            if (!empty($business_details['start_date'])) {
                $business_details['start_date'] = Carbon::createFromFormat('d/m/Y',
$business_details['start_date'])->toDateString();
            }

            //upload logo
            $logo_name = $this->businessUtil->uploadFile($request, 'business_logo',
'business_logos');
            if (!empty($logo_name)) {
                $business_details['logo'] = $logo_name;
            }

            $business_details['pos_settings'] = json_encode([
                "disable_draft" => "1",
                "disable_order_tax" => "1"
            ]);

            $business = $this->businessUtil->createNewBusiness($business_details);

            //Update user with business id
            $user->business_id = $business->id;
```

```
            $user->save();

            $this->businessUtil->newBusinessDefaultResources($business->id, $user->id);
            $new_location_gudang = $this->businessUtil->addLocation($business->id,
$business_location_gudang);
            $new_location = $this->businessUtil->addLocation($business->id, $business_location);

            //create new permission with the new location
            Permission::create(['name' => 'location.' . $new_location->id ]);
            Permission::create(['name' => 'location.' . $new_location_gudang->id ]);

            DB::commit();

            //Process payment information if superadmin is installed & package information is
present
            $is_installed_superadmin = $this->moduleUtil->isSuperadminInstalled();
            $package_id = $request->get('package_id', null);
            if ($is_installed_superadmin && !empty($package_id) && (config('app.env') != 'demo'))
{
                $package = \Modules\Superadmin\Entities\Package::find($package_id);
                if (!empty($package)) {
                    Auth::login($user);
                    return redirect()->route('register-pay', ['package_id' => $package_id]);
                    // return redirect('home');
                }
            }

            $output = ['success' => 1,
                    'msg' => __('business.business_created_succesfully')
                ];

            return redirect('login')->with('status', $output);
        } catch (\Exception $e) {
            DB::rollBack();
            // dd($e->validator->errors());
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            $output = [
                        'msg' => $e->getMessage(),
                    ];
            if (isset($e->validator)) {
                return back()->with('status', $e->validator->errors())->withInput()-
>withErrors($e->validator->errors());
            } else {
                return back()->with('status', $output);
            }
        }
    }

    /**
     * Handles the validation username
     *
     * @return \Illuminate\Http\Response
     */
    public function postCheckUsername(Request $request)
    {
        $username = $request->input('username');

        if (!empty($request->input('username_ext'))) {
            $username .= $request->input('username_ext');
        }

        $count = User::where('username', $username)->count();
        if ($count == 0) {
            echo "true";
            exit;
        } else {
            echo "false";
            exit;
        }
    }

    /**
     * Shows business settings form
     *
     * @return \Illuminate\Http\Response
     */
    public function getBusinessSettings()
    {
        if (!Auth::user()->can('business_settings.access')) {
```

```
            return $this->respondFailed('Tidak diizinkan', 403);
        }

        $business_id = Auth::user()->business_id;
        $business = Business::where('id', $business_id)->select([
            'name',
            'time_zone',
            DB::raw('CONVERT(IFNULL(transaction_edit_days, 0), UNSIGNED INTEGER) as
transaction_edit_days'),
            DB::raw('CONVERT(IFNULL(default_profit_percent, 0), UNSIGNED INTEGER) as
default_profit_percent'),
            DB::raw('CONVERT(IFNULL(default_sales_discount, 0), UNSIGNED INTEGER) as
default_sales_discount'),
            DB::raw('CONVERT(IFNULL(default_unit, 0), UNSIGNED INTEGER) as default_unit'),
            'sku_prefix',
            'logo'
        ])->first();

        if($business->default_sales_discount == null) $business->default_sales_discount = 0;
        if($business->logo != null) $business->logo = env('AWS_CDN_URL') . '/business_logos/' .
$business->logo;

        return $this->respondSuccess($business);

    }

    /**
     * Updates business settings
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function postBusinessSettings(Request $request)
    {
        if (!Auth::user()->can('business_settings.access')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }

        try {
            $business_details = $request->only(['name', 'start_date', 'currency_id',
'tax_label_1', 'tax_number_1', 'tax_label_2', 'tax_number_2', 'default_profit_percent',
'default_sales_tax', 'default_sales_discount', 'sell_price_tax', 'sku_prefix', 'time_zone',
'fy_start_month', 'accounting_method', 'transaction_edit_days', 'sales_cmsn_agnt',
'item_addition_method', 'currency_symbol_placement', 'on_product_expiry',
                'stop_selling_before', 'default_unit', 'expiry_type', 'date_format',
                'time_format', 'ref_no_prefixes', 'theme_color', 'email_settings',
                'sms_settings', 'rp_name', 'amount_for_unit_rp',
                'min_order_total_for_rp', 'max_rp_per_order',
                'redeem_amount_per_unit_rp', 'min_order_total_for_redeem',
                'min_redeem_point', 'max_redeem_point', 'rp_expiry_period',
                'rp_expiry_type', 'custom_labels']);

            if (!empty($request->input('enable_rp')) &&  $request->input('enable_rp') == 1) {
                $business_details['enable_rp'] = 1;
            } else {
                $business_details['enable_rp'] = 0;
            }

            if($business_details['time_zone'] == 'WIB') {
                $business_details['time_zone'] = 'Asia/Jakarta';
            } else if($business_details['time_zone'] == 'WITA') {
                $business_details['time_zone'] = 'Asia/Makassar';
            } else if($business_details['time_zone'] == 'WIT') {
                $business_details['time_zone'] = 'Asia/Jayapura';
            } else {
                $business_details['time_zone'] = 'Asia/Jakarta';
            }

            $business_details['currency_symbol_placement'] = 'before';
            $business_details['fy_start_month'] = 1;
            $business_details['accounting_method'] = 'avco';
            $business_details['date_format'] = 'm/d/Y';
            $business_details['time_format'] = '24';

            $business_details['amount_for_unit_rp'] =
!empty($business_details['amount_for_unit_rp']) ? $this->businessUtil-
>num_uf($business_details['amount_for_unit_rp']) : 1;
            $business_details['min_order_total_for_rp'] =
!empty($business_details['min_order_total_for_rp']) ? $this->businessUtil-
>num_uf($business_details['min_order_total_for_rp']) : 1;
```

```
            $business_details['redeem_amount_per_unit_rp'] =
!empty($business_details['redeem_amount_per_unit_rp']) ? $this->businessUtil-
>num_uf($business_details['redeem_amount_per_unit_rp']) : 1;
            $business_details['min_order_total_for_redeem'] =
!empty($business_details['min_order_total_for_redeem']) ? $this->businessUtil-
>num_uf($business_details['min_order_total_for_redeem']) : 1;

            $business_details['default_profit_percent'] =
!empty($business_details['default_profit_percent']) ? $this->businessUtil-
>num_uf($business_details['default_profit_percent']) : 0;

            $business_details['default_sales_discount'] =
!empty($business_details['default_sales_discount']) ? $this->businessUtil-
>num_uf($business_details['default_sales_discount']) : 0;

            if (!empty($business_details['start_date'])) {
                $business_details['start_date'] = $this->businessUtil-
>uf_date($business_details['start_date']);
            }

            if (!empty($request->input('enable_tooltip')) &&  $request->input('enable_tooltip')
== 1) {
                $business_details['enable_tooltip'] = 1;
            } else {
                $business_details['enable_tooltip'] = 0;
            }

            $business_details['enable_product_expiry'] = !empty($request-
>input('enable_product_expiry')) &&  $request->input('enable_product_expiry') == 1 ? 1 : 0;
            $business_details['on_product_expiry'] = 'keep_selling';
            if ($business_details['on_product_expiry'] == 'keep_selling') {
                $business_details['stop_selling_before'] = null;
            }

            $business_details['stock_expiry_alert_days'] = !empty($request-
>input('stock_expiry_alert_days')) ? $request->input('stock_expiry_alert_days') : 30;

            //Check for Purchase currency
            if (!empty($request->input('purchase_in_diff_currency')) &&  $request-
>input('purchase_in_diff_currency') == 1) {
                $business_details['purchase_in_diff_currency'] = 1;
                $business_details['purchase_currency_id'] = $request-
>input('purchase_currency_id');
                $business_details['p_exchange_rate'] = $request->input('p_exchange_rate');
            } else {
                $business_details['purchase_in_diff_currency'] = 0;
                $business_details['purchase_currency_id'] = null;
                $business_details['p_exchange_rate'] = 1;
            }

            //upload logo
            if($request->file('business_logo')) {
                $logo_name = $this->businessUtil->uploadFile($request, 'business_logo',
'business_logos')->data;
                if (!empty($logo_name)) {
                    $business_details['logo'] = $logo_name;
                }
            }

            $checkboxes = [ 'enable_inline_tax',
                'enable_brand', 'enable_lot_number', 'enable_racks', 'enable_row',
'enable_position', 'enable_sub_units'];

            foreach ($checkboxes as $value) {
                $business_details[$value] = !empty($request->input($value)) &&  $request-
>input($value) == 1 ? 1 : 0;
            }
            $business_details['enable_editing_product_from_purchase'] = 1;
            $business_details['enable_brand'] = 1;
            $business_details['enable_category'] = 1;
            $business_details['enable_sub_category'] = 1;

            $business_id = Auth::user()->business_id;
            $business = Business::where('id', $business_id)->first();

            // //Update business settings
            if (!empty($business_details['logo'])) {
                $business->logo = $business_details['logo'];
            } else {
                unset($business_details['logo']);
            }
```

```
            //System settings
            $shortcuts = $request->input('shortcuts');
            $business_details['keyboard_shortcuts'] = json_encode($shortcuts);

            //Enabled modules
            $enabled_modules = $request->input('enabled_modules');
            $business_details['enabled_modules'] = !empty($enabled_modules) ? $enabled_modules :
['account'];

            $business->fill($business_details);
            $business->save();

            $business = Business::where('id', $business->id)->select([
                'name',
                'time_zone',
                'transaction_edit_days',
                'default_profit_percent',
                'default_sales_discount',
                'default_unit',
                'sku_prefix',
                'logo'
            ])->first();

            if($business->default_sales_discount == null) $business->default_sales_discount = 0;
            if($business->logo != null) $business->logo =  env('AWS_CDN_URL') .
'/business_logos/' . $business->logo;


            return $this->respondSuccess($business, "Berhasil update pengaturan toko");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed($e->getMessage());
        }
    }

    /**
     * Handles the validation email
     *
     * @return \Illuminate\Http\Response
     */
    public function postCheckEmail(Request $request)
    {
        $email = $request->input('email');

        $query = User::where('email', $email);

        if (!empty($request->input('user_id'))) {
            $user_id = $request->input('user_id');
            $query->where('id', '!=', $user_id);
        }

        $exists = $query->exists();
        if (!$exists) {
            echo "true";
            exit;
        } else {
            echo "false";
            exit;
        }
    }

    public function getEcomSettings()
    {
        try {
            $api_token = request()->header('API-TOKEN');
            $api_settings = $this->moduleUtil->getApiSettings($api_token);

            $settings = Business::where('id', $api_settings->business_id)
                        ->value('ecom_settings');

            $settings_array = !empty($settings) ? json_decode($settings, true) : [];

            if (!empty($settings_array['slides'])) {
                foreach ($settings_array['slides'] as $key => $value) {
                    $settings_array['slides'][$key]['image_url'] = !empty($value['image']) ?
url('uploads/img/' . $value['image']) : '';
                }
            }
```

```
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondWentWrong($e);
        }

        return $this->respond($settings_array);
    }
}
```

e. *Brand Controller*

```
<?php

namespace App\Http\Controllers;

use App\Brands;
use App\Business;
use App\User;
use Illuminate\Http\Request;
use Yajra\DataTables\Facades\DataTables;
use Illuminate\Support\Facades\DB;
use Auth;

class BrandController extends Controller
{

    public function __construct()
    {
        // I Love her :)
    }

    public function index()
    {
        if (!Auth::user()->can('brand.view') && !Auth::user()->can('brand.create')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;

        $brands = Brands::where('business_id', $business_id)
                    ->select(['name', 'id', 'is_digital'])
                    ->orderBy('name', 'ASC')
                    ->get();

        return $this->respondArray($brands);
    }

    public function store(Request $request)
    {
        if (!Auth::user()->can('brand.create')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $input = $request->only(['name', 'description']);
            $business_id = Auth::user()->business_id;
            $input['business_id'] = $business_id;
            $input['created_by'] = Auth::user()->id;

            $brand = Brands::create($input);

            return $this->respondSuccess($brand, "Merek berhasil ditambahkan");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menambahkan merek");
        }
    }

    public function update(Request $request, $id)
    {
        if (!Auth::user()->can('brand.update')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $input = $request->only(['name', 'description']);
```

```
            $business_id = Auth::user()->business_id;

            $brand = Brands::where('business_id', $business_id)->findOrFail($id);

            if($brand->is_digital) {
                return $this->respondFailed("Merek digital tidak dapat diedit");
            }

            $brand->name = $input['name'];
            $brand->description = $input['description'];
            $brand->save();

            return $this->respondSuccess($brand, "Merek berhasil diupdate");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal mengupdate merek");
        }
    }

    public function destroy($id)
    {
        if (!Auth::user()->can('brand.delete')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $business_id = Auth::user()->business_id;

            $brand = Brands::where('business_id', $business_id)->findOrFail($id);

            if($brand->is_digital) {
                return $this->respondFailed("Merek digital tidak dapat dihapus");
            }

            $brand->delete();

            return $this->respondSuccess(null, "Merek berhasil dihapus");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menghapus merek");
        }
    }
}
```

f. *Unit Controller*

```
<?php

namespace App\Http\Controllers;

use App\Unit;
use App\Product;
use App\Business;

use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;

use App\Utils\Util;
use App\User;

use Auth;

class UnitController extends Controller
{

    protected $commonUtil;

    public function __construct(Util $commonUtil)
    {
        $this->commonUtil = $commonUtil;
    }

    public function index()
    {
        if (!Auth::user()->can('unit.view') && !Auth::user()->can('unit.create')) {
            return $this->respondFailed("Tidak diizinkan", 403);
```

```
        }

        $business_id = Auth::user()->business_id;

        $unit = Unit::where('business_id', $business_id)
                    ->select(['actual_name', 'short_name', 'allow_decimal', 'id',
                        'base_unit_multiplier'])
                    ->get();

        return $this->respondArray($unit);
    }

    public function store(Request $request)
    {
        if (!Auth::user()->can('unit.create')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $input = $request->only(['actual_name', 'short_name', 'allow_decimal']);
            $input['business_id'] = Auth::user()->business_id;
            $input['created_by'] = Auth::user()->id;

            if ($request->has('define_base_unit')) {
                if (!empty($request->input('base_unit_id')) && !empty($request-
>input('base_unit_multiplier'))) {
                    $base_unit_multiplier = $this->commonUtil->num_uf($request-
>input('base_unit_multiplier'));
                    if ($base_unit_multiplier != 0) {
                        $input['base_unit_id'] = $request->input('base_unit_id');
                        $input['base_unit_multiplier'] = $base_unit_multiplier;
                    }
                }
            }

            $unit = Unit::create($input);

            return $this->respondSuccess($unit, "Satuan berhasil ditambahkan");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menambahkan satuan");
        }
    }

    public function update(Request $request, $id)
    {
        if (!Auth::user()->can('unit.update')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $input = $request->only(['actual_name', 'short_name', 'allow_decimal']);
            $business_id = Auth::user()->business_id;

            $unit = Unit::where('business_id', $business_id)->findOrFail($id);

            $unit->actual_name = $input['actual_name'];
            $unit->short_name = $input['short_name'];
            $unit->allow_decimal = $input['allow_decimal'];
            if ($request->has('define_base_unit')) {
                if (!empty($request->input('base_unit_id')) && !empty($request-
>input('base_unit_multiplier'))) {
                    $base_unit_multiplier = $this->commonUtil->num_uf($request-
>input('base_unit_multiplier'));
                    if ($base_unit_multiplier != 0) {
                        $unit->base_unit_id = $request->input('base_unit_id');
                        $unit->base_unit_multiplier = $base_unit_multiplier;
                    }
                }
            } else {
                $unit->base_unit_id = null;
                $unit->base_unit_multiplier = null;
            }

            $unit->save();

            return $this->respondSuccess($unit, "Satuan berhasil diupdate");
        } catch (\Exception $e) {
```

```
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal mengupdate satuan");
        }
    }

    public function destroy($id)
    {
        if (!Auth::user()->can('unit.delete')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $business_id = Auth::user()->business_id;

            $unit = Unit::where('business_id', $business_id)->findOrFail($id);

            //check if any product associated with the unit
            $exists = Product::where('unit_id', $unit->id)
                            ->exists();
            if (!$exists) {
                $unit->delete();

                return $this->respondSuccess(null, "Satuan berhasil dihapus");
            } else {
                return $this->respondFailed("Gagal menghapus satuan karena terdapat produk yang
terhubung");
            }
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menghapus satuan");
        }
    }
}
```

g. *Category Controller*

```
<?php

namespace App\Http\Controllers;

use DB;
use Auth;
use App\Category;
use App\Business;
use App\Utils\ModuleUtil;
use Illuminate\Http\Request;
use Yajra\DataTables\Facades\DataTables;

class CategoryController extends Controller
{

    protected $moduleUtil;

    public function __construct(ModuleUtil $moduleUtil)
    {
        $this->moduleUtil = $moduleUtil;
    }

    public function index()
    {
        if (!Auth::user()->can('category.view') && !Auth::user()->can('category.create')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }

        $business_id = Auth::user()->business_id;

        $category = Category::where('business_id', $business_id)
                ->where('is_digital', '!=', 1)
                ->where('parent_id', 0)
                ->select(['name', 'short_code', 'id', 'parent_id', 'is_digital']);

        return $this->respondArray($category->get());
    }

    public function getSubcategories($id)
    {
        if (!Auth::user()->can('category.view') && !Auth::user()->can('category.create')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }

        $business_id = Auth::user()->business_id;

        $category = Category::where('business_id', $business_id)
                ->where('parent_id', $id)
                ->where('is_digital', '!=', 1)
                ->select(['name', 'short_code', 'id', 'parent_id', 'is_digital']);

        return $this->respondArray($category->get());
    }

    public function create()
    {
        if (!Auth::user()->can('category.create')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }

        $business_id = Auth::user()->business_id;
        $categories = Category::where('business_id', $business_id)
            ->where('parent_id', 0)
            ->select(['name', 'short_code', 'id'])
            ->get();

        return $this->respondArray($categories);
    }

    public function store(Request $request)
    {
        if (!Auth::user()->can('category.create')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }

        try {
            $input = $request->only(['name', 'is_digital']);
            if (!empty($request->input('add_as_sub_cat')) && $request->input('add_as_sub_cat') ==
1 && !empty($request->input('parent_id'))) {
```

```
                $input['parent_id'] = $request->input('parent_id');
            } else {
                $input['parent_id'] = 0;
            }
            $input['business_id'] = Auth::user()->business_id;
            $input['created_by'] = Auth::user()->id;

            $category = Category::create($input);

            return $this->respondSuccess($category, "Berhasil menambahkan category");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile() . "Line:" . $e->getLine() . "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menambahkan category");
        }

        return $output;
    }

    public function edit($id)
    {
        if (!Auth::user()->can('category.update')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }

        $business_id = Auth::user()->business_id;
        $category = Category::where('business_id', $business_id)->find($id);

        $parent_categories = Category::where('business_id', $business_id)
            ->where('parent_id', 0)
            ->where('id', '!=', $id)
            ->select(['name', 'id'])
            ->get();

        $is_parent = false;

        if ($category->parent_id == 0) {
            $is_parent = true;
            $selected_parent = null;
        } else {
            $selected_parent = $category->parent_id;
        }

        $data = [
            'category' => $category,
            'parent_category' => $parent_categories,
            'is_parent' => $is_parent,
            'selected_parent' => $selected_parent
        ];

        return $this->respondSuccess($data);
    }

    public function update(Request $request, $id)
    {
        if (!Auth::user()->can('category.update')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }


        try {
            $input = $request->only(['name']);
            $business_id = Auth::user()->business_id;

            $category = Category::where('business_id', $business_id)->findOrFail($id);
            $categoryOld = Category::where('business_id', $business_id)->findOrFail($id);

            $category->name = $input['name'];
            if (!empty($request->input('add_as_sub_cat')) && $request->input('add_as_sub_cat') ==
1 && !empty($request->input('parent_id'))) {
                $category->parent_id = $request->input('parent_id');
            } else {
                $category->parent_id = 0;
            }
            $category->save();

            return $this->respondSuccess($category, "Berhasil mengupdate category");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile() . "Line:" . $e->getLine() . "Message:" . $e-
>getMessage());
```

```
            return $this->respondFailed("Gagal mengupdate category");
        }
    }

    public function destroy($id)
    {
        if (!Auth::user()->can('category.delete')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }

        try {
            $business_id = Auth::user()->business_id;

            $category = Category::where('business_id', $business_id)->findOrFail($id);
            $category->delete();

            return $this->respondSuccess($category, "Berhasil menghapus category");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile() . "Line:" . $e->getLine() . "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menghapus category");
        }
    }
}
```

h. *Contact Controller*

```
<?php

namespace App\Http\Controllers;

use App\Business;
use App\Contact;
use App\CustomerGroup;
use App\Transaction;
use App\TransactionPayment;
use App\User;
use App\Utils\ModuleUtil;
use App\Utils\TransactionUtil;
use App\Utils\Util;
use Auth;
use DB;
use Excel;
use Illuminate\Http\Request;
use Yajra\DataTables\Facades\DataTables;

class ContactController extends Controller
{
    protected $commonUtil;
    protected $transactionUtil;
    protected $moduleUtil;

    public function __construct(
        Util $commonUtil,
        ModuleUtil $moduleUtil,
        TransactionUtil $transactionUtil
    ) {
        $this->commonUtil = $commonUtil;
        $this->moduleUtil = $moduleUtil;
        $this->transactionUtil = $transactionUtil;
        $this->transactionTypes = [
                    'sell' => __('sale.sale'),
                    'purchase' => __('lang_v1.purchase'),
                    'sell_return' => __('lang_v1.sell_return'),
                    'purchase_return' =>  __('lang_v1.purchase_return'),
                    'opening_balance' => __('lang_v1.opening_balance'),
                    'payment' => __('lang_v1.payment')
                ];
    }

    public function index()
    {
        $type = request()->get('type');

        $types = ['supplier', 'customer'];

        if (empty($type) || !in_array($type, $types)) {
            return redirect()->back();
        }
```

```
        $business_id = request()->session()->get('user.business_id');
        if($type == 'supplier') {
            $countData = Contact::where('contacts.business_id', $business_id)
                ->onlySuppliers()->count();
        } else {
            $countData = Contact::where('contacts.business_id', $business_id)
                ->onlyCustomers()->count();
        }

        if (request()->ajax()) {
            if ($type == 'supplier') {

                return $this->indexSupplier();
            } elseif ($type == 'customer') {

                return $this->indexCustomer();
            } else {
                die("Not Found");
            }
        }

        $reward_enabled = (request()->session()->get('business.enable_rp') == 1 &&
in_array($type, ['customer'])) ? true : false;

        return view('contact.index')
            ->with(compact('type', 'reward_enabled', 'countData'));
    }

    public function indexSupplier(Request $request)
    {
        if (!Auth::user()->can('supplier.view')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }

        $business_id = Auth::user()->business_id;

        $contacts = Contact::leftjoin('transactions AS t', 'contacts.id', '=', 't.contact_id')
                    ->where('contacts.business_id', $business_id)
                    ->where('is_digital', 0)
                    ->onlySuppliers()
                    ->select(['contacts.contact_id', 'supplier_business_name', 'name', 'mobile',
                        'contacts.type', 'contacts.id', 'contacts.is_default',
'contacts.addition_information', 'contacts.credit_limit', 'contacts.customer_group_id',
                        DB::raw("SUM(IF(t.type = 'purchase', final_total, 0)) as
total_purchase"),
                        DB::raw("SUM(IF(t.type = 'purchase', (SELECT SUM(amount) FROM
transaction_payments WHERE transaction_payments.transaction_id=t.id), 0)) as purchase_paid"),
                        DB::raw("SUM(IF(t.type = 'purchase_return', final_total, 0)) as
total_purchase_return"),
                        DB::raw("SUM(IF(t.type = 'purchase_return', (SELECT SUM(amount) FROM
transaction_payments WHERE transaction_payments.transaction_id=t.id), 0)) as
purchase_return_paid"),
                        DB::raw("SUM(IF(t.type = 'opening_balance', final_total, 0)) as
opening_balance"),
                        DB::raw("SUM(IF(t.type = 'opening_balance', (SELECT SUM(IF(is_return =
1,-1*amount,amount)) FROM transaction_payments WHERE transaction_payments.transaction_id=t.id),
0)) as opening_balance_paid")
                            ])
                    ->groupBy('contacts.id');

        $start = 0;
        $limit = $request->input('limit') ? $request->input('limit') : 20;
        $paged = $request->input('page') ? $request->input('page') : 1;

        $start = ($paged - 1) * $limit;
        $contacts = $contacts->offset($start)->limit($limit)->get();

        foreach($contacts as $key => $contact) {
            $contacts[$key]->credit_limit = $contact->credit_limit ? (int)$contact->credit_limit
: 0;
            $contacts[$key]->customer_group_id = $contact->customer_group_id ? $contact-
>customer_group_id : 0;
        }

        return $this->respondArray($contacts);
    }

    public function indexCustomer(Request $request)
    {
        if (!Auth::user()->can('customer.view')) {
```

```
            return $this->respondFailed('Tidak diizinkan', 403);
        }

        $business_id = Auth::user()->business_id;

        $contacts = Contact::leftjoin('transactions AS t', 'contacts.id', '=', 't.contact_id')
                    ->leftjoin('customer_groups AS cg', 'contacts.customer_group_id', '=',
'cg.id')
                    ->leftjoin('selling_price_groups AS spg', 'contacts.selling_price_group_id',
'=' ,'spg.id')
                    ->where('contacts.business_id', $business_id)
                    ->onlyCustomers()
                    ->select(['contacts.contact_id', 'supplier_business_name', 'contacts.name',
'mobile',
                        'contacts.type', 'contacts.id', 'contacts.is_default',
'contacts.addition_information', 'contacts.credit_limit', 'contacts.customer_group_id', 'cg.name
as cg_name', 'spg.name as spg_name',
                        DB::raw("IFNULL(contacts.selling_price_group_id, 0) as
selling_price_group_id"),
                        DB::raw("SUM(IF(t.type = 'purchase', final_total, 0)) as
total_purchase"),
                        DB::raw("SUM(IF(t.type = 'purchase', (SELECT SUM(amount) FROM
transaction_payments WHERE transaction_payments.transaction_id=t.id), 0)) as purchase_paid"),
                        DB::raw("SUM(IF(t.type = 'purchase_return', final_total, 0)) as
total_purchase_return"),
                        DB::raw("SUM(IF(t.type = 'purchase_return', (SELECT SUM(amount) FROM
transaction_payments WHERE transaction_payments.transaction_id=t.id), 0)) as
purchase_return_paid"),
                        DB::raw("SUM(IF(t.type = 'opening_balance', final_total, 0)) as
opening_balance"),
                        DB::raw("SUM(IF(t.type = 'opening_balance', (SELECT SUM(IF(is_return =
1,-1*amount,amount)) FROM transaction_payments WHERE transaction_payments.transaction_id=t.id),
0)) as opening_balance_paid")
                    ])
                    ->groupBy('contacts.id');

        $start = 0;
        $limit = $request->input('limit') ? $request->input('limit') : 20;
        $paged = $request->input('page') ? $request->input('page') : 1;

        $start = ($paged - 1) * $limit;
        $contacts = $contacts->offset($start)->limit($limit)->get();

        foreach($contacts as $key => $contact) {
            $contacts[$key]->credit_limit = $contact->credit_limit ? (int)$contact->credit_limit
: 0;
            $contacts[$key]->customer_group_id = $contact->customer_group_id ? $contact-
>customer_group_id : 0;
        }

        return $this->respondArray($contacts);
    }

    public function create()
    {
        if (!auth()->user()->can('supplier.create') && !auth()->user()->can('customer.create')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = request()->session()->get('user.business_id');

        //Check if subscribed or not
        if (!$this->moduleUtil->isSubscribed($business_id)) {
            return $this->moduleUtil->expiredResponse();
        }

        $types = [];
        if (auth()->user()->can('supplier.create')) {
            $types['supplier'] = __('report.supplier');
        }
        if (auth()->user()->can('customer.create')) {
            $types['customer'] = __('report.customer');
        }
        if (auth()->user()->can('supplier.create') && auth()->user()->can('customer.create')) {
            $types['both'] = __('lang_v1.both_supplier_customer');
        }

        $customer_groups = CustomerGroup::forDropdown($business_id);

        return view('contact.create')
            ->with(compact('types', 'customer_groups'));
```

```
    }

    public function createSupplier()
    {
        if (!auth()->user()->can('supplier.create')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = request()->session()->get('user.business_id');

        //Check if subscribed or not
        if (!$this->moduleUtil->isSubscribed($business_id)) {
            return $this->moduleUtil->expiredResponse();
        }

        return view('contact.create_supplier');
    }

    public function createCustomer()
    {
        if (!Auth::user()->can('supplier.create')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }

        $business_id = Auth::user()->business_id;

        //Check if subscribed or not
        if (!$this->moduleUtil->isSubscribed($business_id)) {
            return $this->respondFailed("Masa aktif berlangganan anda telah berakhir");
        }

        $customer_groups = CustomerGroup::where('business_id', $business_id)
                                ->select(['id', 'name'])
                                ->get();

        return $this->respondArray($customer_groups);
    }

    public function store(Request $request)
    {
        if (!Auth::user()->can('supplier.create') && !Auth::user()->can('customer.create')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }

        try {
            $business_id = Auth::user()->business_id;

            if (!$this->moduleUtil->isSubscribed($business_id)) {
                return $this->respondFailed("Masa aktif berlangganan anda telah berakhir");
            }

            $input = $request->only(['type', 'is_digital',
                'name', 'tax_number', 'pay_term_number', 'pay_term_type', 'mobile', 'landline',
'alternate_number', 'city', 'state', 'country', 'landmark', 'customer_group_id', 'contact_id',
'custom_field1', 'custom_field2', 'custom_field3', 'custom_field4', 'email',
'addition_information']);

            $input['business_id'] = $business_id;
            $input['supplier_business_name'] = $request->input('name');
            $input['created_by'] = Auth::user()->id;

            if($request->input('selling_price_group_id')) {
                $input['selling_price_group_id'] = $request->input('selling_price_group_id');
            }

            $input['credit_limit'] = $request->input('credit_limit') != '' ? $this->commonUtil-
>num_uf($request->input('credit_limit')) : null;

            //Check Contact id
            $count = 0;
            if (!empty($input['contact_id'])) {
                $count = Contact::where('business_id', $input['business_id'])
                                ->where('contact_id', $input['contact_id'])
                                ->count();
            }

            if ($count == 0) {
                //Update reference count
                $ref_count = $this->commonUtil->setAndGetReferenceCount($input['type'],
$business_id);
```

```
                if (empty($input['contact_id'])) {
                    //Generate reference number
                    $input['contact_id'] = $this->commonUtil-
>generateReferenceNumber($input['type'], $ref_count);
                }

                $contact = Contact::create($input);

                //Add opening balance
                if (!empty($request->input('opening_balance')) && (int)$request-
>input('opening_balance') > 0) {
                    $this->transactionUtil->createOpeningBalanceTransaction($business_id,
$contact->id, $request->input('opening_balance'));
                }

                $contact = Contact::leftjoin('transactions AS t', 'contacts.id', '=',
't.contact_id')
                            ->leftjoin('customer_groups AS cg', 'contacts.customer_group_id',
'=', 'cg.id')
                            ->leftjoin('selling_price_groups AS spg',
'contacts.selling_price_group_id', '=' ,'spg.id')
                            ->where('contacts.business_id', $business_id)
                            ->where('contacts.id', $contact->id)
                            ->select(['contacts.contact_id', 'supplier_business_name',
'contacts.name', 'mobile',
                                'contacts.type', 'contacts.id', 'contacts.is_default',
'contacts.addition_information', 'contacts.credit_limit', 'contacts.customer_group_id', 'cg.name
as cg_name', 'spg.name as spg_name',
                                DB::raw("IFNULL(contacts.selling_price_group_id, 0) as
selling_price_group_id"),
                                DB::raw("SUM(IF(t.type = 'purchase', final_total, 0)) as
total_purchase"),
                                DB::raw("SUM(IF(t.type = 'purchase', (SELECT SUM(amount) FROM
transaction_payments WHERE transaction_payments.transaction_id=t.id), 0)) as purchase_paid"),
                                DB::raw("SUM(IF(t.type = 'purchase_return', final_total, 0)) as
total_purchase_return"),
                                DB::raw("SUM(IF(t.type = 'purchase_return', (SELECT SUM(amount)
FROM transaction_payments WHERE transaction_payments.transaction_id=t.id), 0)) as
purchase_return_paid"),
                                DB::raw("SUM(IF(t.type = 'opening_balance', final_total, 0)) as
opening_balance"),
                                DB::raw("SUM(IF(t.type = 'opening_balance', (SELECT
SUM(IF(is_return = 1,-1*amount,amount)) FROM transaction_payments WHERE
transaction_payments.transaction_id=t.id), 0)) as opening_balance_paid")
                                ])
                            ->groupBy('contacts.id')->first();

                $contact->credit_limit = $contact->credit_limit ? (int)$contact->credit_limit :
0;
                $contact->customer_group_id = $contact->customer_group_id ? $contact-
>customer_group_id : 0;

                return $this->respondSuccess($contact, "Berhasil menambahkan ".$input['type']);
            } else {
                throw new \Exception("Error Processing Request", 1);
            }
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menambahkan ".$input['type'].$e->getMessage());
        }
    }

    public function show($id)
    {
        if (!auth()->user()->can('supplier.view') && !auth()->user()->can('customer.view')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = request()->session()->get('user.business_id');

        $contact = Contact::where('contacts.id', $id)
                            ->where('contacts.business_id', $business_id)
                            ->join('transactions AS t', 'contacts.id', '=', 't.contact_id')
                            ->select(
                                DB::raw("SUM(IF(t.type = 'purchase', final_total, 0)) as
total_purchase"),
                                DB::raw("SUM(IF(t.type = 'sell' AND t.status = 'final',
final_total, 0)) as total_invoice"),
```

```
                                DB::raw("SUM(IF(t.type = 'purchase', (SELECT SUM(amount) FROM
transaction_payments WHERE transaction_payments.transaction_id=t.id), 0)) as purchase_paid"),
                                DB::raw("SUM(IF(t.type = 'sell' AND t.status = 'final', (SELECT
SUM(IF(is_return = 1,-1*amount,amount)) FROM transaction_payments WHERE
transaction_payments.transaction_id=t.id), 0)) as invoice_received"),
                                DB::raw("SUM(IF(t.type = 'opening_balance', final_total, 0)) as
opening_balance"),
                                DB::raw("SUM(IF(t.type = 'opening_balance', (SELECT SUM(amount)
FROM transaction_payments WHERE transaction_payments.transaction_id=t.id), 0)) as
opening_balance_paid"),
                                'contacts.*'
                            )->first();

        $reward_enabled = (request()->session()->get('business.enable_rp') == 1 &&
in_array($contact->type, ['customer', 'both'])) ? true : false;

        return view('contact.show')
             ->with(compact('contact', 'reward_enabled'));
    }

    public function edit($id)
    {
        if (!auth()->user()->can('supplier.update') && !auth()->user()->can('customer.update')) {
            abort(403, 'Unauthorized action.');
        }

        if (request()->ajax()) {
            $business_id = request()->session()->get('user.business_id');
            $contact = Contact::where('business_id', $business_id)->find($id);

            if (!$this->moduleUtil->isSubscribed($business_id)) {
                return $this->moduleUtil->expiredResponse();
            }

            $types = [];
            if (auth()->user()->can('supplier.create')) {
                $types['supplier'] = __('report.supplier');
            }
            if (auth()->user()->can('customer.create')) {
                $types['customer'] = __('report.customer');
            }
            if (auth()->user()->can('supplier.create') && auth()->user()->can('customer.create'))
{
                $types['both'] = __('lang_v1.both_supplier_customer');
            }

            $customer_groups = CustomerGroup::forDropdown($business_id);

            $ob_transaction =  Transaction::where('contact_id', $id)
                                        ->where('type', 'opening_balance')
                                        ->first();
            $opening_balance = !empty($ob_transaction->final_total) ? $ob_transaction-
>final_total : 0;

            //Deduct paid amount from opening balance.
            if (!empty($opening_balance)) {
                $opening_balance_paid = $this->transactionUtil-
>getTotalAmountPaid($ob_transaction->id);
                if (!empty($opening_balance_paid)) {
                    $opening_balance = $opening_balance - $opening_balance_paid;
                }

                $opening_balance = $this->commonUtil->num_f($ob_transaction->final_total);
            }

            return view('contact.edit')
                ->with(compact('contact', 'types', 'customer_groups', 'opening_balance'));
        }
    }

    public function update(Request $request, $id)
    {
        if (!Auth::user()->can('supplier.update') && !Auth::user()->can('customer.update')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }

        try {
            $input = $request->only(['type', 'name', 'mobile', 'addition_information']);

            $input['credit_limit'] = $request->input('credit_limit') != '' ? $this->commonUtil-
>num_uf($request->input('credit_limit')) : null;
```

```
            $input['supplier_business_name'] = $request->input('name');
            $business_id = Auth::user()->business_id;

            $count = 0;

            //Check Contact id
            if (!empty($input['contact_id'])) {
                $count = Contact::where('business_id', $business_id)
                        ->where('contact_id', $input['contact_id'])
                        ->where('id', '!=', $id)
                        ->count();
            }

            if ($count == 0) {
                $contact = Contact::where('business_id', $business_id)->findOrFail($id);
                foreach ($input as $key => $value) {
                    $contact->$key = $value;
                }

                if($request->input('customer_group_id')) {
                    $contact->customer_group_id = $request->input('customer_group_id');
                }

                if($request->input('selling_price_group_id')) {
                    $contact->selling_price_group_id = $request->input('selling_price_group_id');
                }

                if($request->input('credit_limit')) {
                    $contact->credit_limit = $request->input('credit_limit');
                }

                $contact->save();

                $contact = Contact::leftjoin('transactions AS t', 'contacts.id', '='
't.contact_id')
                            ->leftjoin('customer_groups AS cg', 'contacts.customer_group_id',
'=', 'cg.id')
                            ->leftjoin('selling_price_groups AS spg',
'contacts.selling_price_group_id', '=' ,'spg.id')
                            ->where('contacts.business_id', $business_id)
                            ->where('contacts.id', $id)
                            ->select(['contacts.contact_id', 'supplier_business_name',
'contacts.name', 'mobile',
                                'contacts.type', 'contacts.id', 'contacts.is_default',
'contacts.addition_information', 'contacts.credit_limit', 'contacts.customer_group_id', 'cg.name
as cg_name', 'spg.name as spg_name',
                                DB::raw("IFNULL(contacts.selling_price_group_id, 0) as
selling_price_group_id"),
                                DB::raw("SUM(IF(t.type = 'purchase', final_total, 0)) as
total_purchase"),
                                DB::raw("SUM(IF(t.type = 'purchase', (SELECT SUM(amount) FROM
transaction_payments WHERE transaction_payments.transaction_id=t.id), 0)) as purchase_paid"),
                                DB::raw("SUM(IF(t.type = 'purchase_return', final_total, 0)) as
total_purchase_return"),
                                DB::raw("SUM(IF(t.type = 'purchase_return', (SELECT SUM(amount)
FROM transaction_payments WHERE transaction_payments.transaction_id=t.id), 0)) as
purchase_return_paid"),
                                DB::raw("SUM(IF(t.type = 'opening_balance', final_total, 0)) as
opening_balance"),
                                DB::raw("SUM(IF(t.type = 'opening_balance', (SELECT
SUM(IF(is_return = 1,-1*amount,amount)) FROM transaction_payments WHERE
transaction_payments.transaction_id=t.id), 0)) as opening_balance_paid")
                                ])
                            ->groupBy('contacts.id')->first();

                $contact->credit_limit = $contact->credit_limit ? (int)$contact->credit_limit :
0;
                $contact->customer_group_id = $contact->customer_group_id ? $contact-
>customer_group_id : 0;

                return $this->respondSuccess($contact, "Berhasil mengupdate ".$input['type']);
            } else {
                throw new \Exception("Error Processing Request", 1);
            }
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal mengupdate ".$input['type']);
        }
    }
```

```
    public function destroy($id)
    {
        if (!Auth::user()->can('supplier.delete') && !Auth::user()->can('customer.delete')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }

        try {
            $business_id = Auth::user()->business_id;

            $contact = Contact::where('business_id', $business_id)->findOrFail($id);

            //Check if any transaction related to this contact exists
            $count = Transaction::where('business_id', $business_id)
                                ->where('contact_id', $id)
                                ->count();

            if ($count > 0) {
                return $this->respondFailed("Tidak dapat menghapus ".$contact->type." karena
sudah memiliki transaksi");
            }
            if (!$contact->is_default) {
                $contact->delete();
            }

            return $this->respondSuccess(null, "Berhasil menghapus ".$contact->type);
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menghapus contact");
        }
    }

    public function getCustomers()
    {
        if (request()->ajax()) {
            $term = request()->input('q', '');

            $business_id = request()->session()->get('user.business_id');
            $user_id = request()->session()->get('user.id');

            $contacts = Contact::where('business_id', $business_id);

            $selected_contacts = User::isSelectedContacts($user_id);
            if ($selected_contacts) {
                $contacts->join('user_contact_access AS uca', 'contacts.id', 'uca.contact_id')
                ->where('uca.user_id', $user_id);
            }

            if (!empty($term)) {
                $contacts->where(function ($query) use ($term) {
                    $query->where('name', 'like', '%' . $term .'%')
                            ->orWhere('supplier_business_name', 'like', '%' . $term .'%')
                            ->orWhere('mobile', 'like', '%' . $term .'%')
                            ->orWhere('contacts.contact_id', 'like', '%' . $term .'%');
                });
            }

            $contacts->select(
                'contacts.id',
                DB::raw("IF(contacts.contact_id IS NULL OR contacts.contact_id='', name,
CONCAT(name, ' (', contacts.contact_id, ')')) AS text"),
                'mobile',
                'landmark',
                'city',
                'state',
                'pay_term_number',
                'pay_term_type'
            )
                    ->onlyCustomers();

            if (request()->session()->get('business.enable_rp') == 1) {
                $contacts->addSelect('total_rp');
            }
            $contacts = $contacts->get();
            return json_encode($contacts);
        }
    }

    public function checkContactId(Request $request)
```

```
{
    $contact_id = $request->input('contact_id');

    $valid = 'true';
    if (!empty($contact_id)) {
        $business_id = $request->session()->get('user.business_id');
        $hidden_id = $request->input('hidden_id');

        $query = Contact::where('business_id', $business_id)
                        ->where('contact_id', $contact_id);
        if (!empty($hidden_id)) {
            $query->where('id', '!=', $hidden_id);
        }
        $count = $query->count();
        if ($count > 0) {
            $valid = 'false';
        }
    }
    echo $valid;
    exit;
}

public function getImportContacts()
{
    if (!auth()->user()->can('supplier.create') && !auth()->user()->can('customer.create')) {
        abort(403, 'Unauthorized action.');
    }

    $zip_loaded = extension_loaded('zip') ? true : false;

    //Check if zip extension it loaded or not.
    if ($zip_loaded === false) {
        $output = ['success' => 0,
                        'msg' => 'Please install/enable PHP Zip archive for import'
                    ];

        return view('contact.import')
            ->with('notification', $output);
    } else {
        return view('contact.import');
    }
}

public function postImportContacts(Request $request)
{
    if (!auth()->user()->can('supplier.create') && !auth()->user()->can('customer.create')) {
        abort(403, 'Unauthorized action.');
    }

    try {
        //Set maximum php execution time
        ini_set('max_execution_time', 0);

        if ($request->hasFile('contacts_csv')) {
            $file = $request->file('contacts_csv');
            $imported_data = Excel::load($file->getRealPath())
                                ->noHeading()
                                ->skipRows(1)
                                ->get()
                                ->toArray();
            $business_id = $request->session()->get('user.business_id');
            $user_id = $request->session()->get('user.id');

            $formated_data = [];

            $is_valid = true;
            $error_msg = '';

            DB::beginTransaction();
            foreach ($imported_data as $key => $value) {
                //Check if 21 no. of columns exists
                if (count($value) != 21) {
                    $is_valid =  false;
                    $error_msg = "Number of columns mismatch";
                    break;
                }

                $row_no = $key + 1;
                $contact_array = [];

                //Check contact type
```

```
$contact_type = '';
if (!empty($value[0])) {
    $contact_type = strtolower(trim($value[0]));
    if (in_array($contact_type, ['supplier', 'customer', 'both'])) {
        $contact_array['type'] = $contact_type;
    } else {
        $is_valid =  false;
        $error_msg = "Invalid contact type in row no. $row_no";
        break;
    }
} else {
    $is_valid =  false;
    $error_msg = "Contact type is required in row no. $row_no";
    break;
}

//Check contact name
if (!empty($value[1])) {
    $contact_array['name'] = $value[1];
} else {
    $is_valid =  false;
    $error_msg = "Contact name is required in row no. $row_no";
    break;
}

//Check supplier fields
if (in_array($contact_type, ['supplier', 'both'])) {
    //Check business name
    if (!empty(trim($value[2]))) {
        $contact_array['supplier_business_name'] = $value[2];
    } else {
        $is_valid =  false;
        $error_msg = "Business name is required in row no. $row_no";
        break;
    }

    //Check pay term
    if (trim($value[6]) != '') {
        $contact_array['pay_term_number'] = trim($value[6]);
    } else {
        $is_valid =  false;
        $error_msg = "Pay term is required in row no. $row_no";
        break;
    }

    //Check pay period
    $pay_term_type = strtolower(trim($value[7]));
    if (in_array($pay_term_type, ['days', 'months'])) {
        $contact_array['pay_term_type'] = $pay_term_type;
    } else {
        $is_valid =  false;
        $error_msg = "Pay term period is required in row no. $row_no";
        break;
    }
}

//Check contact ID
if (!empty(trim($value[3]))) {
    $count = Contact::where('business_id', $business_id)
                ->where('contact_id', $value[3])
                ->count();


    if ($count == 0) {
        $contact_array['contact_id'] = $value[3];
    } else {
        $is_valid =  false;
        $error_msg = "Contact ID already exists in row no. $row_no";
        break;
    }
}

//Tax number
if (!empty(trim($value[4]))) {
    $contact_array['tax_number'] = $value[4];
}

//Check opening balance
if (!empty(trim($value[5])) && $value[5] != 0) {
    $contact_array['opening_balance'] = trim($value[5]);
}
```

```
                    //Check credit limit
                    if (trim($value[8]) != '' && in_array($contact_type, ['customer', 'both'])) {
                        $contact_array['credit_limit'] = trim($value[8]);
                    }

                    //Check email
                    if (!empty(trim($value[9]))) {
                        if (filter_var(trim($value[9]), FILTER_VALIDATE_EMAIL)) {
                            $contact_array['email'] = $value[9];
                        } else {
                            $is_valid =  false;
                            $error_msg = "Invalid email id in row no. $row_no";
                            break;
                        }
                    }

                    //Mobile number
                    if (!empty(trim($value[10]))) {
                        $contact_array['mobile'] = $value[10];
                    } else {
                        $is_valid =  false;
                        $error_msg = "Mobile number is required in row no. $row_no";
                        break;
                    }

                    //Alt contact number
                    $contact_array['alternate_number'] = $value[11];

                    //Landline
                    $contact_array['landline'] = $value[12];

                    //City
                    $contact_array['city'] = $value[13];

                    //State
                    $contact_array['state'] = $value[14];

                    //Country
                    $contact_array['country'] = $value[15];

                    //Landmark
                    $contact_array['landmark'] = $value[16];

                    //Cust fields
                    $contact_array['custom_field1'] = $value[17];
                    $contact_array['custom_field2'] = $value[18];
                    $contact_array['custom_field3'] = $value[19];
                    $contact_array['custom_field4'] = $value[20];

                    $formated_data[] = $contact_array;
                }
                if (!$is_valid) {
                    throw new \Exception($error_msg);
                }

                if (!empty($formated_data)) {
                    foreach ($formated_data as $contact_data) {
                        $ref_count = $this->transactionUtil->setAndGetReferenceCount('contacts');
                        //Set contact id if empty
                        if (empty($contact_data['contact_id'])) {
                            $contact_data['contact_id'] = $this->commonUtil-
>generateReferenceNumber('contacts', $ref_count);
                        }

                        $opening_balance = 0;
                        if (isset($contact_data['opening_balance'])) {
                            $opening_balance = $contact_data['opening_balance'];
                            unset($contact_data['opening_balance']);
                        }

                        $contact_data['business_id'] = $business_id;
                        $contact_data['created_by'] = $user_id;

                        $contact = Contact::create($contact_data);

                        if (!empty($opening_balance)) {
                            $this->transactionUtil->createOpeningBalanceTransaction($business_id,
$contact->id, $opening_balance);
                        }
                    }
```

```
                }

                $output = ['success' => 1,
                            'msg' => __('product.file_imported_successfully')
                        ];

                DB::commit();
            }
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            $output = ['success' => 0,
                            'msg' => $e->getMessage()
                        ];
            return redirect()->route('contacts.import')->with('notification', $output);
        }

        return redirect()->action('ContactController@index', ['type' => 'supplier'])-
>with('status', $output);
    }

    public function getLedger()
    {
        if (!auth()->user()->can('supplier.view') && !auth()->user()->can('customer.view')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = request()->session()->get('user.business_id');
        $contact_id = request()->input('contact_id');
        $transaction_types = explode(',', request()->input('transaction_types'));
        $show_payments = request()->input('show_payments') == 'true' ? true : false;

        //Get transactions
        $query1 = Transaction::where('transactions.contact_id', $contact_id)
                            ->where('transactions.business_id', $business_id)
                            ->where('status', '!=', 'draft')
                            ->whereIn('type', $transaction_types)
                            ->with(['location']);

        if (!empty(request()->start_date) && !empty(request()->end_date)) {
            $start = request()->start_date;
            $end =  request()->end_date;
            $query1->whereDate('transactions.transaction_date', '>=', $start)
                        ->whereDate('transactions.transaction_date', '<=', $end);
        }

        $transactions = $query1->get();

        $ledger = [];
        foreach ($transactions as $transaction) {
            $ledger[] = [
                'date' => $transaction->transaction_date,
                'ref_no' => in_array($transaction->type, ['sell', 'sell_return']) ? $transaction-
>invoice_no : $transaction->ref_no,
                'type' => $this->transactionTypes[$transaction->type],
                'location' => $transaction->location->name,
                'payment_status' =>  __('lang_v1.' . $transaction->payment_status),
                'total' => $transaction->final_total,
                'payment_method' => '',
                'debit' => '',
                'credit' => '',
                'others' => $transaction->additional_notes
            ];
        }

        if ($show_payments) {
            $query2 = TransactionPayment::join(
                'transactions as t',
                'transaction_payments.transaction_id',
                '=',
                't.id'
            )
                ->leftJoin('business_locations as bl', 't.location_id', '=', 'bl.id')
                ->where('t.contact_id', $contact_id)
                ->where('t.business_id', $business_id)
                ->where('t.status', '!=', 'draft');

            if (!empty(request()->start_date) && !empty(request()->end_date)) {
                $start = request()->start_date;
```

```
                $end =  request()->end_date;
                $query1->whereDate('transactions.transaction_date', '>=', $start)
                            ->whereDate('transactions.transaction_date', '<=', $end);

                if ($show_payments) {
                    $query2->whereDate('paid_on', '>=', $start)
                            ->whereDate('paid_on', '<=', $end);
                }
            }

            $payments = $query2->select('transaction_payments.*', 'bl.name as location_name',
't.type as transaction_type', 't.ref_no', 't.invoice_no')->get();
            $paymentTypes = $this->transactionUtil->payment_types();
            foreach ($payments as $payment) {
                $ref_no = in_array($payment->transaction_type, ['sell', 'sell_return']) ?
$payment->invoice_no :  $payment->ref_no;
                $ledger[] = [
                    'date' => $payment->paid_on,
                    'ref_no' => $payment->payment_ref_no,
                    'type' => $this->transactionTypes['payment'],
                    'location' => $payment->location_name,
                    'payment_status' => '',
                    'total' => '',
                    'payment_method' => !empty($paymentTypes[$payment->method]) ?
$paymentTypes[$payment->method] : '',
                    'debit' => in_array($payment->transaction_type, ['purchase', 'sell_return'])
? $payment->amount : '',
                    'credit' => in_array($payment->transaction_type, ['sell', 'purchase_return',
'opening_balance']) ? $payment->amount : '',
                    'others' => $payment->note . '<small>' . __('account.payment_for') . ': ' .
$ref_no . '</small>'
                ];
            }
        }

        //Sort by date
        if (!empty($ledger)) {
            usort($ledger, function ($a, $b) {
                $t1 = strtotime($a['date']);
                $t2 = strtotime($b['date']);
                return $t2 - $t1;
            });
        }
        return view('contact.ledger')
             ->with(compact('ledger'));
    }

    public function postCustomersApi(Request $request)
    {
        try {
            $api_token = $request->header('API-TOKEN');

            $api_settings = $this->moduleUtil->getApiSettings($api_token);

            $business = Business::find($api_settings->business_id);

            $data = $request->only(['name', 'email']);

            $customer = Contact::where('business_id', $api_settings->business_id)
                                ->where('email', $data['email'])
                                ->whereIn('type', ['customer', 'both'])
                                ->first();

            if (empty($customer)) {
                $data['type'] = 'customer';
                $data['business_id'] = $api_settings->business_id;
                $data['created_by'] = $business->owner_id;
                $data['mobile'] = 0;

                $ref_count = $this->commonUtil->setAndGetReferenceCount('contacts', $business-
>id);

                $data['contact_id'] = $this->commonUtil->generateReferenceNumber('contacts',
$ref_count, $business->id);

                $customer = Contact::create($data);
            }
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());
```

```
            return $this->respondWentWrong($e);
        }

        return $this->respond($customer);
    }
}
```

i. *Employee Controller*

```
<?php

namespace App\Http\Controllers;

use App\BusinessLocation;
use App\Contact;
use App\System;

use App\User;
use App\Utils\ModuleUtil;
use DB;
use Auth;

use Illuminate\Http\Request;

use Spatie\Permission\Models\Role;
use Yajra\DataTables\Facades\DataTables;
use Illuminate\Support\Facades\Hash;

class EmployeeController extends Controller
{
    protected $moduleUtil;

    public function __construct(ModuleUtil $moduleUtil)
    {
        $this->moduleUtil = $moduleUtil;
    }

    public function index(Request $request)
    {
        $me = Auth::user();

        $users = User::where('business_id', $me->business_id)
                    ->where('id', '!=', $me->id)
                    ->where('is_cmmsn_agnt', 0);

        if($request->input('name')) {
            $name = $request->input('name');
            $users = $users->where(function($query) use ($name){
                $query->where('first_name', 'LIKE', '%'.$name.'%');
                $query->orWhere('last_name', 'LIKE', '%'.$name.'%');
            });
        }

        if($request->input('start_date') && $request->input('end_date')) {
            $start_date = $request->input('start_date');
            $end_date = $request->input('end_date');

            $users = $users->whereBetween('created_at', [date('Y-m-d', strtotime($start_date))."
00:00:00", date('Y-m-d', strtotime($end_date))." 23:59:59"]);
        }

        $users = $users->select(['id', 'username', 'business_id', 'cmmsn_percent',
                DB::raw("CONCAT(COALESCE(surname, ''), ' ', COALESCE(first_name, ''), ' ',
COALESCE(last_name, '')) as full_name"), 'email']);

        $start = 0;
        $limit = $request->input('limit') ? $request->input('limit') : 20;
        $paged = $request->input('page') ? $request->input('page') : 1;

        $start = ($paged - 1) * $limit;
        $users = $users->offset($start)->limit($limit)->get();

        foreach($users as $key => $user) {
            $role_name = $this->moduleUtil->getUserRoleObject($user->id);
            $users[$key]->full_name = substr($user->full_name, 0, 1) === ' ' ? substr($user-
>full_name, 1) : $user->full_name;
            $users[$key]->role = $role_name;
            $users[$key]->locations = $user->getPermittedLocations(true, false);
            unset($users[$key]->roles);
            unset($users[$key]->permissions);
        }
```

```
        return $this->respondArray($users);
    }

    public function create()
    {
        if (!Auth::user()->can('user.create')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;

        //Check if subscribed or not, then check for users quota
        if (!$this->moduleUtil->isSubscribed($business_id)) {
            return $this->respondFailed("Masa aktif berlangganan anda telah berakhir");
        } elseif (!$this->moduleUtil->isQuotaAvailable('users', $business_id)) {
            return $this->respondQuotaExpired("users", $business_id);
        }

        $roles = $this->getRolesArray($business_id);
        $username_ext = $this->getUsernameExtension();
        //$contacts = Contact::contactDropdown($business_id, true, false);
        $locations = BusinessLocation::where('business_id', $business_id)
            ->select(['id', 'name'])->get();

        $data = [
            'roles' => $roles,
            'username_ext' => $username_ext,
            'locations' => $locations
        ];

        return $this->respondSuccess($data);
    }

    public function store(Request $request)
    {
        if (!Auth::user()->can('user.create')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $splitname = explode(" ", $request->input('fullname'));

            $user_details = $request->only(['email', 'password', 'selected_contacts',
'marital_status',
                'blood_group', 'contact_number', 'fb_link', 'twitter_link', 'social_media_1',
                'social_media_2', 'permanent_address', 'current_address',
                'guardian_name', 'custom_field_1', 'custom_field_2',
                'custom_field_3', 'custom_field_4', 'id_proof_name', 'id_proof_number',
'cmmsn_percent']);

            $user_details['first_name'] = $splitname[0];
            if (count($splitname) > 1) {
                $user_details['last_name'] = $splitname[1];
            } else {
                $user_details['last_name'] = '';
            }

            $user_details['username'] = $request->input('email');
            $user_details['status'] = !empty($request->input('is_active')) ? 'active' :
'inactive';

            if (!isset($user_details['selected_contacts'])) {
                $user_details['selected_contacts'] = false;
            }

            if (!empty($request->input('dob'))) {
                $user_details['dob'] = $this->moduleUtil->uf_date($request->input('dob'));
            }

            if (!empty($request->input('bank_details'))) {
                $user_details['bank_details'] = json_encode($request->input('bank_details'));
            }

            $business_id = Auth::user()->business_id;
            $user_details['business_id'] = $business_id;
            $user_details['password'] = Hash::make($user_details['password']);

            DB::beginTransaction();

            $ref_count = $this->moduleUtil->setAndGetReferenceCount('username', $business_id);
            if (blank($user_details['username'])) {
```

```
                $user_details['username'] = $this->moduleUtil-
>generateReferenceNumber('username', $ref_count, $business_id);
            }

            //Check if subscribed or not, then check for users quota
            if (!$this->moduleUtil->isSubscribed($business_id)) {
                return $this->respondFailed("Masa aktif berlangganan anda telah berakhir");
            } elseif (!$this->moduleUtil->isQuotaAvailable('users', $business_id)) {
                return $this->respondQuotaExpired("users", $business_id);
            }

            //Sales commission percentage
            $user_details['cmmsn_percent'] = !empty($user_details['cmmsn_percent']) ? $this-
>moduleUtil->num_uf($user_details['cmmsn_percent']) : 0;

            //Create the user
            $user = User::create($user_details);

            $role_id = $request->input('role');
            $role = Role::findOrFail($role_id);
            $user->assignRole($role->name);

            //Assign selected contacts
            if ($user_details['selected_contacts'] == 1) {
                $contact_ids = $request->get('selected_contact_ids');
                $user->contactAccess()->sync($contact_ids);
            }

            //Grant Location permissions
            $this->giveLocationPermissions($user, $request);

            $user = User::where('id', $user->id)
                    ->select(['id', 'username', 'business_id', 'cmmsn_percent',
                        DB::raw("CONCAT(COALESCE(surname, ''), ' ', COALESCE(first_name, ''), '
', COALESCE(last_name, '')) as full_name"), 'email'])
                    ->first();

            if($user) {
                $role_name = $this->moduleUtil->getUserRoleObject($user->id);
                $user->full_name =  substr($user->full_name, 1);
                $user->role = $role_name;
                $user->locations = $user->getPermittedLocations(true, false);
                unset($user->roles);
                unset($user->permissions);
            }

            DB::commit();

            return $this->respondSuccess($user, "Karyawan berhasil ditambahkan");
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile() . "Line:" . $e->getLine() . "Message:" . $e-
>getMessage());
            //dd($e);
            $statusCode = $e->errorInfo[1];
            $message = "Gagal menambahkan Karyawan";

            if($statusCode == 1062) {
                $message = "Email atau username sudah tersedia";
            }

            return $this->respondFailed($message);
        }
    }

    public function show($id)
    {
        $me = Auth::user();
        $business_id = $me->business_id;
        $user = User::where('business_id', $business_id)
                    ->with(['contactAccess'])
                    ->find($id);

        dd($user);

        return $this->respondSuccess($user);
    }

    public function edit()
    {
        if (!Auth::user()->can('user.update')) {
```

```
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;
        $roles = $this->getRolesArray($business_id);
        $locations = BusinessLocation::where('business_id', $business_id)
                ->select(['id', 'name'])->get();

        $data = [
            'roles' => $roles,
            'locations' => $locations
        ];

        return $this->respondSuccess($data);
    }

    public function update(Request $request, $id)
    {
        if (!Auth::user()->can('user.update')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $splitname = explode(" ", $request->input('fullname'));

            $user_data = $request->only(['email', 'selected_contacts', 'marital_status',
                'blood_group', 'contact_number', 'fb_link', 'twitter_link', 'social_media_1',
                'social_media_2', 'permanent_address', 'current_address',
                'guardian_name', 'custom_field_1', 'custom_field_2',
                'custom_field_3', 'custom_field_4', 'id_proof_name', 'id_proof_number',
'cmmsn_percent']);


            $user_data['first_name'] = $splitname[0];
            if (count($splitname) > 1) {
                $user_data['last_name'] = $splitname[1];
            } else {
                $user_data['last_name'] = '';
            }

            $user_data['username'] = $request->input('email');

            $user_data['status'] = !empty($request->input('is_active')) ? 'active' : 'inactive';
            $business_id = Auth::user()->business_id;

            if (!isset($user_data['selected_contacts'])) {
                $user_data['selected_contacts'] = 0;
            }

            if (!empty($request->input('password'))) {
                $user_data['password'] = Hash::make($request->input('password'));
            }

            //Sales commission percentage
            $user_data['cmmsn_percent'] = !empty($user_data['cmmsn_percent']) ? $this-
>moduleUtil->num_uf($user_data['cmmsn_percent']) : 0;

            if (!empty($request->input('dob'))) {
                $user_data['dob'] = $this->moduleUtil->uf_date($request->input('dob'));
            }

            if (!empty($request->input('bank_details'))) {
                $user_data['bank_details'] = json_encode($request->input('bank_details'));
            }

            $user = User::where('business_id', $business_id)
                ->findOrFail($id);

            $user->update($user_data);

            $role_id = $request->input('role');
            $user_role = $user->roles->first();

            if ($user_role->id != $role_id) {
                $user->removeRole($user_role->name);

                $role = Role::findOrFail($role_id);
                $user->assignRole($role->name);
            }

            //Assign selected contacts
```

```
            if ($user_data['selected_contacts'] == 1) {
                $contact_ids = $request->get('selected_contact_ids');
            } else {
                $contact_ids = [];
            }
            $user->contactAccess()->sync($contact_ids);

            //Grant Location permissions
            $this->giveLocationPermissions($user, $request);

            $user = User::where('id', $id)
                    ->select(['id', 'username', 'business_id', 'cmmsn_percent',
                        DB::raw("CONCAT(COALESCE(surname, ''), ' ', COALESCE(first_name, ''), '
', COALESCE(last_name, '')) as full_name"), 'email'])
                    ->first();

            if($user) {
                $role_name = $this->moduleUtil->getUserRoleObject($user->id);
                $user->full_name =  substr($user->full_name, 1);
                $user->role = $role_name;
                $user->locations = $user->getPermittedLocations(true, false);
                unset($user->roles);
                unset($user->permissions);
            }

            return $this->respondSuccess($user, "Karyawan berhasil diupdate");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile() . "Line:" . $e->getLine() . "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal mengupdate Karyawan ".$e->getMessage());
        }
    }

    public function destroy($id)
    {
        try {
            $me = Auth::user();
            $business_id = $me->business_id;

            User::where('business_id', $business_id)
                ->where('id', $id)->delete();

            return $this->respondSuccess(null, "User berhasil dihapus");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed($e->getMessage());
        }

        return $output;
    }

    private function getUsernameExtension()
    {
        $extension = !empty(System::getProperty('enable_business_based_username')) ? '-'
.str_pad(Auth::user()->business_id, 2, 0, STR_PAD_LEFT) : null;
        return $extension;
    }

    private function getRolesArray($business_id)
    {
        $roles_array = Role::where('business_id', $business_id)->get();
        $roles = [];

        $is_admin = $this->moduleUtil->is_admin(Auth::user(), $business_id);
        foreach ($roles_array as $key => $value) {
            if ($value->name == 'Owner#' . $business_id) {
                continue;
            }
            $temp = (object)[];
            $temp->id = $value->id;
            $temp->name = str_replace('#' . $business_id, '', $value->name);
            $roles[] = $temp;
        }
        return $roles;
    }

    private function giveLocationPermissions($user, $request)
    {
```

```
        $permitted_locations = $user->permitted_locations();

        // dd($permitted_locations);
        $permissions = $request->input('access_all_locations');
        $revoked_permissions = [];
        //If not access all location then revoke permission
        if ($permitted_locations == 'all' && $permissions != 'access_all_locations') {
            $user->revokePermissionTo('access_all_locations');
        }

        //Include location permissions
        $location_permissions = $request->input('location_permissions');

        // dd($location_permissions);
        if (empty($permissions) && !empty($location_permissions)) {
            $permissions = [];
            // dd($location_permissions);
            foreach ($location_permissions as $location_permission) {
                $permissions[] = $location_permission;
            }

            if (is_array($permitted_locations)) {
                foreach ($permitted_locations as $key => $value) {
                    if (!in_array('location.' . $value, $permissions)) {
                        $revoked_permissions[] = 'location.' . $value;
                    }
                }
            }
        }

        if (!empty($revoked_permissions)) {
            $user->revokePermissionTo($revoked_permissions);
        }

        // dd($permissions);
        if (!empty($permissions)) {
            $user->givePermissionTo($permissions);
        }
    }
}
```

j. *CustomerGroup Controller*

```
<?php

namespace App\Http\Controllers;

use Auth;

use App\CustomerGroup;
use App\Utils\Util;
use Illuminate\Http\Request;
use Yajra\DataTables\Facades\DataTables;

class CustomerGroupController extends Controller
{

    public function __construct(Util $commonUtil)
    {
        $this->commonUtil = $commonUtil;
    }

    public function index()
    {
        if (!Auth::user()->can('customer.view')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }
        $business_id = Auth::user()->business_id;

        $customer_group = CustomerGroup::where('business_id', $business_id)
                ->select(['name', 'amount', 'id', 'is_default'])->get();

        return $this->respondArray($customer_group);
    }

    public function create()
    {
        if (!auth()->user()->can('customer.create')) {
            abort(403, 'Unauthorized action.');
        }
```

```
        return view('customer_group.create');
    }

    public function store(Request $request)
    {
        if (!Auth::user()->can('customer.create')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }

        try {
            $input = $request->only(['name', 'amount']);
            $input['business_id'] = Auth::user()->business_id;
            $input['created_by'] = Auth::user()->id;
            $input['amount'] = !empty($input['amount']) ? $this->commonUtil-
>num_uf($input['amount']) : 0;

            $customer_group = CustomerGroup::create($input);

            $customer_group = CustomerGroup::where('business_id', $input['business_id'])
                    ->where('id', $customer_group->id)
                    ->select(['name', 'amount', 'id', 'is_default'])->first();

            return $this->respondSuccess($customer_group, "Berhasil menambahkan Grup Pelanggan");
        } catch (\Exception $e) {
            dd($e);
            \Log::emergency("File:" . $e->getFile() . "Line:" . $e->getLine() . "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menambahkan Grup Pelanggan");
        }
    }

    public function edit($id)
    {
        if (!auth()->user()->can('customer.update')) {
            abort(403, 'Unauthorized action.');
        }

        if (request()->ajax()) {
            $business_id = request()->session()->get('user.business_id');
            $customer_group = CustomerGroup::where('business_id', $business_id)->find($id);

            // return json_encode($customer_group);
            return view('customer_group.edit')
                ->with(compact('customer_group'));
        }
    }

    public function update(Request $request, $id)
    {
        if (!Auth::user()->can('customer.update')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }

        try {
            $input = $request->only(['name', 'amount']);
            $business_id = Auth::user()->business_id;

            $input['amount'] = !empty($input['amount']) ? $this->commonUtil-
>num_uf($input['amount']) : 0;

            $customer_group = CustomerGroup::where('business_id', $business_id)->findOrFail($id);
            $customer_group->name = $input['name'];
            $customer_group->amount = $input['amount'];
            $customer_group->save();

            $customer_group = CustomerGroup::where('business_id', $business_id)
                    ->where('id', $customer_group->id)
                    ->select(['name', 'amount', 'id', 'is_default'])->first();

            return $this->respondSuccess($customer_group, "Berhasil mengupdate Grup Pelanggan");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile() . "Line:" . $e->getLine() . "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal mengupdate Grup Pelanggan");
        }
    }

    public function destroy($id)
    {
```

```
        if (!Auth::user()->can('customer.delete')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }

        try {
            $business_id = Auth::user()->business_id;

            $cg = CustomerGroup::where('business_id', $business_id)->findOrFail($id);
            $cg->delete();

            return $this->respondSuccess(null, "Berhasil menghapus Grup Pelanggan");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile() . "Line:" . $e->getLine() . "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menghapus Grup Pelanggan");
        }
    }
}
```

k. *SellingPriceGroup Controller*

```
<?php

namespace App\Http\Controllers;

use App\SellingPriceGroup;
use Illuminate\Http\Request;
use Spatie\Permission\Models\Permission;
use Yajra\DataTables\Facades\DataTables;
use Auth;

class SellingPriceGroupController extends Controller
{

    public function index(Request $request)
    {
        $business_id = Auth::user()->business_id;
        $countData = SellingPriceGroup::where('business_id', $business_id)->count();

        $price_groups = SellingPriceGroup::where('business_id', $business_id)
            ->select(['name', 'description', 'id']);

        $start = 0;
        $limit = $request->input('limit') ? $request->input('limit') : 20;
        $paged = $request->input('page') ? $request->input('page') : 1;

        $start = ($paged - 1) * $limit;
        $price_groups = $price_groups->offset($start)->limit($limit)->get();

        return $this->respondArray($price_groups);
    }

    public function create()
    {
        if (!auth()->user()->can('product.create')) {
            abort(403, 'Unauthorized action.');
        }

        return view('selling_price_group.create');
    }

    public function store(Request $request)
    {
        if (!Auth::user()->can('product.create')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $input = $request->only(['name', 'description']);
            $business_id = Auth::user()->business_id;
            $input['business_id'] = $business_id;

            $spg = SellingPriceGroup::create($input);

            //Create a new permission related to the created selling price group
            Permission::create(['guard_name' => 'web', 'name' => 'selling_price_group.' . $spg-
>id]);

            return $this->respondSuccess($spg, "Berhasil menambahkan grup harga jual");
        } catch (\Exception $e) {
```

```
            \Log::emergency("File:" . $e->getFile() . "Line:" . $e->getLine() . "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menambahkan grup harga jual");
        }

        return $output;
    }

    public function edit($id)
    {
        if (!auth()->user()->can('product.create')) {
            abort(403, 'Unauthorized action.');
        }

        if (request()->ajax()) {
            $business_id = request()->session()->get('user.business_id');
            $spg = SellingPriceGroup::where('business_id', $business_id)->find($id);

            return view('selling_price_group.edit')
                ->with(compact('spg'));

        }
    }

    public function update(Request $request, $id)
    {
        if (!Auth::user()->can('product.create')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $input = $request->only(['name', 'description']);
            $business_id = Auth::user()->business_id;

            $spg = SellingPriceGroup::where('business_id', $business_id)->findOrFail($id);
            $spg->name = $input['name'];
            $spg->description = $input['description'];
            $spg->save();

            return $this->respondSuccess($spg, "Berhasil merubah grup harga jual");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile() . "Line:" . $e->getLine() . "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal merubah grup harga jual");
        }
    }

    public function destroy($id)
    {
        if (!Auth::user()->can('product.create')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $business_id = Auth::user()->business_id;

            $spg = SellingPriceGroup::where('business_id', $business_id)->findOrFail($id);
            $spg->delete();

            return $this->respondSuccess(null, "Berhasil menghapus harga jual");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile() . "Line:" . $e->getLine() . "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menghapus grup harga jual");
        }
    }
}
```

1. *Outlet Controller*

```
<?php

namespace App\Http\Controllers;

use Auth;
use App\BusinessLocation;
use App\InvoiceLayout;
use App\InvoiceScheme;
```

```
use App\Account;
use App\AccountCategory;

use Yajra\DataTables\Facades\DataTables;
use Illuminate\Http\Request;
use Spatie\Permission\Models\Permission;

use App\Utils\Util;
use App\Utils\ModuleUtil;

class OutletController extends Controller
{
    protected $commonUtil;

    public function __construct(Util $commonUtil, ModuleUtil $moduleUtil)
    {
        $this->commonUtil = $commonUtil;
        $this->moduleUtil = $moduleUtil;
    }

    public function index(Request $request)
    {
        $business_id = Auth::user()->business_id;

        $locations = BusinessLocation::where('business_locations.business_id', $business_id)
            ->leftjoin(
                'invoice_schemes as ic',
                'business_locations.invoice_scheme_id',
                '=',
                'ic.id'
            )
            ->leftjoin(
                'invoice_layouts as il',
                'business_locations.invoice_layout_id',
                '=',
                'il.id'
            )
            ->select(['business_locations.name', 'location_id', 'landmark', 'city', 'zip_code',
'state',
                'country', 'business_locations.id', 'business_locations.is_default',
'business_locations.is_active', 'business_locations.mobile', 'business_locations.email',
'business_locations.website', 'ic.name as invoice_scheme', 'il.name as invoice_layout']);

        $permitted_locations = Auth::user()->permitted_locations();
        if ($permitted_locations != 'all') {
            $locations->whereIn('business_locations.id', $permitted_locations);
        }

        $start = 0;
        $limit = $request->input('limit') ? $request->input('limit') : 20;
        $paged = $request->input('page') ? $request->input('page') : 1;

        $start = ($paged - 1) * $limit;
        $locations = $locations->offset($start)->limit($limit)->get();

        return $this->respondArray($locations);
    }

    public function create()
    {
        if (!Auth::user()->can('business_settings.access')) {
            return $this->respondFailed("Tidak diizinkan");
        }
        $business_id = Auth::user()->business_id;

        //Check if subscribed or not, then check for location quota
        if (!$this->moduleUtil->isSubscribed($business_id)) {
            return $this->respondFailed("Masa aktif berlangganan anda telah berakhir");
        } elseif (!$this->moduleUtil->isQuotaAvailable('locations', $business_id)) {
            return $this->respondQuotaExpired("locations", $business_id);
        }

        $invoice_layouts = InvoiceLayout::where('business_id', $business_id)
                            ->first();
        $invoice_schemes = InvoiceScheme::where('business_id', $business_id)
                            ->first();

        $data = [
            'invoice_layouts' => $invoice_layouts,
            'invoice_schemes' => $invoice_schemes
        ];
```

```
        return $this->respondSuccess($data);
    }

    public function store(Request $request)
    {
        if (!Auth::user()->can('business_settings.access')) {
            return $this->respondFailed("Tidak diizinkan");
        }

        try {
            $business_id = Auth::user()->business_id;

            $invoice_layout_id = InvoiceLayout::where('business_id', $business_id)->first()->id;
            $invoice_scheme_id = InvoiceScheme::where('business_id', $business_id)->first()->id;

            //Check if subscribed or not, then check for location quota
            if (!$this->moduleUtil->isSubscribed($business_id)) {
                return $this->respondFailed("Masa aktif berlangganan anda telah berakhir");
            } elseif (!$this->moduleUtil->isQuotaAvailable('locations', $business_id)) {
                return $this->respondQuotaExpired("locations", $business_id);
            }

            $input = $request->only(['name', 'landmark', 'city', 'state', 'country', 'zip_code',
'mobile', 'alternate_number', 'email', 'website', 'custom_field1', 'custom_field2',
'custom_field3', 'custom_field4', 'location_id']);

            $input['invoice_scheme_id'] = $invoice_scheme_id;
            $input['invoice_layout_id'] = $invoice_layout_id;
            $input['business_id'] = $business_id;
            $input['is_default'] = 0;
            $input['is_active'] = 1;
            //Update reference count
            $ref_count = $this->commonUtil->setAndGetReferenceCount('business_location',
$business_id);

            if (empty($input['location_id'])) {
                $input['location_id'] = $this->commonUtil-
>generateReferenceNumber('business_location', $ref_count);
            }

            $location = BusinessLocation::create($input);

            //Create a new permission related to the created location
            Permission::create(['guard_name' => 'web', 'name' => 'location.' . $location->id ]);

            $accountCategory = AccountCategory::find(1);
            $accountTotal = Account::where('account_categories_id', 1)->where('business_id',
$business_id)->count();
            $account = [
                'business_id' => $business_id,
                'outlet_id' => $location->id,
                'name' => "Kas " . $location->name,
                'account_type' => 'saving_current',
                'is_default' => 1,
                "is_digital" => 0,
                "is_closed" => 0,
                'type' => 'cash',
                "note" => null,
                'created_by' => Auth::id(),
                'account_categories_id' => $accountCategory->id,
                'account_number' => $accountCategory->code  . sprintf('%04d', $accountTotal+1),
                "deleted_at" => null,
                'created_at' => date("Y-m-d h:i:s"),
                "updated_at" => date("Y-m-d h:i:s")
            ];

            Account::create($account);

            return $this->respondSuccess($location, "Outlet berhasil dibuat");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed($e->getMessage());
        }
    }

    public function update(Request $request, $id)
    {
        if (!Auth::user()->can('business_settings.access')) {
```

```
            return $this->respondFailed("Tidak diizinkan");
        }

        try {
            $business_id = Auth::user()->business_id;
            $invoice_layout_id = InvoiceLayout::where('business_id', $business_id)->first()->id;
            $invoice_scheme_id = InvoiceScheme::where('business_id', $business_id)->first()->id;

            $input = $request->only(['name', 'landmark', 'city', 'state', 'country', 'zip_code',
'mobile', 'alternate_number', 'email', 'website', 'custom_field1', 'custom_field2',
'custom_field3', 'custom_field4', 'location_id']);

            $input['invoice_scheme_id'] = $invoice_scheme_id;
            $input['invoice_layout_id'] = $invoice_layout_id;
            $input['is_default'] = 0;
            $input['is_active'] = 1;

            $location = BusinessLocation::findOrFail($id);
            $location->update($input);

            return $this->respondSuccess($location, "Outlet berhasil diupdate");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed($e->getMessage());
        }
    }

    public function destroy($id)
    {
        try {
            $me = Auth::user();
            $business_id = $me->business_id;

            BusinessLocation::where('business_id', $business_id)
                ->where('id', $id)->delete();

            return $this->respondSuccess(null, "Outlet berhasil dihapus");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed($e->getMessage());
        }
    }

    public function set_active($id, Request $request)
    {
        if (!Auth::user()->can('business_settings.access')) {
            return $this->respondFailed("Tidak diizinkan");
        }

        try {
            $is_active = $request->get('is_active', 1);

            $business_id = Auth::user()->business_id;

            BusinessLocation::where('business_id', $business_id)
                            ->where('is_default', '!=', 1)
                            ->where('id', $id)
                            ->update(['is_active' => $is_active]);


            return $this->respondSuccess(null, "Outlet berhasil diupdate");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Outlet gagal diupdate");
        }
    }

    public function checkLocationId(Request $request)
    {
        $location_id = $request->input('location_id');

        $valid = 'true';
        if (!empty($location_id)) {
            $business_id = $request->session()->get('user.business_id');
            $hidden_id = $request->input('hidden_id');
```

```
            $query = BusinessLocation::where('business_id', $business_id)
                                ->where('location_id', $location_id);
            if (!empty($hidden_id)) {
                $query->where('id', '!=', $hidden_id);
            }
            $count = $query->count();
            if ($count > 0) {
                $valid = 'false';
            }
        }
        echo $valid;
        exit;
    }
}
```

m. *Expense Controller*

```
<?php

namespace App\Http\Controllers;

use App\AccountTransaction;

use App\BusinessLocation;
use App\ExpenseCategory;
use App\Transaction;
use App\User;
use App\Account;
use App\Utils\ModuleUtil;

use App\Events\TransactionPaymentAdded;
use App\Events\TransactionPaymentDeleted;

use App\Events\TransactionPaymentUpdated;
use App\TransactionPayment;

use App\Utils\AccountUtil;
use App\Utils\TransactionUtil;

use DB;
use Auth;
use Illuminate\Http\Request;

use Yajra\DataTables\Facades\DataTables;

class ExpenseController extends Controller
{

    public function __construct(TransactionUtil $transactionUtil, ModuleUtil $moduleUtil)
    {
        $this->transactionUtil = $transactionUtil;
        $this->moduleUtil = $moduleUtil;
    }

    public function index(Request $request)
    {

        $business_id = Auth::user()->business_id;

        $expenses = Transaction::leftJoin('expense_categories AS ec',
'transactions.expense_category_id', '=', 'ec.id')
                    ->join(
                        'business_locations AS bl',
                        'transactions.location_id',
                        '=',
                        'bl.id'
                    )
                    ->leftJoin('users AS U', 'transactions.expense_for', '=', 'U.id')
                    ->leftJoin(
                        'transaction_payments AS TP',
                        'transactions.id',
                        '=',
                        'TP.transaction_id'
                    )
                    ->where('transactions.business_id', $business_id)
                    ->where('transactions.type', 'expense')
                    ->select(
                        'transactions.id',
                        'transactions.document',
                        'transaction_date',
```

```
                        'ref_no',
                        'ec.name as category',
                        DB::raw('CONVERT(IFNULL( ec.id, 0), UNSIGNED INTEGER) as category_id'),
                        'payment_status',
                        'additional_notes',
                        'final_total',
                        'bl.name as location_name',
                        DB::raw('CONVERT(IFNULL( bl.id, 0), UNSIGNED INTEGER) as location_id'),
                        DB::raw('SUM(TP.amount) as amount_paid'),
                        DB::raw('(final_total - SUM(TP.amount)) as payment_due')
                    )
                    ->groupBy('transactions.id')->orderBy('transaction_date', 'desc');

        //Add condition for expense for,used in sales representative expense report & list of
expense
        if ($request->input('expense_for')) {
            $expense_for = $request->input('expense_for');
            if (!empty($expense_for)) {
                $expenses->where('transactions.expense_for', $expense_for);
            }
        }

        $location_id = null;
        if ($request->input('location_id')) {
            $location_id = $request->input('location_id');
            $expenses->where('transactions.location_id', $location_id);
        }

        //Add condition for expense category, used in list of expense,
        if ($request->input('expense_category_id')) {
            $expense_category_id = $request->input('expense_category_id');
            $expenses->where('transactions.expense_category_id', $expense_category_id);
        }

        //Add condition for start and end date filter, uses in sales representative expense
report & list of expense
        if ($request->input('start_date') && $request->input('end_date')) {
            $start = $request->input('start_date');
            $end =  $request->input('end_date');
            $expenses->whereDate('transaction_date', '>=', $start)
                    ->whereDate('transaction_date', '<=', $end);
        }

        //Add condition for expense category, used in list of expense,
        if ($request->input('expense_category_id')) {
            $expense_category_id = $request->input('expense_category_id');
            $expenses->where('transactions.expense_category_id', $expense_category_id);
        }

        $permitted_locations = Auth::user()->permitted_locations();
        if ($permitted_locations != 'all') {
            $expenses->whereIn('transactions.location_id', $permitted_locations);
        }

        //Add condition for payment status for the list of expense
        if ($request->input('payment_status')) {
            $payment_status = $request->input('payment_status');
            $expenses->where('transactions.payment_status', $payment_status);
        }

        $start = 0;
        $limit = $request->input('limit') ? $request->input('limit') : 20;
        $paged = $request->input('page') ? $request->input('page') : 1;

        $start = ($paged - 1) * $limit;
        $expenses = $expenses->offset($start)->limit($limit)->get();

        return $this->respondArray($expenses);
    }

    public function categories(){
        $business_id = Auth::user()->business_id;

        $expense_categories = ExpenseCategory::where('business_id', $business_id)-
>where('is_default', 0)
                                ->select(['name', 'id'])
                                ->get();

        return $this->respondArray($expense_categories);
    }
```

```
    public function addCategory(Request $request){
        try {
            $input['name'] = $request->input('name');
            $input['business_id'] = Auth::user()->business_id;

            $ec = ExpenseCategory::create($input);

            return $this->respondSuccess($ec, "Tambah kategori biaya berhasil dilakukan");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Tambah kategori biaya tidak dapat dilakukan");
        }
    }

    public function updateCategory(Request $request, $id){
        try {
            $business_id = Auth::user()->business_id;

            $expense_category = ExpenseCategory::where('business_id', $business_id)-
>findOrFail($id);
            $expense_category->name = $request->input('name');
            $expense_category->save();

            return $this->respondSuccess($expense_category, "Update kategori biaya berhasil
dilakukan");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Update kategori biaya tidak dapat dilakukan");
        }
    }

    public function deleteCategory($id){
        try {
            $business_id = Auth::user()->business_id;

            $expense_category = ExpenseCategory::where('business_id', $business_id)-
>findOrFail($id);

            $transaction = Transaction::where('expense_category_id', $id)->get();

            if (count($transaction) > 0) {
                return $this->respondFailed("Kategori tidak bisa dihapus, sudah ada biaya yang
dimasukkan");
            } else {
                $expense_category->delete();

                return $this->respondSuccess($expense_category, "Kategori biaya berhasil
dihapus");
            }
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Kategori biaya tidak dapat dihapus");
        }
    }

    public function create()
    {
        $business_id = Auth::user()->business_id;

        //Check if subscribed or not
        if (!$this->moduleUtil->isSubscribed($business_id)) {
            return $this->respondFailed("User subscription was expired");
        }

        $ref_count = $this->transactionUtil->setAndGetReferenceCount('expense');
        $ref_no = $this->transactionUtil->generateReferenceNumber('expense', $ref_count);

        $expense_categories = ExpenseCategory::where('business_id', $business_id)-
>where('is_default', 0)
                                ->select(['name', 'id'])
                                ->get();

        $response = [
            'expense_categories' => $expense_categories,
            'ref_no' => $ref_no
```

```
        ];

        return $this->respondSuccess($response);
    }

    public function store(Request $request)
    {

        try {
            $business_id = Auth::user()->business_id;

            $transaction_data = $request->only([ 'ref_no', 'transaction_date', 'location_id',
'final_total', 'expense_for', 'additional_notes', 'expense_category_id']);

            $user_id = Auth::user()->id;
            $transaction_data['business_id'] = $business_id;
            $transaction_data['created_by'] = $user_id;
            $transaction_data['type'] = 'expense';
            $transaction_data['status'] = 'final';
            $transaction_data['payment_status'] = 'paid';
            $transaction_data['expense_category_id'] =
!empty($transaction_data['expense_category_id']) ? $transaction_data['expense_category_id'] :
null;
            $transaction_data['transaction_date'] = $this->transactionUtil-
>uf_date($transaction_data['transaction_date'], true);
            $transaction_data['final_total'] = $this->transactionUtil->num_uf(
                $transaction_data['final_total']
            );
            $transaction_data['total_before_tax'] = $transaction_data['final_total'];

            //Update reference count
            $ref_count = $this->transactionUtil->setAndGetReferenceCount('expense');
            //Generate reference number
            if (empty($transaction_data['ref_no'])) {
                $transaction_data['ref_no'] = $this->transactionUtil-
>generateReferenceNumber('expense', $ref_count);
            }

            $getBalance = DB::select(
                'SELECT SUM(
                    IF(AT.type="credit", AT.amount, -1 * AT.amount)
                ) as balance from account_transactions as AT
                WHERE AT.deleted_at IS NULL
                AND AT.account_id = '.Account::where('outlet_id', $request-
>input('location_id'))->first()->id.';'
            )[0];
            $balance = $getBalance->balance ? $getBalance->balance : 0;

            if($balance < $transaction_data['final_total']) {
                return $this->respondFailed('Saldo akun keuangan tidak mencukupi');
            }

                                                $transaction_data['document'] = '';

            DB::beginTransaction();

            $transaction = Transaction::create($transaction_data);

            $tp_data['paid_on'] = $transaction_data['transaction_date'];
            $tp_data['transaction_id'] = $transaction->id;
            $tp_data['amount'] = $transaction_data['final_total'];
            $tp_data['created_by'] = Auth::user()->id;
            $tp_data['payment_for'] = null;
            $tp_data['account_id'] = Account::where('outlet_id', $request->input('location_id'))-
>first()->id;
            $tp_data['note'] = $request->input('additional_notes');

            $ref_count = $this->transactionUtil->setAndGetReferenceCount('expense_payment');
            $tp_data['payment_ref_no'] = $this->transactionUtil-
>generateReferenceNumber('expense_payment', $ref_count);

            $tp_data['business_id'] = $business_id;

            $tp = TransactionPayment::create($tp_data);

            //update payment status
            $this->transactionUtil->updatePaymentStatus($transaction->id, $transaction-
>final_total);
            $tp_data['transaction_type'] = $transaction->type;
            //event(new TransactionPaymentAdded($tp, $tp_data));
            $account_transaction_data = [
```

```
                'amount' => $tp_data['amount'],
                'account_id' => $tp_data['account_id'],
                'type' =>
AccountTransaction::getAccountTransactionType($tp_data['transaction_type']),
                'operation_date' => $tp->paid_on,
                'created_by' => $tp->created_by,
                'transaction_id' => $tp->transaction_id,
                'transaction_payment_id' =>  $tp->id
            ];
            AccountTransaction::createAccountTransaction($account_transaction_data);

            $neraca = $this->transactionUtil->getBalanceSheet();
            $transaction->neraca = $neraca;
            $transaction->location_name = $transaction->location->name;
            $transaction->category = $transaction->expenseCategory->name;

            DB::commit();

            activity('transaction')
                ->causedBy(Auth::user())
                ->performedOn($transaction)
                ->withProperties($transaction)
                ->log('created');

            unset($transaction->neraca);
            unset($transaction->document);
            unset($transaction->business_id);
            unset($transaction->updated_at);

            return $this->respondSuccess($transaction, "Berhasil menambahkan biaya");
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());
            dd($e);

            return $this->respondFailed("Gagal menambahkan biaya");
        }
    }

    public function edit($id)
    {
        if (!Auth::user()->can('expense.access')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = request()->session()->get('user.business_id');

        //Check if subscribed or not
        if (!$this->moduleUtil->isSubscribed($business_id)) {
            return $this->moduleUtil->expiredResponse(action('ExpenseController@index'));
        }

        $business_locations = BusinessLocation::forDropdown($business_id);

        $expense_categories = ExpenseCategory::where('business_id', $business_id)
                                ->pluck('name', 'id');
        $expense = Transaction::where('business_id', $business_id)
                                ->where('id', $id)
                                ->first();
        $users = User::forDropdown($business_id, true, true);

        return view('expense.edit')
            ->with(compact('expense', 'expense_categories', 'business_locations', 'users'));
    }

    public function update(Request $request, $id)
    {
        try {
            $transaction_data = $request->only([ 'ref_no', 'transaction_date', 'location_id',
'final_total', 'expense_for', 'additional_notes', 'expense_category_id']);

            $business_id = Auth::user()->business_id;

            $transaction_data['expense_category_id'] =
!empty($transaction_data['expense_category_id']) ? $transaction_data['expense_category_id'] :
null;
            // $transaction_data['transaction_date'] = $this->transactionUtil-
>uf_date($transaction_data['transaction_date'], true);
            $transaction_data['final_total'] = $this->transactionUtil->num_uf(
                $transaction_data['final_total']
```

```
            );
            $transaction_data['total_before_tax'] = $transaction_data['final_total'];

            $transaction = Transaction::findOrFail($id);
            $transaction->update($transaction_data);

            $paymentLines = $transaction->payment_lines()->latest()->first();
            if($paymentLines) {
                $paymentLines->amount = $transaction_data['final_total'];
                $paymentLines->save();
                //event(new TransactionPaymentUpdated($paymentLines, $transaction->type));
                AccountTransaction::updateAccountTransaction($paymentLines, $transaction->type);
            }

            $neraca = $this->transactionUtil->getBalanceSheet();
            $transaction->neraca = $neraca;

            activity('transaction')
                ->causedBy(Auth::user())
                ->performedOn($transaction)
                ->withProperties($transaction)
                ->log('updated');

            return $this->respondSuccess($transaction, "Berhasil mengupdate biaya");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());
            dd($e);
            return $this->respondFailed("Gagal mengupdate biaya");
        }
    }

    public function destroy($id)
    {
        try {
            $business_id = Auth::user()->business_id;

            $expense = Transaction::where('business_id', $business_id)
                                  ->where('type', 'expense')
                                  ->where('id', $id)
                                  ->first();
            $expense->delete();

            //Delete account transactions
            AccountTransaction::where('transaction_id', $expense->id)->delete();

            $neraca = $this->transactionUtil->getBalanceSheet();
            $expense->neraca = $neraca;

            activity('transaction')
                ->causedBy(Auth::user())
                ->performedOn($expense)
                ->withProperties($expense)
                ->log('deleted');

            return $this->respondSuccess(null, "Berhasil menghapus biaya");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menghapus biaya");
        }
    }
}
```

n. *InvoiceLayout Controller*

```
<?php

namespace App\Http\Controllers;

use App\InvoiceLayout;

use Auth;
use App\BusinessLocation;
use App\Utils\Util;
use Illuminate\Http\Request;
use Validator;

class InvoiceLayoutController extends Controller
{
```

```
    protected $designs;
    protected $commonUtil;

    public function __construct(Util $commonUtil)
    {
        $this->commonUtil = $commonUtil;

        $this->designs = ['classic' => 'Classic',
                'elegant' => 'Elegant',
                'detailed' => 'Detailed',
                'columnize-taxes' => 'Columnize Taxes'
            ];
    }

    public function index($location_id = -1)
    {

        $invoice_layout = InvoiceLayout::where('business_id', Auth::user()->business_id)
                ->select(['id', 'header_text', 'footer_text', 'sub_heading_line1'])
                ->first();

        if($location_id > 0) {
            $outlet = BusinessLocation::where('business_id',Auth::user()->business_id)-
>find($location_id);
        } else {
            $outlet = BusinessLocation::where('business_id',Auth::user()->business_id)->first();
        }

        $data['id'] = $invoice_layout->id;
        $data['header_text'] = $invoice_layout->header_text;
        $data['footer_text'] = $invoice_layout->footer_text;
        $data['sub_heading_line1'] = $invoice_layout->sub_heading_line1;
        $data['outlet'] = $outlet;

        return $this->respondArray($data);
    }

    public function create()
    {
        if (!auth()->user()->can('invoice_settings.access')) {
            abort(403, 'Unauthorized action.');
        }

        $designs = $this->designs;

        return view('invoice_layout.create')->with(compact('designs'));
    }

    public function store(Request $request)
    {
        if (!auth()->user()->can('invoice_settings.access')) {
            abort(403, 'Unauthorized action.');
        }

        try {
            $validator = Validator::make($request->all(), [
                'logo' => 'mimes:jpeg,gif,png|1000',
            ]);

            $input = $request->only(['name', 'header_text',
                'invoice_no_prefix', 'invoice_heading', 'sub_total_label', 'discount_label',
'tax_label', 'total_label', 'highlight_color', 'footer_text', 'invoice_heading_not_paid',
'invoice_heading_paid', 'total_due_label', 'customer_label', 'paid_label', 'sub_heading_line1',
'sub_heading_line2',
                    'sub_heading_line3', 'sub_heading_line4', 'sub_heading_line5',
                    'table_product_label', 'table_qty_label', 'table_unit_price_label',
                    'table_subtotal_label', 'client_id_label', 'date_label', 'quotation_heading',
'quotation_no_prefix', 'design', 'client_tax_label', 'cat_code_label', 'cn_heading',
'cn_no_label', 'cn_amount_label', 'sales_person_label', 'prev_bal_label', 'date_time_format']);

            $business_id = Auth::user()->business_id;
            $input['business_id'] = $business_id;

            //Set value for checkboxes
            $checkboxes = ['show_business_name', 'show_location_name', 'show_landmark',
'show_city', 'show_state', 'show_country', 'show_zip_code', 'show_mobile_number',
'show_alternate_number', 'show_email', 'show_tax_1', 'show_tax_2', 'show_logo', 'show_barcode',
'show_payments', 'show_customer', 'show_client_id',
                'show_brand', 'show_sku', 'show_cat_code', 'show_sale_description',
'show_sales_person', 'show_expiry', 'show_lot', 'show_previous_bal', 'show_image',
'show_reward_point'];
```

```
            foreach ($checkboxes as $name) {
                $input[$name] = !empty($request->input($name)) ? 1 : 0;
            }

            //Upload Logo
            $logo_name = $this->commonUtil->uploadFile($request, 'logo', 'invoice_logos');
            if (!empty($logo_name)) {
                $input['logo'] = $logo_name;
            }

            if (!empty($request->input('is_default'))) {
                //get_default
                $default = InvoiceLayout::where('business_id', $business_id)
                            ->where('is_default', 1)
                            ->update(['is_default' => 0 ]);
                $input['is_default'] = 1;
            }

            //Module info
            if ($request->has('module_info')) {
                $input['module_info'] = json_encode($request->input('module_info'));
            }

            if (!empty($request->input('table_tax_headings'))) {
                $input['table_tax_headings'] = json_encode($request->input('table_tax_headings'));
            }
            $input['product_custom_fields'] = !empty($request->input('product_custom_fields')) ? $request->input('product_custom_fields') : null;
            $input['contact_custom_fields'] = !empty($request->input('contact_custom_fields')) ? $request->input('contact_custom_fields') : null;
            $input['location_custom_fields'] = !empty($request->input('location_custom_fields')) ? $request->input('location_custom_fields') : null;

            InvoiceLayout::create($input);
            $output = ['success' => 1,
                            'msg' => __("invoice.layout_added_success")
                        ];
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e->getMessage());

            $output = ['success' => 0,
                            'msg' => __("messages.something_went_wrong")
                        ];
        }

        return redirect('invoice-schemes')->with('status', $output);
    }

    public function edit($id)
    {
        if (!auth()->user()->can('invoice_settings.access')) {
            abort(403, 'Unauthorized action.');
        }

        $invoice_layout = InvoiceLayout::findOrFail($id);

        //Module info
        $invoice_layout->module_info = json_decode($invoice_layout->module_info, true);
        $invoice_layout->table_tax_headings = !empty($invoice_layout->table_tax_headings) ? json_decode($invoice_layout->table_tax_headings) : ['', '', '', ''];

        $designs = $this->designs;

        return view('invoice_layout.edit')
                ->with(compact('invoice_layout', 'designs'));
    }

    public function update(Request $request)
    {
        if (!Auth::user()->can('invoice_settings.access')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {

            $input = $request->only(['name', 'header_text',
                'invoice_no_prefix', 'invoice_heading', 'sub_total_label', 'discount_label', 'tax_label', 'total_label', 'highlight_color', 'footer_text', 'invoice_heading_not_paid',
```

```
'invoice_heading_paid', 'total_due_label', 'customer_label', 'paid_label', 'sub_heading_line1',
'sub_heading_line2',
                    'sub_heading_line3', 'sub_heading_line4', 'sub_heading_line5',
                    'table_product_label', 'table_qty_label', 'table_unit_price_label',
                    'table_subtotal_label', 'client_id_label', 'date_label', 'quotation_heading',
'quotation_no_prefix', 'design',
                    'client_tax_label', 'cat_code_label', 'cn_heading', 'cn_no_label',
'cn_amount_label',
                    'sales_person_label', 'prev_bal_label', 'date_time_format']);

            $business_id = Auth::user()->business_id;

            $invoice_layout = InvoiceLayout::where('business_id', Auth::user()->business_id)
                ->select(['id', 'header_text', 'footer_text', 'sub_heading_line1'])
                ->first();

            $invoice_layout->header_text = $input['header_text'];
            $invoice_layout->footer_text = $input['footer_text'];
            $invoice_layout->sub_heading_line1 = $input['sub_heading_line1'];
            $invoice_layout->save();


            return $this->respondSuccess($invoice_layout, 'Berhasil mengupdate struk');
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());
            dd($e);
            return $this->respondFailed('Gagal mengupdate struk');
        }
    }

}
```

o. *Account Controller*

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Http\Response;

use App\Account;
use App\AccountTransaction;
use App\AccountPayment;
use App\ManualConfirmation;
use App\TransactionPayment;

use Yajra\DataTables\Facades\DataTables;

use App\Utils\Util;
use App\Utils\AccountUtil;
use Carbon\Carbon;

use Auth;
use DB;

class AccountController extends Controller
{
    protected $commonUtil;
    protected $accountUtil;

    public function __construct(Util $commonUtil, AccountUtil $accountUtil)
    {
        $this->commonUtil = $commonUtil;
        $this->accountUtil = $accountUtil;
    }

    public function index(Request $request)
    {
        if (!Auth::user()->can('account.access')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;
        $countData = Account::where('business_id', $business_id)->count();


        $accounts = Account::leftjoin('account_transactions as AT', function ($join) {
            $join->on('AT.account_id', '=', 'accounts.id');
            $join->whereNull('AT.deleted_at');
```

```
        })
                            ->where('business_id', $business_id)
                            ->select(['name', 'account_number', 'is_default', 'accounts.note',
'accounts.id', 'accounts.type',
                                'is_closed', DB::raw("SUM( IF(AT.type='credit', amount, -
1*amount) ) as balance")])
                            ->groupBy('accounts.id');

        $account_type = $request->input('account_type');

        if ($account_type == 'capital') {
            $accounts->where('account_type', 'capital');
        } elseif ($account_type == 'other') {
            $accounts->where(function ($q) {
                $q->where('account_type', '!=', 'capital');
                $q->orWhereNull('account_type');
            });
        }

        $category_id = $request->input('category_id');
        if($category_id) {
            $accounts->where('accounts.account_categories_id', $category_id);
        }

        $digital = $request->input('digital');
        if($digital == "false") {
            $accounts->where('accounts.is_digital', '!=', 1);
        }

        $accounts = $accounts->get();

        foreach($accounts as $key => $acc) {
            $accounts[$key]->balance = ($acc->balance == null) ? 0 : $acc->balance;
        }

        return $this->respondArray($accounts);
    }

    public function getAccountCashBank() {
        $business_id = Auth::user()->business_id;

        $saldoKipa = $this->accountUtil->getBalanceAccount($business_id);

        $from_accounts = Account::where('business_id', $business_id)
            ->where('id', '!=', $saldoKipa)
            ->where('type', '!=','saldo')
            ->where('account_categories_id', 1)
            ->NotClosed()
            ->select('id','name')->get();

        return $this->respondArray($from_accounts);
    }

    public function getWallet(Request $request)
    {
        $business_id = Auth::user()->business_id;

        $wallet = Account::leftjoin('account_transactions as AT', function ($join) {
            $join->on('AT.account_id', '=', 'accounts.id');
            $join->whereNull('AT.deleted_at');
        })
                            ->where('business_id', $business_id)
                            ->where('accounts.type', 'saldo')
                            ->select(['name', 'account_number', 'is_default', 'is_digital',
'accounts.note', 'accounts.id',
                                'is_closed', DB::raw("SUM( IF(AT.type='credit', amount, -
1*amount) ) as balance")])
                            ->groupBy('accounts.id');

        $wallet = $wallet->first();

        if($wallet)
            $wallet->balance = ($wallet->balance == null) ? 0 : $wallet->balance;
        else
            $wallet['balance'] = 0;

        $saldoKipaId = $this->accountUtil->getBalanceAccount($business_id);
        $saldoBonus = $this->accountUtil->getSubBalance($saldoKipaId, 'bonus');
        $saldoUtama = $this->accountUtil->getSubBalance($saldoKipaId, 'topup');

        $wallet['main'] = (int)$saldoUtama;
```

```
        $wallet['bonus'] = (int)$saldoBonus;

        return $this->respondSuccess($wallet);
    }

    public function create()
    {
        if (!Auth::user()->can('account.access')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $account_types = Account::accountTypes();

        return $this->respondSuccess($account_types);
    }

    public function store(Request $request)
    {
        if (!Auth::user()->can('account.access')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $input = $request->only(['name', 'account_number', 'note', 'is_digital']);
            $business_id = Auth::user()->business_id;
            $user_id = Auth::user()->id;
            $input['business_id'] = $business_id;
            $input['created_by'] = $user_id;
            $input['account_type'] = 'saving_current';

            $account = Account::create($input);

            //Opening Balance
            $opening_bal = $request->input('opening_balance');

            if (!empty($opening_bal)) {
                $ob_transaction_data = [
                    'amount' =>$this->commonUtil->num_uf($opening_bal),
                    'account_id' => $account->id,
                    'type' => 'credit',
                    'sub_type' => 'opening_balance',
                    'operation_date' => \Carbon\Carbon::now(),
                    'created_by' => $user_id
                ];

                AccountTransaction::createAccountTransaction($ob_transaction_data);
            }

            return $this->respondSuccess($account, "Berhasil menambahkan akun");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menambahkan akun");
        }
    }

    public function show($id, Request $request)
    {
        if (!Auth::user()->can('account.access')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;


        $accounts = AccountTransaction::join(
            'accounts as A',
            'account_transactions.account_id',
            '=',
            'A.id'
        )
                        ->where('A.business_id', $business_id)
                        ->where('A.id', $id)
                        ->with(['transaction', 'transaction.contact', 'transfer_transaction'])
                        ->select(['type', 'amount', 'operation_date',
                            'sub_type', 'transfer_transaction_id',
                            DB::raw('(SELECT SUM(IF(AT.type="credit", AT.amount, -1 * AT.amount))
from account_transactions as AT WHERE AT.operation_date <= account_transactions.operation_date
AND AT.account_id  =account_transactions.account_id AND AT.deleted_at IS NULL) as balance'),
                            'transaction_id',
```

```
                            'account_transactions.id'
                            ])
                        ->groupBy('account_transactions.id')
                        ->orderBy('account_transactions.operation_date', 'desc');
        if (!empty($request->input('type'))) {
            $accounts->where('type', $request->input('type'));
        }

        $start_date = $request->input('start_date');
        $end_date = $request->input('end_date');

        if (!empty($start_date) && !empty($end_date)) {
            $accounts->whereBetween(DB::raw('date(operation_date)'), [$start_date, $end_date]);
        }

        $account = Account::where('business_id', $business_id)
                        ->find($id);

        $data = [
            'account' => $account,
            'transactions' => $accounts
        ];

        return $this->respondSuccess($data);
    }

    public function edit($id)
    {
        if (!Auth::user()->can('account.access')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;
        $account = Account::where('business_id', $business_id)
                        ->find($id);

        $account_types = Account::accountTypes();

        $data = [
            'account' => $account,
            'account_types' => $account_types
        ];

        return $this->respondSuccess($data);
    }

    public function update(Request $request, $id)
    {
        if (!Auth::user()->can('account.access')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $input = $request->only(['name', 'account_number', 'note']);

            $business_id = Auth::user()->business_id;
            $account = Account::where('business_id', $business_id)
                                ->findOrFail($id);
            $account->name = $input['name'];
            $account->account_number = $input['account_number'];
            $account->note = $input['note'];
            $account->save();

            return $this->respondSuccess($account, "Berhasil mengupdate akun");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());
            dd($e);
            return $this->respondFailed("Gagal mengupdate akun");
        }
    }

    public function destroyAccountTransaction($id)
    {
        if (!Auth::user()->can('account.access')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $business_id = Auth::user()->business_id;
```

```
            $account_transaction = AccountTransaction::findOrFail($id);

            if (in_array($account_transaction->sub_type, ['fund_transfer', 'deposit'])) {
                //Delete transfer transaction for fund transfer
                if (!empty($account_transaction->transfer_transaction_id)) {
                    $transfer_transaction = AccountTransaction::findOrFail($account_transaction-
>transfer_transaction_id);
                    $transfer_transaction->delete();
                }
                $account_transaction->delete();
            }

            return $this->respondSuccess($account, "Berhasil menghapus akun");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menghapus akun");
        }
    }

    public function close($id)
    {
        if (!Auth::user()->can('account.access')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $business_id = Auth::user()->business_id;

            $account = Account::where('business_id', $business_id)
                                                ->findOrFail($id);
            $account->is_closed = 1;
            $account->save();

            return $this->respondSuccess($account, "Berhasil menutup akun");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menutup akun");
        }
    }

    public function getFundTransfer($id)
    {
        if (!Auth::user()->can('account.access')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;

        $from_account = Account::where('business_id', $business_id)
                        ->NotClosed()
                        ->find($id);

        $to_accounts = Account::where('business_id', $business_id)
                        ->where('id', '!=', $id)
                        ->where('is_default', 0)
                        ->NotClosed()
                        ->select(['name', 'id'])
                        ->get();

        $data = [
            'from_account' => $from_account,
            'to_accounts' => $to_accounts
        ];

        return $this->respondSuccess($data);
    }

    public function postFundTransfer(Request $request)
    {
        if (!Auth::user()->can('account.access')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $business_id = Auth::user()->business_id;

            $amount = $this->commonUtil->num_uf($request->input('amount'));
```

```
            $from = $request->input('from_account');
            $to = $request->input('to_account');
            $note = $request->input('note');
            if (!empty($amount)) {
                $debit_data = [
                    'amount' => $amount,
                    'account_id' => $from,
                    'type' => 'debit',
                    'sub_type' => 'fund_transfer',
                    'created_by' => Auth::user()->id,
                    'note' => $note,
                    'transfer_account_id' => $to,
                    'operation_date' => $this->commonUtil->uf_date(Carbon::now()->format('m/d/Y
H:i'), true),
                ];

                DB::beginTransaction();
                $debit = AccountTransaction::createAccountTransaction($debit_data);

                $credit_data = [
                        'amount' => $amount,
                        'account_id' => $to,
                        'type' => 'credit',
                        'sub_type' => 'fund_transfer',
                        'created_by' => Auth::user()->id,
                        'note' => $note,
                        'transfer_account_id' => $from,
                        'transfer_transaction_id' => $debit->id,
                        'operation_date' => $this->commonUtil->uf_date(Carbon::now()-
>format('m/d/Y H:i'), true),
                    ];

                $credit = AccountTransaction::createAccountTransaction($credit_data);

                $debit->transfer_transaction_id = $credit->id;
                $debit->save();
                DB::commit();
            }

            return $this->respondSuccess(null, "Berhasil menambah deposit");
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menambah deposit ".$e->getMessage());
        }
    }

    public function getDeposit($id = -1)
    {

        $business_id = Auth::user()->business_id;

        if($id > 0){
            $account = Account::where('business_id', $business_id)
                        ->NotClosed()
                        ->find($id)->id;
        } else {
            $account = $this->accountUtil->getBalanceAccount($business_id);
        }

        $from_accounts = Account::where('business_id', $business_id)
                            ->where('id', '!=', $account)
                            ->NotClosed()
                            ->select('id','name');

        if($id == -1) {
            $from_accounts->where('type', '!=','saldo')->where('account_categories_id', 1);
        }

        return $this->respondArray($from_accounts->get());
    }


    public function postAdjustment(Request $request)
    {
        if (!Auth::user()->can('account.adjust')) {
            return $this->respondFailed('Tidak diizinkan', 403);
        }
```

```
        try {
            $business_id = Auth::user()->business_id;

            $amount = $this->commonUtil->num_uf($request->input('amount'));
            $account_id = $request->input('account_id');
            $note = $request->input('note');

            $from_account = $request->input('from_account');

            $account = Account::where('business_id', $business_id)
                              ->findOrFail($account_id);

            if (!empty($amount)) {
                $credit_data = [
                    'amount' => $amount,
                    'account_id' => $account_id,
                    'type' => 'credit',
                    'sub_type' => 'deposit',
                    'operation_date' => $this->commonUtil->uf_date(Carbon::now()->format('m/d/Y
H:i'), true),
                    'created_by' => Auth::user()->id,
                    'note' => $note
                ];
                $credit = AccountTransaction::createAccountTransaction($credit_data);

                $debit_data = $credit_data;
                $debit_data['type'] = 'debit';
                $debit_data['account_id'] = $from_account;
                $debit_data['transfer_transaction_id'] = $credit->id;

                $debit = AccountTransaction::createAccountTransaction($debit_data);

                $credit->transfer_transaction_id = $debit->id;

                $credit->save();
            }

            return $this->respondSuccess(null, 'Jurnal penyesuaian berhasil dilakukan');

        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed('Jurnal penyesuaian tidak dapat dilakukan');
        }
    }

    public function postDeposit(Request $request)
    {
        if (!Auth::user()->can('account.access')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {


            $business_id = Auth::user()->business_id;

            $amount = $this->commonUtil->num_uf($request->input('amount'));
            $account_id = $request->input('account_id');
            $note = $request->input('note');

            $from_account = $request->input('from_account');

            $account = Account::where('business_id', $business_id)
                              ->findOrFail($account_id);


            if (!empty($amount)) {
                $transaction = [
                    'amount' => $amount,
                    'account_id' => $account_id,
                    'sub_type' => 'deposit',
                    'operation_date' => $this->commonUtil->uf_date(Carbon::now()->format('m/d/Y
H:i'), true),
                    'created_by' => Auth::user()->id,
                    'note' => $note
                ];

                if ($account->category->type == 'activa') {
```

```
                    $transaction['type'] = 'credit';
                } else {
                    $transaction['type'] = 'debit';
                }

                $transaction = AccountTransaction::createAccountTransaction($transaction);
            }

            return $this->respondSuccess($account, "Berhasil menambah deposit");
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menambah deposit");
        }

        return $output;
    }

    public function postPayment(Request $request)
    {

        try {
            $business_id = Auth::user()->business_id;

            $amount = $this->commonUtil->num_uf($request->input('amount'));
            $account_id = $this->accountUtil->getBalanceAccount($business_id);
            $from_account = $request->input('from_account');
            $note = $request->input('note');
            $order_id = $request->input('order_id');
            $status = $request->input('status');

            $account = Account::where('business_id', $business_id)
                            ->findOrFail($account_id);

            if (!empty($amount)) {
                $accountPayment = AccountPayment::create([
                    'account_id' => $account_id,
                    'amount' => $amount,
                    'status' => $status,
                    'from_account' => $from_account,
                    'order_id' => $order_id,
                    'user_id' => Auth::user()->id,
                    'note' => $note
                ]);

                if ($status == 'approved') {
                    $credit_data = [
                        'amount' => $amount,
                        'account_id' => $account_id,
                        'type' => 'credit',
                        'sub_type' => 'deposit',
                        'operation_date' => $this->commonUtil->uf_date(date('m/d/Y h:i:s'),
true),
                        'created_by' => Auth::user()->id,
                        'note' => $note
                    ];
                    $credit = AccountTransaction::createAccountTransaction($credit_data);

                    if ($from_account != null) {
                        $debit_data = $credit_data;
                        $debit_data['account_id'] = $from_account;
                        $debit_data['type'] = 'debit';

                        $debit = AccountTransaction::createAccountTransaction($debit_data);
                    }
                }

                if(env('MIDTRANS_STATUS') == false) {
                    ManualConfirmation::create([
                        'business_id' => $business_id,
                        'type' => 'saldo',
                        'no_rek' => $request->input('no_rek'),
                        'name' => $request->input('name'),
                        'account_payment_id' => $accountPayment->id
                    ]);
                }
            }

            return $this->respondSuccess($account, "Berhasil menambah deposit");
```

```
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal menambah deposit");
        }

        return redirect()->back();
    }

    public function getAccountBalance($id)
    {
        if (!Auth::user()->can('account.access')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;
        $account = Account::leftjoin(
            'account_transactions as AT',
            'AT.account_id',
            '=',
            'accounts.id'
        )
            ->whereNull('AT.deleted_at')
            ->where('accounts.business_id', $business_id)
            ->where('accounts.id', $id)
            ->select('accounts.*', DB::raw("SUM( IF(AT.type='credit', amount, -1 * amount) ) as
balance"))
            ->first();

        return $account;
    }

    public function cashFlow(Request $request)
    {

        $business_id = Auth::user()->business_id;
        $accountId = $request->input('account_id');

        if($accountId == -1) {
            $accountId = $this->accountUtil->getBalanceAccount($business_id);
        }


        $accounts = AccountTransaction::join(
                'accounts as A',
                'account_transactions.account_id',
                '=',
                'A.id'
            )
            ->leftjoin('transaction_payments as TP',
'account_transactions.transaction_payment_id',
                '=',
                'TP.id'
            )
            ->where('A.business_id', $business_id)
            ->with(['transaction', 'transaction.contact', 'transfer_transaction'])
            ->select(['account_transactions.type', 'account_transactions.amount',
'operation_date',
                'account_transactions.sub_type', 'transfer_transaction_id',
                DB::raw('(SELECT SUM(IF(AT.type="credit", AT.amount, -1 * AT.amount)) from
account_transactions as AT WHERE AT.operation_date <= account_transactions.operation_date AND
AT.deleted_at IS NULL and AT.account_id = account_transactions.account_id) as balance'),
                'account_transactions.transaction_id',
                'account_transactions.id',
                'A.name as account_name',
                'account_transactions.note as note',
                'TP.payment_ref_no as payment_ref_no'
                ])
             ->groupBy('account_transactions.id')
             ->orderBy('account_transactions.operation_date', 'desc');

        if (!empty($request->input('type'))) {
            $accounts->where('A.type', $request->input('type'));
        }

        if (!empty($request->input('account_id'))) {
            $accounts->where('A.id', $accountId);
        }
```

```
        $start_date = $request->input('start_date');
        $end_date = $request->input('end_date');

        if ((!empty($start_date) || $start_date == "") && (!empty($end_date) && $end_date == ""))
{
            $accounts->whereBetween(DB::raw('date(operation_date)'), [$start_date, $end_date]);
        }

        return $this->respondArray($accounts->get());
    }


    public function historyTopup(Request $request)
    {

        $business_id = Auth::user()->business_id;
        $accountId = $this->accountUtil->getBalanceAccount($business_id);


        $account = Account::where('business_id', $business_id)->find($accountId);

        return $this->respondArray($account->payment);
    }
}
```

p.*Product Controller*

```
<?php

namespace App\Http\Controllers;

use Auth;
use App\Brands;
use App\Business;
use App\BusinessLocation;
use App\Category;
use App\Media;
use App\Product;
use App\ProductVariation;
use App\PurchaseLine;
use App\SellingPriceGroup;
use App\TaxRate;
use App\Unit;
use App\Utils\ModuleUtil;
use App\Utils\ProductUtil;
use App\Variation;
use App\Transaction;

use App\VariationGroupPrice;
use App\VariationLocationDetails;
use App\VariationTemplate;
use Illuminate\Http\Request;

use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Storage;
use Yajra\DataTables\Facades\DataTables;

class ProductController extends Controller
{

    protected $productUtil;
    protected $moduleUtil;

    private $barcode_types;

    public function __construct(ProductUtil $productUtil, ModuleUtil $moduleUtil)
    {
        $this->productUtil = $productUtil;
        $this->moduleUtil = $moduleUtil;

        //barcode types
        $this->barcode_types = $this->productUtil->barcode_types();

        //Product types also includes modifier.
        //TODO: uncomment combo
        $this->product_types =
                        [
                            [
                                'value' => 'single',
                                'name' => 'Single'
                            ],
```

```
                            [
                                'value' => 'variable',
                                'name' => 'Variable'
                            ]
                        ];
    }

    public function index(Request $request)
    {
        $business_id = Auth::user()->business_id;

        $products = Product::leftJoin('brands', 'products.brand_id', '=', 'brands.id')
            ->join('units', 'products.unit_id', '=', 'units.id')
            ->leftJoin('categories as c1', 'products.category_id', '=', 'c1.id')
            ->leftJoin('categories as c2', 'products.sub_category_id', '=', 'c2.id')
            ->leftJoin('tax_rates', 'products.tax', '=', 'tax_rates.id')
            ->join('variations as v', 'v.product_id', '=', 'products.id')
            ->leftJoin('variation_location_details as vld', 'vld.variation_id', '=', 'v.id')
            ->where('products.business_id', $business_id)
            ->where('products.type', '!=', 'modifier')
            ->with(['product_variations' => function($query) {
                $query->select([
                    'id',
                    DB::raw('CONVERT(IFNULL( variation_template_id, 0), UNSIGNED INTEGER) as
variation_template_id'),
                    'name',
                    'product_id'
                ]);
                $query->with(['variations' => function($q) {
                    $q->select([
                        'id',
                        "name",
                        "product_id",
                        "sub_sku",
                        "product_variation_id",
                        DB::raw('CONVERT(IFNULL( variation_value_id, 0), UNSIGNED INTEGER) as
variation_value_id'),
                        "default_purchase_price",
                        "dpp_inc_tax",
                        "profit_percent",
                        "default_sell_price",
                        "sell_price_inc_tax"
                    ]);
                }]);
            }])
            ->select(
                'products.id',
                'products.name as product',
                'products.type',
                'c1.name as category',
                'c2.name as sub_category',
                DB::raw('CONVERT(IFNULL( c1.id, 0), UNSIGNED INTEGER) as category_id'),
                DB::raw('CONVERT(IFNULL( c2.id, 0), UNSIGNED INTEGER) as sub_category_id'),
                'units.actual_name as unit',
                'brands.name as brand',
                DB::raw('CONVERT(IFNULL( units.id, 0), UNSIGNED INTEGER) as unit_id'),
                DB::raw('CONVERT(IFNULL( brands.id, 0), UNSIGNED INTEGER) as brand_id'),
                'tax_rates.name as tax',
                'products.sku',
                'products.image',
                'products.enable_stock',
                'products.is_inactive',
                'products.not_for_selling',
                'products.product_type',
                DB::raw('CONVERT(IFNULL( products.alert_quantity, 0), UNSIGNED INTEGER) as
alert_quantity'),
                DB::raw('CONVERT(IFNULL( products.weight, 0), UNSIGNED INTEGER) as weight'),
                DB::raw('IFNULL(SUM(vld.qty_available), 0) as current_stock'),
                DB::raw('MAX(v.default_sell_price) as max_price'),
                DB::raw('MIN(v.default_sell_price) as min_price')
            )->groupBy('products.id');
            // ->orderBy('products.product_type', 'desc')


        $name = $request->input('name', null);
        if($name) {
            $products->where('products.name', 'LIKE', '%'.$name.'%');
        }

        $category_id = $request->input('category_id', null);
        if ($category_id) {
```

```
            $products->where('products.category_id', $category_id);
            $sub_category_id = $request->input('sub_category_id', null);
            if ($sub_category_id) {
                $products->where('products.sub_category_id', $sub_category_id);
            }
        }

        $brand_id = $request->input('brand_id', null);
        if ($brand_id) {
            $products->where('products.brand_id', $brand_id);
        }

        $unit_id = $request->input('unit_id', null);
        if ($unit_id) {
            $products->where('products.unit_id', $unit_id);
        }

        $product_type = $request->input('product_type', null);
        if ($product_type) {
            $products->where('products.product_type', $product_type);
        }

        if($product_type == "digital") {
            $products->orderBy('brands.name', 'ASC')
            ->orderBy('c2.name', 'DESC')
            ->orderBy('v.default_sell_price', 'ASC');
        } else {
            $products->orderBy('products.name', 'ASC');
        }

        $start = 0;
        $limit = $request->input('limit') ? $request->input('limit') : 20;
        $paged = $request->input('page') ? $request->input('page') : 1;

        $start = ($paged - 1) * $limit;
        $products = $products->offset($start)->limit($limit)->get();

        foreach($products as $key => $value) {
            $openStockTrx = Transaction::where('type', 'opening_stock')
                                ->where('business_id', $business_id)
                                ->where('opening_stock_product_id', $value->id)
                                ->first();
            $purchaseTrx = Transaction::where('type', 'purchase')
                                ->leftJoin('purchase_lines as PL', 'transactions.id', '=',
'PL.transaction_id')
                                ->where('business_id', $business_id)
                                ->where('PL.product_id', $value->id)
                                ->first();
            $adjustTrx = Transaction::where('type', 'stock_adjustment')
                                ->leftJoin('stock_adjustment_lines as SAL', 'transactions.id',
'=', 'SAL.transaction_id')
                                ->where('business_id', $business_id)
                                ->where('SAL.product_id', $value->id)
                                ->first();


            $value->enable_add_stock = (!$openStockTrx && !$purchaseTrx && !$adjustTrx);
        }

        return $this->respondArray($products);
    }

    public function store(Request $request)
    {
        if (!Auth::user()->can('product.create')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $business_id = Auth::user()->business_id;
            $form_fields = ['name', 'unit_id', 'tax', 'type', 'barcode_type', 'sku',
'alert_quantity', 'tax_type', 'weight', 'product_custom_field1', 'product_custom_field2',
'product_custom_field3', 'product_custom_field4', 'product_description', 'sub_unit_ids',
'product_type'];

            $product_details = $request->only($form_fields);

            if($product_details['sku']) {
                if(!$this->checkProductSku($product_details['sku'])) {
                    return $this->respondFailed("Produk dengan SKU " . $product_details['sku'] .
" sudah ada");
```

```
                }
            }

            $product_details['business_id'] = $business_id;
            $product_details['created_by'] = Auth::user()->id;

            $product_details['enable_stock'] = (!empty($request->input('enable_stock')) &&
$request->input('enable_stock') == 1) ? 1 : 0;
            $product_details['not_for_selling'] = (!empty($request->input('not_for_selling')) &&
$request->input('not_for_selling') == 1) ? 1 : 0;
            $product_details['alert_quantity'] = !empty($product_details['alert_quantity']) ?
$product_details['alert_quantity'] : 0;
            $product_details['product_type'] = !empty($product_details['product_type']) ?
$product_details['product_type'] : 'fisik';

            if (!empty($request->input('brand_id'))) {
                $product_details['brand_id'] = $request->input('brand_id') ;
            }

            if (!empty($request->input('category_id'))) {
                $product_details['category_id'] = $request->input('category_id') ;
            }

            if (!empty($request->input('sub_category_id'))) {
                $product_details['sub_category_id'] = $request->input('sub_category_id') ;
            }

            if (empty($product_details['sku'])) {
                $product_details['sku'] = ' ';
            }

            $expiry_enabled = Auth::user()->business->enable_product_expiry; //$request-
>session()->get('business.enable_product_expiry');
            if (!empty($request->input('expiry_period_type')) && !empty($request-
>input('expiry_period')) && !empty($expiry_enabled) && ($product_details['enable_stock'] == 1)) {
                $product_details['expiry_period_type'] = $request->input('expiry_period_type');
                $product_details['expiry_period'] = $this->productUtil->num_uf($request-
>input('expiry_period'));
            }

            if (!empty($request->input('enable_sr_no')) &&  $request->input('enable_sr_no') == 1)
{
                $product_details['enable_sr_no'] = 1 ;
            }

            //upload document
            if($request->hasFile('image')){
                $file_name = $this->productUtil->uploadFile($request, 'image',
config('constants.product_img_path'))->data;
                if (!empty($file_name)) {
                    $product_details['image'] = $file_name;
                }
            }

            DB::beginTransaction();

            $product = Product::create($product_details);

            if (empty(trim($request->input('sku')))) {
                $sku = $this->productUtil->generateProductSku($product->id);
                $product->sku = $sku;
                $product->save();
            }

            if ($product->type == 'single') {
                $variation_id = $this->productUtil->createSingleProductVariation($request,
$product->id, $product->sku, $request->input('single_dpp'), $request->input('single_dpp'),
$request->input('profit_percent'), $request->input('single_dsp'), $request->input('single_dsp'));
            } elseif ($product->type == 'variable') {
                if (!empty($request->input('product_variation'))) {
                    $input_variations = $request->input('product_variation');
                    // dd($input_variations);
                    $this->productUtil->createVariableProductVariations($request, $product->id,
$input_variations);
                }
            } elseif ($product->type == 'combo') {

                //Create combo_variations array by combining variation_id and quantity.
                $combo_variations = [];
                if (!empty($request->input('composition_variation_id'))) {
                    $composition_variation_id = $request->input('composition_variation_id');
```

```
                    $quantity = $request->input('quantity');
                    $unit = $request->input('unit');

                    foreach ($composition_variation_id as $key => $value) {
                        $combo_variations[] = [
                                'variation_id' => $value,
                                'quantity' => $quantity[$key],
                                'unit_id' => $unit[$key]
                            ];
                    }
                }

                $this->productUtil->createSingleProductVariation($request, $product->id,
$product->sku, $request->input('item_level_purchase_price_total'), $request-
>input('item_level_purchase_price_total'), $request->input('profit_percent'), $request-
>input('selling_price'), $request->input('selling_price'), $combo_variations);
            }

            //Add product racks details.
            $product_racks = $request->get('product_racks', null);
            if (!empty($product_racks)) {
                $this->productUtil->addRackDetails($business_id, $product->id, $product_racks);
            }

            $debug = Product::leftJoin('brands', 'products.brand_id', '=', 'brands.id')
                ->join('units', 'products.unit_id', '=', 'units.id')
                ->leftJoin('categories as c1', 'products.category_id', '=', 'c1.id')
                ->leftJoin('categories as c2', 'products.sub_category_id', '=', 'c2.id')
                ->leftJoin('tax_rates', 'products.tax', '=', 'tax_rates.id')
                ->join('variations as v', 'v.product_id', '=', 'products.id')
                ->leftJoin('variation_location_details as vld', 'vld.variation_id', '=', 'v.id')
                ->where('products.business_id', $business_id)
                ->where('products.id', $product->id)
                ->with(['product_variations' => function($query) {
                    $query->select([
                        'id',
                        'variation_template_id',
                        'name',
                        'product_id'
                    ]);
                    $query->with(['variations' => function($q) {
                        $q->select([
                            'id',
                            "name",
                            "product_id",
                            "sub_sku",
                            "product_variation_id",
                            "variation_value_id",
                            "default_purchase_price",
                            "dpp_inc_tax",
                            "profit_percent",
                            "default_sell_price",
                            "sell_price_inc_tax"
                        ]);
                    }]);
                }])
                ->select(
                    'products.id',
                    'products.name as product',
                    'products.type',
                    'c1.name as category',
                    'c2.name as sub_category',
                    'c1.id as category_id',
                    'c2.id as sub_category_id',
                    'units.actual_name as unit',
                    'brands.name as brand',
                    'units.id as unit_id',
                    'brands.id as brand_id',
                    'tax_rates.name as tax',
                    'products.sku',
                    'products.image',
                    'products.enable_stock',
                    'products.is_inactive',
                    'products.not_for_selling',
                    'products.product_type',
                    'products.weight',
                    DB::raw('CONVERT(IFNULL( products.alert_quantity, 0), UNSIGNED INTEGER) as
alert_quantity'),
                    DB::raw('IFNULL(SUM(vld.qty_available),0) as current_stock'),
                    DB::raw('MAX(v.default_sell_price) as max_price'),
                    DB::raw('MIN(v.default_sell_price) as min_price')
```

```
                )->groupBy('products.id')->first();


            foreach($debug->product_variations as $pkey => $pval) {
                $debug->product_variations[$pkey]->variation_template_id = $pval->variation_template_id ? $pval->variation_template_id : 0;
                foreach($pval->variations as $vkey => $vval) {
                    $debug->product_variations[$pkey]->variations[$vkey]->variation_value_id = $vval->variation_value_id ? $vval->variation_value_id : 0;
                }
            }
            $debug->weight = $debug->weight ? (double)$debug->weight : 0;
            $debug->category_id = $debug->category_id ? $debug->category_id : 0;
            $debug->sub_category_id = $debug->sub_category_id ? $debug->sub_category_id : 0;
            $debug->unit_id = $debug->unit_id ? $debug->unit_id : 0;
            $debug->brand_id = $debug->brand_id ? $debug->brand_id : 0;

            DB::commit();
            return $this->respondSuccess($debug, "Berhasil menambahkan produk");
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e->getMessage());

            return $this->respondFailed("Gagal menambahkan produk ".$e->getMessage());
        }
    }

    public function show($id)
    {
        if (!Auth::user()->can('product.view')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;
        $details = $this->productUtil->getRackDetails($business_id, $id, true);
        $products = Product::where('business_id', $business_id)
                        ->findOrFail($id);

        return $this->respondSuccess($products);
    }

    public function edit($id)
    {
        if (!Auth::user()->can('product.update')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;
        $categories = Category::forDropdown($business_id);
        $brands = Brands::where('business_id', $business_id)
                            ->select(['name', 'id'])
                            ->get();

        $tax_dropdown = TaxRate::forBusinessDropdown($business_id, true, true);
        $taxes = $tax_dropdown['tax_rates'];

        $barcode_types = $this->barcode_types;

        $product = Product::where('business_id', $business_id)
                            ->where('id', $id)
                            ->first();

        //Sub-category
        $sub_categories = Category::where('business_id', $business_id)
                        ->where('parent_id', $product->category_id)
                        ->select(['name', 'id'])
                        ->get();

        $default_profit_percent = Business::where('id', $business_id)->value('default_profit_percent');

        //Get units.
        $units = Unit::forDropdown($business_id, true);
        $sub_units = $this->productUtil->getSubUnits($business_id, $product->unit_id, true);

        //Get all business locations
        $business_locations = BusinessLocation::forDropdown($business_id);
        //Rack details
        $rack_details = $this->productUtil->getRackDetails($business_id, $id);
```

```
        $selling_price_group_count = SellingPriceGroup::countSellingPriceGroups($business_id);

        $product_types = $this->product_types;

        $data = [
            'categories' => $categories,
            'brands' => $brands,
            'units' => $units,
            'sub_units' => $sub_units,
            'taxes' => $taxes,
            'barcode_types' => $barcode_types,
            'product' => $product,
            'sub_categories' => $sub_categories,
            'default_profit_percent' => $default_profit_percent,
            'business_locations' => $business_locations,
            'rack_details' => $rack_details,
            'selling_price_group_count' => $selling_price_group_count,
            'product_types' => $product_types
        ];

        return $this->respondSuccess($data);
    }

    public function update($id, Request $request)
    {
        if (!Auth::user()->can('product.update')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        try {
            $business_id = Auth::user()->business_id;
            $product_details = $request->only(['name', 'unit_id', 'category_id', 'tax',
'barcode_type', 'sku', 'alert_quantity', 'tax_type', 'weight', 'product_custom_field1',
'product_custom_field2', 'product_custom_field3', 'product_custom_field4', 'product_description',
'sub_unit_ids']);

            DB::beginTransaction();

            $product = Product::where('business_id', $business_id)
                                ->where('id', $id)
                                ->with(['product_variations'])
                                ->first();
            $productAll = Product::where('created_by', Auth::user()->id)
                                ->where('id', '!=',$id)
                                ->where('name', $product->name)
                                ->with(['product_variations'])->get();

            $product->name = $product_details['name'];

            if (!empty($request->input('brand_id'))) {
                $product->brand_id = $request->input('brand_id') ;
            }

            if (!empty($request->input('category_id'))) {
                $product->category_id = $request->input('category_id') ;
            }

            if (!empty($request->input('sub_category_id'))) {
                $product->sub_category_id = $request->input('sub_category_id') ;
            }

            $product->unit_id = $product_details['unit_id'];
            $product->sku = $product_details['sku'];
            $product->alert_quantity = $product_details['alert_quantity'];
            $product->tax_type = 'exclusive';
            $product->weight = $product_details['weight'];
            $product->product_description = $product_details['product_description'];
            $product->sub_unit_ids = !empty($product_details['sub_unit_ids']) ?
$product_details['sub_unit_ids'] : null;

            if (!empty($request->input('enable_stock')) &&  $request->input('enable_stock') == 1)
{
                $product->enable_stock = 1;
            } else {
                $product->enable_stock = 0;
            }

            $product->not_for_selling = (!empty($request->input('not_for_selling')) &&  $request-
>input('not_for_selling') == 1) ? 1 : 0;
```

```
            $expiry_enabled = Auth::user()->business->enable_product_expiry; //$request-
>session()->get('business.enable_product_expiry');
            if (!empty($expiry_enabled)) {
                if (!empty($request->input('expiry_period_type')) && !empty($request-
>input('expiry_period')) && ($product->enable_stock == 1)) {
                    $product->expiry_period_type = $request->input('expiry_period_type');
                    $product->expiry_period = $this->productUtil->num_uf($request-
>input('expiry_period'));
                } else {
                    $product->expiry_period_type = null;
                    $product->expiry_period = null;
                }
            }

            if (!empty($request->input('enable_sr_no')) &&  $request->input('enable_sr_no') == 1)
{
                $product->enable_sr_no = 1;
            } else {
                $product->enable_sr_no = 0;
            }

            //upload document
            if($request->hasFile('image')) {
                $file_name = $this->productUtil->uploadFile($request, 'image',
config('constants.product_img_path'))->data;
                if (!empty($file_name)) {
                    $product->image = $file_name;
                }
            }

            $product->save();

            if ($product->type == 'single') {
                $single_data = $request->only(['single_variation_id', 'single_dpp',
'single_dpp_inc_tax', 'single_dsp_inc_tax', 'profit_percent', 'single_dsp']);
                $variation = Variation::find($single_data['single_variation_id']);

                $variation->sub_sku = $product->sku;
                $variation->default_purchase_price = $this->productUtil-
>num_uf($single_data['single_dpp']);
                $variation->dpp_inc_tax = $this->productUtil->num_uf($single_data['single_dpp']);
                $variation->profit_percent = $this->productUtil-
>num_uf($single_data['profit_percent']);
                $variation->default_sell_price = $this->productUtil-
>num_uf($single_data['single_dsp']);
                $variation->sell_price_inc_tax = $this->productUtil-
>num_uf($single_data['single_dsp']);
                $variation->save();
            } elseif ($product->type == 'variable') {
                //Update existing variations
                $input_variations_edit = $request->get('product_variation_edit');
                if (!empty($input_variations_edit)) {
                    $this->productUtil->updateVariableProductVariations($product->id,
$input_variations_edit);
                }

                //Add new variations created.
                $input_variations = $request->input('product_variation');
                if (!empty($input_variations)) {
                    $this->productUtil->createVariableProductVariations($request, $product->id,
$input_variations);
                }
            } elseif ($product->type == 'combo') {

                //Create combo_variations array by combining variation_id and quantity.
                $combo_variations = [];
                if (!empty($request->input('composition_variation_id'))) {
                    $composition_variation_id = $request->input('composition_variation_id');
                    $quantity = $request->input('quantity');
                    $unit = $request->input('unit');

                    foreach ($composition_variation_id as $key => $value) {
                        $combo_variations[] = [
                                'variation_id' => $value,
                                'quantity' => $quantity[$key],
                                'unit_id' => $unit[$key]
                            ];
                    }
                }

                $variation = Variation::find($request->input('combo_variation_id'));
```

```
                $variation->sub_sku = $product->sku;
                $variation->default_purchase_price = $this->productUtil->num_uf($request-
>input('item_level_purchase_price_total'));
                $variation->dpp_inc_tax = $this->productUtil->num_uf($request-
>input('purchase_price_inc_tax'));
                $variation->profit_percent = $this->productUtil->num_uf($request-
>input('profit_percent'));
                $variation->default_sell_price = $this->productUtil->num_uf($request-
>input('selling_price'));
                $variation->sell_price_inc_tax = $this->productUtil->num_uf($request-
>input('selling_price_inc_tax'));
                $variation->combo_variations = $combo_variations;
                $variation->save();
            }

            //Add product racks details.
            $product_racks = $request->get('product_racks', null);
            if (!empty($product_racks)) {
                $this->productUtil->addRackDetails($business_id, $product->id, $product_racks);
            }

            $product_racks_update = $request->get('product_racks_update', null);
            if (!empty($product_racks_update)) {
                $this->productUtil->updateRackDetails($business_id, $product->id,
$product_racks_update);
            }

            $debug = Product::leftJoin('brands', 'products.brand_id', '=', 'brands.id')
                ->join('units', 'products.unit_id', '=', 'units.id')
                ->leftJoin('categories as c1', 'products.category_id', '=', 'c1.id')
                ->leftJoin('categories as c2', 'products.sub_category_id', '=', 'c2.id')
                ->leftJoin('tax_rates', 'products.tax', '=', 'tax_rates.id')
                ->join('variations as v', 'v.product_id', '=', 'products.id')
                ->leftJoin('variation_location_details as vld', 'vld.variation_id', '=', 'v.id')
                ->where('products.business_id', $business_id)
                ->where('products.id', $id)
                ->with(['product_variations' => function($query) {
                    $query->select([
                        'id',
                        'variation_template_id',
                        'name',
                        'product_id'
                    ]);
                    $query->with(['variations' => function($q) {
                        $q->select([
                            'id',
                            "name",
                            "product_id",
                            "sub_sku",
                            "product_variation_id",
                            "variation_value_id",
                            "default_purchase_price",
                            "dpp_inc_tax",
                            "profit_percent",
                            "default_sell_price",
                            "sell_price_inc_tax"
                        ]);
                    }]);
                }])
                ->select(
                    'products.id',
                    'products.name as product',
                    'products.type',
                    'c1.name as category',
                    'c2.name as sub_category',
                    'c1.id as category_id',
                    'c2.id as sub_category_id',
                    'units.actual_name as unit',
                    'brands.name as brand',
                    'units.id as unit_id',
                    'brands.id as brand_id',
                    'tax_rates.name as tax',
                    'products.sku',
                    'products.image',
                    'products.enable_stock',
                    'products.is_inactive',
                    'products.not_for_selling',
                    'products.product_type',
                    'products.weight',
                    DB::raw('CONVERT(IFNULL( products.alert_quantity, 0), UNSIGNED INTEGER) as
alert_quantity'),
```

```
                    DB::raw('IFNULL(SUM(vld.qty_available),0) as current_stock'),
                    DB::raw('MAX(v.default_sell_price) as max_price'),
                    DB::raw('MIN(v.default_sell_price) as min_price')
                )->groupBy('products.id')->first();


            foreach($debug->product_variations as $pkey => $pval) {
                $debug->product_variations[$pkey]->variation_template_id = $pval-
>variation_template_id ? $pval->variation_template_id : 0;
                foreach($pval->variations as $vkey => $vval) {
                    $debug->product_variations[$pkey]->variations[$vkey]->variation_value_id =
$vval->variation_value_id ? $vval->variation_value_id : 0;
                }
            }
            $debug->weight = $debug->weight ? (double)$debug->weight : 0;
            $debug->category_id = $debug->category_id ? $debug->category_id : 0;
            $debug->sub_category_id = $debug->sub_category_id ? $debug->sub_category_id : 0;
            $debug->unit_id = $debug->unit_id ? $debug->unit_id : 0;
            $debug->brand_id = $debug->brand_id ? $debug->brand_id : 0;

            DB::commit();

            return $this->respondSuccess($debug, "Berhasil mengupdate produk");
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Gagal mengupdate product. ".$e->getMessage());
        }
    }

    public function destroy($id)
    {
        if (!Auth::user()->can('product.delete')) {
            return $this->respondFailed("Akses ditolak", 403);
        }

        try {
            $business_id = Auth::user()->business_id;

            $can_be_deleted = true;
            $error_msg = '';

            //Check if any purchase or transfer exists
            $count = PurchaseLine::join(
                'transactions as T',
                'purchase_lines.transaction_id',
                '=',
                'T.id'
            )
                                ->whereIn('T.type', ['purchase'])
                                ->where('T.business_id', $business_id)
                                ->where('purchase_lines.product_id', $id)
                                ->count();
            if ($count > 0) {
                $can_be_deleted = false;
                $error_msg = "Pembelian sudah terdaftar";
            } else {
                //Check if any opening stock sold
                $count = PurchaseLine::join(
                    'transactions as T',
                    'purchase_lines.transaction_id',
                    '=',
                    'T.id'
                 )
                                ->where('T.type', 'opening_stock')
                                ->where('T.business_id', $business_id)
                                ->where('purchase_lines.product_id', $id)
                                ->where('purchase_lines.quantity_sold', '>', 0)
                                ->count();
                if ($count > 0) {
                    $can_be_deleted = false;
                    $error_msg = "Opening stok terjual habis";
                } else {
                    //Check if any stock is adjusted
                    $count = PurchaseLine::join(
                        'transactions as T',
                        'purchase_lines.transaction_id',
                        '=',
                        'T.id'
```

```
                    )
                                ->where('T.business_id', $business_id)
                                ->where('purchase_lines.product_id', $id)
                                ->where('purchase_lines.quantity_adjusted', '>', 0)
                                ->count();
                    if ($count > 0) {
                        $can_be_deleted = false;
                        $error_msg = "Stok diperbaharui";
                    }
                }
            }

            if ($can_be_deleted) {
                $product = Product::where('id', $id)
                            ->where('business_id', $business_id)
                            ->first();
                if (!empty($product)) {
                    DB::beginTransaction();
                    //Delete variation location details
                    VariationLocationDetails::where('product_id', $id)
                                            ->delete();
                    $product->delete();

                    DB::commit();
                }

                return $this->respondSuccess(null, "Produk berhasil dihapus");
            } else {
                return $this->respondFailed($error_msg);
            }
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed($e->getMessage());
        }
    }

    public function getSubCategories(Request $request)
    {
        if (!empty($request->input('cat_id'))) {
            $category_id = $request->input('cat_id');
            $business_id = $request->session()->get('user.business_id');
            $sub_categories = Category::where('business_id', $business_id)
                        ->where('parent_id', $category_id)
                        ->select(['name', 'id'])
                        ->get();
            $html = '<option value="">None</option>';
            if (!empty($sub_categories)) {
                foreach ($sub_categories as $sub_category) {
                    $html .= '<option value="' . $sub_category->id .'">' .$sub_category->name .
'</option>';
                }
            }
            echo $html;
            exit;
        }
    }

    public function getProductVariationFormPart(Request $request)
    {
        $business_id = Auth::user()->business_id;

        $variation_templates = VariationTemplate::where('business_id', $business_id)
            ->with(['values' => function($query) {
                $query->select(['id', 'name', 'variation_template_id']);
            }])
            ->select(['id', 'name'])
            ->get();

        return $this->respondArray($variation_templates);
    }

    public function getProductForTable(Request $request)
    {
        $business_id = Auth::user()->business_id;
        $data = $request->input();
        $location_id = $request->input('location_id', '');
        $check_qty = $request->input('check_qty', false);
        $price_group_id = $request->input('price_group', '');
```

```
        $products = Product::join('variations', 'products.id', '=', 'variations.product_id')
                ->active()
                ->whereNull('variations.deleted_at')
                ->where('variations.is_active', 1)
                ->leftjoin('units as U', 'products.unit_id', '=', 'U.id')
                ->leftjoin(
                    'variation_location_details AS VLD',
                    function ($join) use ($location_id) {
                        $join->on('variations.id', '=', 'VLD.variation_id');

                        //Include Location
                        if (!empty($location_id)) {
                            $join->where(function ($query) use ($location_id) {
                                $query->where('VLD.location_id', '=', $location_id);
                                //Check null to show products even if no quantity is available in
a location.
                                //TODO: Maybe add a settings to show product not available at a
location or not.
                                $query->orWhereNull('VLD.location_id');
                            });
                            ;
                        }
                    }
                );

            $not_for_selling = $request->input('not_for_selling', '');
            if ($not_for_selling) {
                $products->where('products.not_for_selling', $not_for_selling);
            }

            if (!empty($price_group_id)) {
                $products->leftjoin(
                    'variation_group_prices AS VGP',
                    function ($join) use ($price_group_id) {
                        $join->on('variations.id', '=', 'VGP.variation_id')
                            ->where('VGP.price_group_id', '=', $price_group_id);
                    }
                );
            }

            $products->where('products.business_id', $business_id)
                ->where('products.type', '!=', 'modifier')
                ->where('products.product_type', 'fisik');

            $product_types = $request->input('product_types', '');
            if ($product_types) {
                $products->whereIn('products.type', $product_types);
            }

            //Include check for quantity
            if ($check_qty) {
                $products->where('VLD.qty_available', '>', 0);
            }

            $products->select(
                'products.id as product_id',
                'products.name',
                'products.type',
                'products.enable_stock',
                'variations.id as variation_id',
                'variations.name as variation',
                DB::raw('IFNULL(VLD.qty_available, 0) as qty_available'),
                'variations.sell_price_inc_tax as selling_price',
                'variations.sub_sku',
                DB::raw('1 as qty'),
                'variations.default_purchase_price as purchase_price',
                'U.short_name as unit'
            );

            if (!empty($price_group_id)) {
                $products->addSelect('VGP.price_inc_tax as variation_group_price');
            }

            $type = $request->input('type');
            if (!empty($type)) {
                $products->where('products.type', $type);
            }

            $category_id = $request->input('category_id');
            if (!empty($category_id)) {
```

```
                $products->where('products.category_id', $category_id);
            }

            $sub_category_id = $request->input('sub_category_id');
            if (!empty($sub_category_id)) {
                $products->where('products.sub_category_id', $sub_category_id);
            }

            $brand_id = $request->input('brand_id');
            if (!empty($brand_id)) {
                $products->where('products.brand_id', $brand_id);
            }

            return $this->respondArray($products->get());
    }

    public function getVariationValueRow(Request $request)
    {
        $business_id = $request->session()->get('user.business_id');
        $business = Business::findorfail($business_id);
        $profit_percent = $business->default_profit_percent;

        $variation_index = $request->input('variation_row_index');
        $value_index = $request->input('value_index') + 1;

        $row_type = $request->input('row_type', 'add');

        return view('product.partials.variation_value_row')
                ->with(compact('profit_percent', 'variation_index', 'value_index', 'row_type'));
    }

    public function getProductVariationRow(Request $request)
    {
        $business_id = $request->session()->get('user.business_id');
        $business = Business::findorfail($business_id);
        $profit_percent = $business->default_profit_percent;

        $variation_templates = VariationTemplate::where('business_id', $business_id)
                                                ->pluck('name', 'id')->toArray();
        $variation_templates = [ "" => __('messages.please_select')] + $variation_templates;

        $row_index = $request->input('row_index', 0);
        $action = $request->input('action');

        return view('product.partials.product_variation_row')
                    ->with(compact('variation_templates', 'row_index', 'action',
'profit_percent'));
    }

    public function getVariationTemplate(Request $request)
    {
        $business_id = $request->session()->get('user.business_id');
        $business = Business::findorfail($business_id);
        $profit_percent = $business->default_profit_percent;

        $template = VariationTemplate::where('id', $request->input('template_id'))
                                                ->with(['values'])
                                                ->first();
        $row_index = $request->input('row_index');

        return view('product.partials.product_variation_template')
                    ->with(compact('template', 'row_index', 'profit_percent'));
    }

    public function getComboProductEntryRow(Request $request)
    {
        if (request()->ajax()) {
            $product_id = $request->input('product_id');
            $variation_id = $request->input('variation_id');
            $business_id = $request->session()->get('user.business_id');

            if (!empty($product_id)) {
                $product = Product::where('id', $product_id)
                        ->with(['unit'])
                        ->first();

                $query = Variation::where('product_id', $product_id)
                        ->with(['product_variation']);

                if ($variation_id !== '0') {
                    $query->where('id', $variation_id);
```

```
                }
                $variations =  $query->get();

                $sub_units = $this->productUtil->getSubUnits($business_id, $product['unit']->id);

                return view('product.partials.combo_product_entry_row')
                ->with(compact('product', 'variations', 'sub_units'));
            }
        }
    }

    public function getProducts()
    {
        if (request()->ajax()) {
            $term = request()->input('term', '');
            $location_id = request()->input('location_id', '');

            $check_qty = request()->input('check_qty', false);

            $price_group_id = request()->input('price_group', '');

            $business_id = request()->session()->get('user.business_id');

            $products = Product::join('variations', 'products.id', '=', 'variations.product_id')
                ->active()
                ->whereNull('variations.deleted_at')
                ->leftjoin('units as U', 'products.unit_id', '=', 'U.id')
                ->leftjoin(
                    'variation_location_details AS VLD',
                    function ($join) use ($location_id) {
                        $join->on('variations.id', '=', 'VLD.variation_id');

                        //Include Location
                        if (!empty($location_id)) {
                            $join->where(function ($query) use ($location_id) {
                                $query->where('VLD.location_id', '=', $location_id);
                                //Check null to show products even if no quantity is available in
a location.
                                //TODO: Maybe add a settings to show product not available at a
location or not.
                                $query->orWhereNull('VLD.location_id');
                            });
                            ;
                        }
                    }
                );
            if (request()->has('not_for_selling')) {
                $products->where('products.not_for_selling', request()->get('not_for_selling'));
            }
            if (!empty($price_group_id)) {
                $products->leftjoin(
                    'variation_group_prices AS VGP',
                    function ($join) use ($price_group_id) {
                        $join->on('variations.id', '=', 'VGP.variation_id')
                            ->where('VGP.price_group_id', '=', $price_group_id);
                    }
                );
            }
            $products->where('products.business_id', $business_id)
                ->where('products.type', '!=', 'modifier');
            if (request()->has('product_types')) {
                $products->whereIn('products.type', request()->get('product_types'));
            }
            //Include search
            if (!empty($term)) {
                $products->where(function ($query) use ($term) {
                    $query->where('products.name', 'like', '%' . $term .'%');
                    $query->orWhere('sku', 'like', '%' . $term .'%');
                    $query->orWhere('sub_sku', 'like', '%' . $term .'%');
                });
            }

            //Include check for quantity
            if ($check_qty) {
                $products->where('VLD.qty_available', '>', 0);
            }

            $products->select(
                'products.id as product_id',
                'products.name',
                'products.type',
```

```
                'products.enable_stock',
                'variations.id as variation_id',
                'variations.name as variation',
                'VLD.qty_available',
                'variations.sell_price_inc_tax as selling_price',
                'variations.sub_sku',
                'U.short_name as unit'
            );
            if (!empty($price_group_id)) {
                $products->addSelect('VGP.price_inc_tax as variation_group_price');
            }
            $result = $products->orderBy('VLD.qty_available', 'desc')
                        ->get();
            return json_encode($result);
        }
    }

    public function getProductsWithoutVariations()
    {
        if (request()->ajax()) {
            $term = request()->input('term', '');

            $business_id = request()->session()->get('user.business_id');

            $products = Product::join('variations', 'products.id', '=', 'variations.product_id')
                ->where('products.business_id', $business_id)
                ->where('products.type', '!=', 'modifier');

            //Include search
            if (!empty($term)) {
                $products->where(function ($query) use ($term) {
                    $query->where('products.name', 'like', '%' . $term .'%');
                    $query->orWhere('sku', 'like', '%' . $term .'%');
                    $query->orWhere('sub_sku', 'like', '%' . $term .'%');
                });
            }

            $products = $products->groupBy('products.id')
                ->select(
                    'products.id as product_id',
                    'products.name',
                    'products.type',
                    'products.enable_stock',
                    'products.sku'
                )
                    ->orderBy('products.name')
                    ->get();
            return json_encode($products);
        }
    }

    public function checkProductSku($sku, $product_id = '')
    {
        $business_id = Auth::user()->business_id;

        //check in products table
        $query = Product::where('business_id', $business_id)
                        ->where('sku', $sku);

        if (!empty($product_id)) {
            $query->where('id', '!=', $product_id);
        }

        $count = $query->count();

        //check in variation table if $count = 0
        if ($count == 0) {
            $count = Variation::where('sub_sku', $sku)
                            ->join('products', 'variations.product_id', '=', 'products.id')
                            ->where('product_id', '!=', $product_id)
                            ->where('business_id', $business_id)
                            ->count();
        }

        if ($count == 0) {
            return true;
        } else {
            return false;
        }
    }
```

```
    public function quickAdd()
    {
        if (!auth()->user()->can('product.create')) {
            abort(403, 'Unauthorized action.');
        }

        $product_name = !empty(request()->input('product_name'))? request()-
>input('product_name') : '';

        $product_for = !empty(request()->input('product_for'))? request()->input('product_for') :
null;

        $business_id = request()->session()->get('user.business_id');
        $categories = Category::forDropdown($business_id);
        $brands = Brands::where('business_id', $business_id)
                        ->pluck('name', 'id');
        $units = Unit::forDropdown($business_id, true);

        $tax_dropdown = TaxRate::forBusinessDropdown($business_id, true, true);
        $taxes = $tax_dropdown['tax_rates'];
        $tax_attributes = $tax_dropdown['attributes'];

        $barcode_types = $this->barcode_types;

        $default_profit_percent = Business::where('id', $business_id)-
>value('default_profit_percent');

        $locations = BusinessLocation::forDropdown($business_id);

        $enable_expiry = request()->session()->get('business.enable_product_expiry');
        $enable_lot = request()->session()->get('business.enable_lot_number');

        $module_form_parts = $this->moduleUtil->getModuleData('product_form_part');

        return view('product.partials.quick_add_product')
                ->with(compact('categories', 'brands', 'units', 'taxes', 'barcode_types',
'default_profit_percent', 'tax_attributes', 'product_name', 'locations', 'product_for',
'enable_expiry', 'enable_lot', 'module_form_parts'));
    }

    public function saveQuickProduct(Request $request)
    {
        if (!auth()->user()->can('product.create')) {
            abort(403, 'Unauthorized action.');
        }

        try {
            $business_id = $request->session()->get('user.business_id');
            $form_fields = ['name', 'brand_id', 'unit_id', 'category_id', 'tax',
'barcode_type','tax_type', 'sku',
                'alert_quantity', 'type', 'sub_unit_ids'];

            $module_form_fields = $this->moduleUtil->getModuleData('product_form_fields');
            if (!empty($module_form_fields)) {
                foreach ($module_form_fields as $key => $value) {
                    if (!empty($value) && is_array($value)) {
                        $form_fields = array_merge($form_fields, $value);
                    }
                }
            }
            $product_details = $request->only($form_fields);

            $product_details['type'] = empty($product_details['type']) ? 'single' :
$product_details['type'];
            $product_details['product_description'] = $request->input('product_description');
            $product_details['business_id'] = $business_id;
            $product_details['created_by'] = $request->session()->get('user.id');
            if (!empty($request->input('enable_stock')) &&  $request->input('enable_stock') == 1)
{
                $product_details['enable_stock'] = 1 ;
                //TODO: Save total qty
                //$product_details['total_qty_available'] = 0;
            }
            if (!empty($request->input('not_for_selling')) &&  $request->input('not_for_selling')
== 1) {
                $product_details['not_for_selling'] = 1 ;
            }
            if (empty($product_details['sku'])) {
                $product_details['sku'] = ' ';
            }
```

```
            $expiry_enabled = $request->session()->get('business.enable_product_expiry');
            if (!empty($request->input('expiry_period_type')) && !empty($request-
>input('expiry_period')) && !empty($expiry_enabled)) {
                $product_details['expiry_period_type'] = $request->input('expiry_period_type');
                $product_details['expiry_period'] = $this->productUtil->num_uf($request-
>input('expiry_period'));
            }

            if (!empty($request->input('enable_sr_no')) &&  $request->input('enable_sr_no') == 1)
{
                $product_details['enable_sr_no'] = 1 ;
            }

            DB::beginTransaction();

            $product = Product::create($product_details);

            if (empty(trim($request->input('sku')))) {
                $sku = $this->productUtil->generateProductSku($product->id);
                $product->sku = $sku;
                $product->save();
            }

            $this->productUtil->createSingleProductVariation(
                $product->id,
                $product->sku,
                $request->input('single_dpp'),
                $request->input('single_dpp_inc_tax'),
                $request->input('profit_percent'),
                $request->input('single_dsp'),
                $request->input('single_dsp_inc_tax')
            );

            if ($product->enable_stock == 1 && !empty($request->input('opening_stock'))) {
                $user_id = $request->session()->get('user.id');

                $transaction_date = $request->session()->get("financial_year.start");
                $transaction_date = \Carbon::createFromFormat('Y-m-d', $transaction_date)-
>toDateTimeString();

                $this->productUtil->addSingleProductOpeningStock($business_id, $product,
$request->input('opening_stock'), $transaction_date, $user_id);
            }

            DB::commit();

            $output = ['success' => 1,
                            'msg' => __('product.product_added_success'),
                            'product' => $product,
                            'variation' => $product->variations->first()
                        ];
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            $output = ['success' => 0,
                            'msg' => __("messages.something_went_wrong")
                        ];
        }

        return $output;
    }

    public function view($id)
    {
        if (!auth()->user()->can('product.view')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = request()->session()->get('user.business_id');

        $product = Product::where('business_id', $business_id)
                    ->where('id', $id)
                    ->with(['brand', 'unit', 'category', 'sub_category', 'product_tax',
'variations', 'variations.product_variation', 'variations.group_prices', 'variations.media'])
                    ->first();

        $price_groups = SellingPriceGroup::where('business_id', $business_id)->pluck('name',
'id');
```

```
        $allowed_group_prices = [];
        foreach ($price_groups as $key => $value) {
            if (auth()->user()->can('selling_price_group.' . $key)) {
                $allowed_group_prices[$key] = $value;
            }
        }

        $group_price_details = [];

        foreach ($product->variations as $variation) {
            foreach ($variation->group_prices as $group_price) {
                $group_price_details[$variation->id][$group_price->price_group_id] =
$group_price->price_inc_tax;
            }
        }

        $rack_details = $this->productUtil->getRackDetails($business_id, $id, true);

        $combo_variations = [];
        if ($product->type == 'combo') {
            $combo_variations = $this->__getComboProductDetails($product['variations'][0]-
>combo_variations, $business_id);
        }

        return view('product.view-modal')->with(compact(
            'product',
            'rack_details',
            'allowed_group_prices',
            'group_price_details',
            'combo_variations'
        ));
    }

    private function __getComboProductDetails($combo_variations, $business_id)
    {
        foreach ($combo_variations as $key => $value) {
            $combo_variations[$key]['variation'] =
                Variation::with(['product'])
                    ->find($value['variation_id']);

            $combo_variations[$key]['sub_units'] = $this->productUtil->getSubUnits($business_id,
$combo_variations[$key]['variation']['product']->unit_id, true);

            $combo_variations[$key]['multiplier'] = 1;

            if (!empty($combo_variations[$key]['sub_units'])) {
                if
(isset($combo_variations[$key]['sub_units'][$combo_variations[$key]['unit_id']])) {
                    $combo_variations[$key]['multiplier'] =
$combo_variations[$key]['sub_units'][$combo_variations[$key]['unit_id']]['multiplier'];
                    $combo_variations[$key]['unit_name'] =
$combo_variations[$key]['sub_units'][$combo_variations[$key]['unit_id']]['name'];
                }
            }
        }

        return $combo_variations;
    }

    public function massDestroy(Request $request)
    {
        if (!auth()->user()->can('product.delete')) {
            abort(403, 'Unauthorized action.');
        }
        try {
            $purchase_exist = false;

            if (!empty($request->input('selected_rows'))) {
                $business_id = $request->session()->get('user.business_id');

                $selected_rows = explode(',', $request->input('selected_rows'));

                $products = Product::where('business_id', $business_id)
                                    ->whereIn('id', $selected_rows)
                                    ->with('purchase_lines')
                                    ->get();
                $deletable_products = [];

                DB::beginTransaction();

                foreach ($products as $product) {
```

```
                    //Delete if no purchase found
                    if (empty($product->purchase_lines->toArray())) {
                        //Delete variation location details
                        VariationLocationDetails::where('product_id', $product->id)
                                                ->delete();
                        $product->delete();
                    } else {
                        $purchase_exist = true;
                    }
                }

                DB::commit();
            }

            if (!$purchase_exist) {
                $output = ['success' => 1,
                        'msg' => __('lang_v1.deleted_success')
                    ];
            } else {
                $output = ['success' => 0,
                        'msg' => __('lang_v1.products_could_not_be_deleted')
                    ];
            }
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            $output = ['success' => 0,
                        'msg' => __("messages.something_went_wrong")
                    ];
        }

        return redirect()->back()->with(['status' => $output]);
    }

    public function addSellingPrices($id)
    {
        if (!Auth::user()->can('product.create')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = Auth::user()->business_id;
        $product = Product::where('business_id', $business_id)
                    ->with(['variations', 'variations.group_prices',
'variations.product_variation'])
                    ->findOrFail($id);

        $price_groups = SellingPriceGroup::where('business_id', $business_id)
                                        ->get();

        $variation = $product->variations[0];

        $variation_prices = [];
        foreach ($variation->group_prices as $group_price) {
            $variation_prices[$variation->id][$group_price->price_group_id] = $group_price-
>price_inc_tax;
        }

        // dd($price_groups);
        foreach($price_groups as $pg){
            if(count($variation_prices) > 0 && array_key_exists($pg->id,
$variation_prices[$variation->id]))
                $pg->price = $variation_prices[$variation->id][$pg->id];
            else
                $pg->price = "0";
        }

        // dd($product,$price_groups, $variation_prices);
        return $this->respondArray($price_groups);
    }

    public function saveSellingPrices(Request $request)
    {
        if (!Auth::user()->can('product.create')) {
            abort(403, 'Unauthorized action.');
        }

        try {
            $business_id = Auth::user()->business_id;
            $product = Product::where('business_id', $business_id)
```

```
                            ->with(['variations'])
                            ->findOrFail($request->input('product_id'));
            DB::beginTransaction();
            $variationId = $request->input('variation_id');
            $priceGroups = $request->input('price_groups');
            $variation = Variation::findOrFail($variationId);
            // foreach ($product->variations as $variation) {
                $variation_group_prices = [];
                foreach ($priceGroups as $value) {
                    if ($value['price'] > 0) {
                        $variation_group_price =
                        VariationGroupPrice::where('variation_id', $variation->id)
                                            ->where('price_group_id', $value['id'])
                                            ->first();
                        if (empty($variation_group_price)) {
                            $variation_group_price = new VariationGroupPrice([
                                    'variation_id' => $variation->id,
                                    'price_group_id' => $value['id']
                                ]);
                        }

                        $variation_group_price->price_inc_tax = $this->productUtil-
>num_uf($value['price']);
                        $variation_group_prices[] = $variation_group_price;
                    }
                }

                // dd($variation_group_prices);

                if (!empty($variation_group_prices)) {
                    $variation->group_prices()->saveMany($variation_group_prices);
                }
            // }
            DB::commit();

            return $this->respondSuccess(null, "Grup harga produk berhasil disimpan");
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Terjadi kesalahan saat menyimpan grup harga produk");
        }
    }

    public function viewGroupPrice($id)
    {
        if (!auth()->user()->can('product.view')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = request()->session()->get('user.business_id');

        $product = Product::where('business_id', $business_id)
                            ->where('id', $id)
                            ->with(['variations', 'variations.product_variation',
'variations.group_prices'])
                            ->first();

        $price_groups = SellingPriceGroup::where('business_id', $business_id)->pluck('name',
'id');

        $allowed_group_prices = [];
        foreach ($price_groups as $key => $value) {
            if (auth()->user()->can('selling_price_group.' . $key)) {
                $allowed_group_prices[$key] = $value;
            }
        }

        $group_price_details = [];

        foreach ($product->variations as $variation) {
            foreach ($variation->group_prices as $group_price) {
                $group_price_details[$variation->id][$group_price->price_group_id] =
$group_price->price_inc_tax;
            }
        }

        return view('product.view-product-group-prices')->with(compact('product',
'allowed_group_prices', 'group_price_details'));
    }
```

```
    public function massDeactivate(Request $request)
    {
        if (!Auth::user()->can('product.update')) {
            abort(403, 'Unauthorized action.');
        }
        try {
            if (!empty($request->input('selected_products'))) {
                $business_id = Auth::user()->business_id;

                $selected_products = explode(',', $request->input('selected_products'));

                DB::beginTransaction();

                $products = Product::where('business_id', $business_id)
                                    ->whereIn('id', $selected_products)
                                    ->update(['is_inactive' => 1]);

                DB::commit();
            }

            return $this->respondSuccess(null, "Produk berhasil dinonaktifkan");
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Produk gagal dinonaktifkan");
        }
    }

    public function activate($id)
    {
        if (!Auth::user()->can('product.update')) {
            abort(403, 'Unauthorized action.');
        }

        try {
            $business_id = Auth::user()->business_id;
            $product = Product::where('id', $id)
                            ->where('business_id', $business_id)
                            ->update(['is_inactive' => 0]);

            return $this->respondSuccess(null, "Produk berhasil diaktifkan");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed("Produk gagal diaktifkan");
        }
    }

    public function deleteMedia($media_id)
    {
        if (!auth()->user()->can('product.update')) {
            abort(403, 'Unauthorized action.');
        }

        if (request()->ajax()) {
            try {
                $business_id = request()->session()->get('user.business_id');

                Media::deleteMedia($business_id, $media_id);

                $output = ['success' => true,
                            'msg' => __("lang_v1.file_deleted_successfully")
                            ];
            } catch (\Exception $e) {
                \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" .
$e->getMessage());

                $output = ['success' => false,
                            'msg' => __("messages.something_went_wrong")
                            ];
            }

            return $output;
        }
    }

    public function getProductsApi($id = null)
```

```
    {
        try {
            $api_token = request()->header('API-TOKEN');
            $filter_string = request()->header('FILTERS');
            $order_by = request()->header('ORDER-BY');

            parse_str($filter_string, $filters);

            $api_settings = $this->moduleUtil->getApiSettings($api_token);

            $limit = !empty(request()->input('limit')) ? request()->input('limit') : 10;

            $location_id = $api_settings->location_id;

            $query = Product::where('business_id', $api_settings->business_id)
                            ->active()
                            ->with(['brand', 'unit', 'category', 'sub_category',
                                'product_variations', 'product_variations.variations',
'product_variations.variations.media',
                                'product_variations.variations.variation_location_details' =>
function ($q) use ($location_id) {
                                    $q->where('location_id', $location_id);
                                }]);

            if (!empty($filters['categories'])) {
                $query->whereIn('category_id', $filters['categories']);
            }

            if (!empty($filters['brands'])) {
                $query->whereIn('brand_id', $filters['brands']);
            }

            if (!empty($filters['category'])) {
                $query->where('category_id', $filters['category']);
            }

            if (!empty($filters['sub_category'])) {
                $query->where('sub_category_id', $filters['sub_category']);
            }

            if ($order_by == 'name') {
                $query->orderBy('name', 'asc');
            } elseif ($order_by == 'date') {
                $query->orderBy('created_at', 'desc');
            }

            if (empty($id)) {
                $products = $query->paginate($limit);
            } else {
                $products = $query->find($id);
            }
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondWentWrong($e);
        }

        return $this->respond($products);
    }

    public function getVariationsApi()
    {
        try {
            $api_token = request()->header('API-TOKEN');
            $variations_string = request()->header('VARIATIONS');

            if (is_numeric($variations_string)) {
                $variation_ids = intval($variations_string);
            } else {
                parse_str($variations_string, $variation_ids);
            }

            $api_settings = $this->moduleUtil->getApiSettings($api_token);
            $location_id = $api_settings->location_id;
            $business_id = $api_settings->business_id;

            $query = Variation::with([
                                'product_variation',
                                'product' => function ($q) use ($business_id) {
                                    $q->where('business_id', $business_id);
```

```
                                },
                                'product.unit',
                                'variation_location_details' => function ($q) use ($location_id)
{
                                    $q->where('location_id', $location_id);
                                }
                            ]);

            $variations = is_array($variation_ids) ? $query->whereIn('id', $variation_ids)->get()
: $query->where('id', $variation_ids)->first();
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondWentWrong($e);
        }

        return $this->respond($variations);
    }

    public function bulkEdit(Request $request)
    {
        if (!auth()->user()->can('product.update')) {
            abort(403, 'Unauthorized action.');
        }

        $selected_products_string = $request->input('selected_products');
        if (!empty($selected_products_string)) {
            $selected_products = explode(',', $selected_products_string);
            $business_id = $request->session()->get('user.business_id');

            $products = Product::where('business_id', $business_id)
                                ->whereIn('id', $selected_products)
                                ->with(['variations', 'variations.product_variation',
'variations.group_prices'])
                                ->get();

            $all_categories = Category::catAndSubCategories($business_id);

            $categories = [];
            $sub_categories = [];
            foreach ($all_categories as $category) {
                $categories[$category['id']] = $category['name'];

                if (!empty($category['sub_categories'])) {
                    foreach ($category['sub_categories'] as $sub_category) {
                        $sub_categories[$category['id']][$sub_category['id']] =
$sub_category['name'];
                    }
                }
            }

            $brands = Brands::where('business_id', $business_id)
                            ->pluck('name', 'id');

            $tax_dropdown = TaxRate::forBusinessDropdown($business_id, true, true);
            $taxes = $tax_dropdown['tax_rates'];
            $tax_attributes = $tax_dropdown['attributes'];

            $price_groups = SellingPriceGroup::where('business_id', $business_id)->pluck('name',
'id');

            return view('product.bulk-edit')->with(compact(
                'products',
                'categories',
                'brands',
                'taxes',
                'tax_attributes',
                'sub_categories',
                'price_groups'
            ));
        }
    }

    public function bulkUpdate(Request $request)
    {
        if (!auth()->user()->can('product.update')) {
            abort(403, 'Unauthorized action.');
        }

        try {
```

```
            $products = $request->input('products');
            $business_id = $request->session()->get('user.business_id');

            DB::beginTransaction();
            foreach ($products as $id => $product_data) {
                $update_data = [
                    'category_id' => $product_data['category_id'],
                    'sub_category_id' => $product_data['sub_category_id'],
                    'brand_id' => $product_data['brand_id'],
                    'tax' => $product_data['tax'],
                ];

                //Update product
                $product = Product::where('business_id', $business_id)
                                ->findOrFail($id);

                $product->update($update_data);

                $variations_data = [];

                //Format variations data
                foreach ($product_data['variations'] as $key => $value) {
                    $variation = Variation::where('product_id', $product->id)->findOrFail($key);
                    $variation->default_purchase_price = $this->productUtil-
>num_uf($value['default_purchase_price']);
                    $variation->dpp_inc_tax = $this->productUtil->num_uf($value['dpp_inc_tax']);
                    $variation->profit_percent = $this->productUtil-
>num_uf($value['profit_percent']);
                    $variation->default_sell_price = $this->productUtil-
>num_uf($value['default_sell_price']);
                    $variation->sell_price_inc_tax = $this->productUtil-
>num_uf($value['sell_price_inc_tax']);
                    $variations_data[] = $variation;

                    //Update price groups
                    if (!empty($value['group_prices'])) {
                        foreach ($value['group_prices'] as $k => $v) {
                            VariationGroupPrice::updateOrCreate(
                                ['price_group_id' => $k, 'variation_id' => $variation->id],
                                ['price_inc_tax' => $this->productUtil->num_uf($v)]
                            );
                        }
                    }
                }
                $product->variations()->saveMany($variations_data);
            }
            DB::commit();

            $output = ['success' => 1,
                            'msg' => __("lang_v1.updated_success")
                        ];
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            $output = ['success' => 0,
                            'msg' => __("messages.something_went_wrong")
                        ];
        }

        return redirect('products')->with('status', $output);
    }

    public function getProductToEdit($product_id)
    {
        if (!auth()->user()->can('product.update')) {
            abort(403, 'Unauthorized action.');
        }
        $business_id = request()->session()->get('user.business_id');

        $product = Product::where('business_id', $business_id)
                        ->with(['variations', 'variations.product_variation',
'variations.group_prices'])
                        ->findOrFail($product_id);
        $all_categories = Category::catAndSubCategories($business_id);

        $categories = [];
        $sub_categories = [];
        foreach ($all_categories as $category) {
            $categories[$category['id']] = $category['name'];
```

```
            if (!empty($category['sub_categories'])) {
                foreach ($category['sub_categories'] as $sub_category) {
                    $sub_categories[$category['id']][$sub_category['id']] =
$sub_category['name'];
                }
            }
        }

        $brands = Brands::where('business_id', $business_id)
                        ->pluck('name', 'id');

        $tax_dropdown = TaxRate::forBusinessDropdown($business_id, true, true);
        $taxes = $tax_dropdown['tax_rates'];
        $tax_attributes = $tax_dropdown['attributes'];

        $price_groups = SellingPriceGroup::where('business_id', $business_id)->pluck('name',
'id');

        return view('product.partials.bulk_edit_product_row')->with(compact(
            'product',
            'categories',
            'brands',
            'taxes',
            'tax_attributes',
            'sub_categories',
            'price_groups'
        ));
    }

    public function getSubUnits(Request $request)
    {
        if (!empty($request->input('unit_id'))) {
            $unit_id = $request->input('unit_id');
            $business_id = $request->session()->get('user.business_id');
            $sub_units = $this->productUtil->getSubUnits($business_id, $unit_id, true);

            //$html = '<option value="">' . __('lang_v1.all') . '</option>';
            $html = '';
            if (!empty($sub_units)) {
                foreach ($sub_units as $id => $sub_unit) {
                    $html .= '<option value="' . $id .'">' .$sub_unit['name'] . '</option>';
                }
            }

            return $html;
        }
    }
}
```

q. *OpeningStock Controller*

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use Auth;

use App\Product;
use App\Transaction;
use App\BusinessLocation;
use App\PurchaseLine;
use App\Variation;

use App\Utils\ProductUtil;
use App\Utils\TransactionUtil;
use App\Utils\BusinessUtil;

use Illuminate\Support\Facades\DB;

class OpeningStockController extends Controller
{

    protected $productUtil;
    protected $transactionUtil;
    protected $business_util;

    public function __construct(ProductUtil $productUtil, TransactionUtil $transactionUtil,
BusinessUtil $business_util)
```

```
    {
        $this->productUtil = $productUtil;
        $this->transactionUtil = $transactionUtil;
        $this->business_util = $business_util;
    }

    public function add($product_id)
    {
        if (!Auth::user()->can('product.update')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = Auth::user()->business_id;

        //Get the product
        $product = Product::where('business_id', $business_id)
                          ->where('id', $product_id)
                          ->with(['variations', 'variations.product_variation', 'unit'])
                          ->first();
        if (!empty($product) && $product->enable_stock == 1) {
            //Get Opening Stock Transactions for the product if exists
            $transactions = Transaction::where('business_id', $business_id)
                                ->where('opening_stock_product_id', $product_id)
                                ->where('type', 'opening_stock')
                                ->with(['purchase_lines'])
                                ->get();

            $purchases = [];
            foreach ($transactions as $transaction) {
                $purchase_lines = [];

                foreach ($transaction->purchase_lines as $purchase_line) {
                    if (!empty($purchase_lines[$purchase_line->variation_id])) {
                        $k = count($purchase_lines[$purchase_line->variation_id]);
                    } else {
                        $k = 0;
                        $purchase_lines[$purchase_line->variation_id] = [];
                    }

                    //Show only remaining quantity for editing opening stock.
                    $purchase_lines[$purchase_line->variation_id][$k]['quantity'] =
$purchase_line->quantity_remaining;
                    $purchase_lines[$purchase_line->variation_id][$k]['purchase_price'] =
$purchase_line->purchase_price;
                    $purchase_lines[$purchase_line->variation_id][$k]['purchase_line_id'] =
$purchase_line->id;
                    $purchase_lines[$purchase_line->variation_id][$k]['exp_date'] =
$purchase_line->exp_date;
                    $purchase_lines[$purchase_line->variation_id][$k]['lot_number'] =
$purchase_line->lot_number;
                }
                $purchases[$transaction->location_id] = $purchase_lines;
            }

            $locations = BusinessLocation::forDropdown($business_id);

            $enable_expiry = Auth::user()->business->enable_product_expiry;
            $enable_lot = Auth::user()->business->enable_lot_number;

            foreach($locations as $location) {
                $location->purchase_price = $product->variations[0]->default_purchase_price;
                $location->quantity = 0;
            }

            return $this->respondArray($locations);
        }
    }

    public function save(Request $request)
    {
        if (!Auth::user()->can('product.update')) {
            return $this->respondFailed("Tidak diizinkan");
        }

        try {
            $opening_stocks = $request->input('stocks');
            $product_id = $request->input('product_id');
            $variation = Variation::findOrFail($request->input('variation_id'));

            $business_id = Auth::user()->business_id;
            $user_id = Auth::user()->id;
```

```
            $product = Product::where('business_id', $business_id)
                                ->where('id', $product_id)
                                ->with(['variations', 'product_tax'])
                                ->first();

            $locations = BusinessLocation::forDropdown2($business_id)->toArray();

            if (!empty($product) && $product->enable_stock == 1) {
                //Get product tax
                $tax_percent = !empty($product->product_tax->amount) ? $product->product_tax-
>amount : 0;
                $tax_id = !empty($product->product_tax->id) ? $product->product_tax->id : null;

                //Get start date for financial year.
                $transaction_date = $this->business_util->getCurrentFinancialYear($business_id);
                $transaction_date = \Carbon\Carbon::createFromFormat('Y-m-d',
$transaction_date['start'])->toDateTimeString();

                DB::beginTransaction();
                //$key is the location_id
                foreach ($opening_stocks as $key_os => $value) {
                    $location_id = $value['id'];
                    $purchase_total = 0;
                    //Check if valid location
                    $purchase_lines = [];
                    $updated_purchase_line_ids = [];

                    $purchase_price = $this->productUtil->num_uf(trim($value['purchase_price']));
                    $item_tax = 0;
                    $purchase_price_inc_tax = $purchase_price + $item_tax;
                    $qty_remaining = $this->productUtil->num_uf(trim($value['quantity']));

                    $exp_date = null;
                    if (!empty($value['exp_date'])) {
                        $exp_date = $this->productUtil->uf_date($v['exp_date']);
                    }

                    $lot_number = null;
                    if (!empty($value['lot_number'])) {
                        $lot_number = $v['lot_number'];
                    }

                    $purchase_line = null;

                    if (isset($value['purchase_line_id'])) {
                        $purchase_line = PurchaseLine::findOrFail($v['purchase_line_id']);
                        //Quantity = remaining + used
                        $qty_remaining = $qty_remaining + $purchase_line->quantity_used;

                        if ($qty_remaining != 0) {
                            //Calculate transaction total
                            $purchase_total += ($purchase_price_inc_tax * $qty_remaining);

                            $updated_purchase_line_ids[] = $purchase_line->id;

                            $old_qty = $purchase_line->quantity;

                            $this->productUtil->updateProductQuantity($location_id, $product->id,
$k, $qty_remaining, $old_qty, null, false);
                        }
                    } else {
                        // if ($qty_remaining != 0) {

                            //create newly added purchase lines
                            $purchase_line = new PurchaseLine();
                            $purchase_line->product_id = $product->id;
                            $purchase_line->variation_id = $variation->id;

                            $this->productUtil->updateProductQuantity($location_id, $product->id,
$variation->id, $qty_remaining, 0, null, false);

                            //Calculate transaction total
                            $purchase_total += ($purchase_price_inc_tax * $qty_remaining);
                        // }
                    }

                    if (!is_null($purchase_line)) {
                        $purchase_line->item_tax = $item_tax;
                        $purchase_line->tax_id = $tax_id;
                        $purchase_line->quantity = $qty_remaining;
```

```
                    $purchase_line->pp_without_discount = $purchase_price;
                    $purchase_line->purchase_price = $purchase_price;
                    $purchase_line->purchase_price_inc_tax = $purchase_price_inc_tax;
                    $purchase_line->exp_date = $exp_date;
                    $purchase_line->lot_number = $lot_number;

                    $purchase_lines[] = $purchase_line;
                }


                //create transaction & purchase lines
                if (!empty($purchase_lines)) {
                    $is_new_transaction = false;

                    $transaction = Transaction::where('type', 'opening_stock')
                            ->where('business_id', $business_id)
                            ->where('opening_stock_product_id', $product->id)
                            ->where('location_id', $location_id)
                            ->first();
                    if (!empty($transaction)) {
                        $transaction->total_before_tax = $purchase_total;
                        $transaction->final_total = $purchase_total;
                        $transaction->update();
                    } else {
                        $is_new_transaction = true;

                        $transaction = Transaction::create(
                            [
                                'type' => 'opening_stock',
                                'opening_stock_product_id' => $product->id,
                                'status' => 'received',
                                'business_id' => $business_id,
                                'transaction_date' => $transaction_date,
                                'total_before_tax' => $purchase_total,
                                'location_id' => $location_id,
                                'final_total' => $purchase_total,
                                'payment_status' => 'paid',
                                'created_by' => $user_id
                            ]
                        );
                    }

                    //unset deleted purchase lines
                    $delete_purchase_line_ids = [];
                    $delete_purchase_lines = null;
                    $delete_purchase_lines = PurchaseLine::where('transaction_id',
$transaction->id)
                                ->whereNotIn('id', $updated_purchase_line_ids)
                                ->get();

                    if ($delete_purchase_lines->count()) {
                        foreach ($delete_purchase_lines as $delete_purchase_line) {
                            $delete_purchase_line_ids[] = $delete_purchase_line->id;

                            //decrease deleted only if previous status was received
                            $this->productUtil->decreaseProductQuantity(
                                $delete_purchase_line->product_id,
                                $delete_purchase_line->variation_id,
                                $transaction->location_id,
                                $delete_purchase_line->quantity
                            );
                        }
                        //Delete deleted purchase lines
                        PurchaseLine::where('transaction_id', $transaction->id)
                                    ->whereIn('id', $delete_purchase_line_ids)
                                    ->delete();
                    }
                    $transaction->purchase_lines()->saveMany($purchase_lines);

                    //Update mapping of purchase & Sell.
                    if (!$is_new_transaction) {
                        $this->transactionUtil-
>adjustMappingPurchaseSellAfterEditingPurchase('received', $transaction, $delete_purchase_lines);
                    }

                    //Adjust stock over selling if found
                    $this->productUtil->adjustStockOverSelling($transaction);
                } else {
                    //Delete transaction if all purchase line quantity is 0 (Only if
transaction exists)
                    $delete_transaction = Transaction::where('type', 'opening_stock')
```

```
                                ->where('business_id', $business_id)
                                ->where('opening_stock_product_id', $product->id)
                                ->where('location_id', $location_id)
                                ->with(['purchase_lines'])
                                ->first();

                            if (!empty($delete_transaction)) {
                                $delete_purchase_lines = $delete_transaction->purchase_lines;

                                foreach ($delete_purchase_lines as $delete_purchase_line) {
                                    $this->productUtil->decreaseProductQuantity($product->id,
$delete_purchase_line->variation_id, $location_id, $delete_purchase_line->quantity);
                                    $delete_purchase_line->delete();
                                }

                                //Update mapping of purchase & Sell.
                                $this->transactionUtil-
>adjustMappingPurchaseSellAfterEditingPurchase('received', $delete_transaction,
$delete_purchase_lines);

                                $delete_transaction->delete();
                            }
                        }
                }

                DB::commit();
            }

            return $this->respondSuccess(null, "Berhasil mengatur stok awal");
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());
            dd($e);
            return $this->respondFailed("Gagal mengatur stok awal");
        }
    }
}
```

r. *CashRegister Controller*

```
<?php

namespace App\Http\Controllers;

use Auth;
use App\CashRegister;
use App\Account;
use DB;
use App\Utils\CashRegisterUtil;

use Illuminate\Http\Request;

class CashRegisterController extends Controller
{

    protected $cashRegisterUtil;

    public function __construct(CashRegisterUtil $cashRegisterUtil)
    {
        $this->cashRegisterUtil = $cashRegisterUtil;
    }

    public function index()
    {
        return view('cash_register.index');
    }

    public function create()
    {
        //Check if there is a open register, if yes then redirect to POS screen.
        if ($this->cashRegisterUtil->countOpenedRegister() != 0) {
            return redirect()->action('SellPosController@create');
        }

        return view('cash_register.create');
    }

    public function store(Request $request)
    {
        try {
```

```
            $initial_amount = 0;
            if (!empty($request->input('amount'))) {
                $initial_amount = $this->cashRegisterUtil->num_uf($request->input('amount'));
            }

            $account = Account::where('outlet_id', $request->input('location_id'))->first();
            $getBalance = DB::select(
                                                        'SELECT SUM(
                                                                IF(AT.type="credit",
AT.amount, -1 * AT.amount)
                                                        ) as balance from
account_transactions as AT
                                                        WHERE AT.deleted_at IS NULL
                                                        AND AT.account_id = '.$account-
>id.';'
                                            )[0];

                                            //if account balance less then amountToPay
                                            if ($getBalance->balance < $initial_amount) {
                                                        return $this->respondFailed("Tidak
boleh melebihi jumlah kas kasir.");
                                            }

            $user_id = Auth::id();
            $business_id = Auth::user()->business_id;

            $register = CashRegister::create([
                        'business_id' => $business_id,
                        'user_id' => $user_id,
                        'status' => 'open'
                    ]);
            $register->cash_register_transactions()->create([
                            'amount' => $initial_amount,
                            'pay_method' => 'cash',
                            'type' => 'credit',
                            'transaction_type' => 'initial'
                        ]);

            return $this->respondSuccess($register, "Kasir berhasil dibuka");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());
            return $this->respondFailed($e->getMessage());
        }
    }

    public function show($id)
    {
        $register_details =  $this->cashRegisterUtil->getRegisterDetails($id);
        $user_id = $register_details->user_id;
        $open_time = $register_details['open_time'];
        $close_time = \Carbon\Carbon::now()->toDateTimeString();
        $details = $this->cashRegisterUtil->getRegisterTransactionDetails($user_id, $open_time,
$close_time);

        $payment_types = $this->cashRegisterUtil->payment_types();

        $data = [
            'register_details' => $register_details,
            'details' => $details,
            'payment_types' => $payment_types
        ];

        return $this->respondSuccess($data);
    }

    public function getRegisterDetails(Request $request)
    {
        $register_details =  $this->cashRegisterUtil->getRegisterDetails();
        $register_details->total = $register_details->cash_in_hand + $register_details-
>total_cash - $register_details->total_cash_refund;

        $user_id = Auth::id();
        $open_time = $register_details['open_time'];

        if(!$open_time)
            return $this->respondSuccess2(null, 'There is no open cash register', false);

        $close_time = \Carbon\Carbon::now()->toDateTimeString();
        $details = $this->cashRegisterUtil->getRegisterTransactionDetails($user_id, $open_time,
$close_time);
```

```
        $payment_types = $this->cashRegisterUtil->payment_types();

        $accounts = Account::leftjoin('account_transactions as AT', function ($join) {
                                $join->on('AT.account_id', '=', 'accounts.id');
                                $join->whereNull('AT.deleted_at');
                            })
                            ->leftJoin('account_categories as AC', function ($join) {
                                $join->on('AC.id', '=', 'accounts.account_categories_id');
                            })
                            ->where('business_id', Auth::user()->business_id)
                            ->where('accounts.outlet_id', $request->input('location_id'))
                            ->select(['accounts.name', 'accounts.account_number',
                                'AC.name as category',DB::raw("CONVERT(IFNULL(
SUM(IF(AT.type='credit', amount, -1*amount)), 0), INTEGER) as balance")])
                            ->groupBy('accounts.id')->first();

        $data = [
            'register_details' => $register_details,
            'details' => $details,
            'payment_types' => $payment_types,
            'accounts' => $accounts
        ];

        return $this->respondSuccess($data);
    }

    public function getCloseRegister()
    {
        $register_details =  $this->cashRegisterUtil->getRegisterDetails();

        $user_id = auth()->user()->id;
        $open_time = $register_details['open_time'];
        $close_time = \Carbon::now()->toDateTimeString();
        $details = $this->cashRegisterUtil->getRegisterTransactionDetails($user_id, $open_time,
$close_time);
        $payment_types = $this->cashRegisterUtil->payment_types();
        return view('cash_register.close_register_modal')
                    ->with(compact('register_details', 'details', 'payment_types'));
    }

    public function postCloseRegister(Request $request)
    {
        try {
            $input = $request->only(['total_card_slips', 'total_cheques',
                                     'closing_note']);

            $register_details =  $this->cashRegisterUtil->getRegisterDetails();
            $total = $register_details->cash_in_hand + $register_details->total_cash -
$register_details->total_cash_refund;

            $input['closing_amount'] = $this->cashRegisterUtil->num_uf($total);
            $user_id = Auth::id();
            $input['closed_at'] = \Carbon\Carbon::now()->format('Y-m-d H:i:s');
            $input['status'] = 'close';

            CashRegister::where('user_id', $user_id)
                                ->where('status', 'open')
                                ->update($input);

            return $this->respondSuccess(null, "Kasir berhasil ditutup");
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed($e->getMessage());
        }
    }
}
```

s. *Cashier Controller*

```
<?php

namespace App\Http\Controllers;

use Auth;
use App\Account;
use App\AccountTransaction;
use App\Brands;
use App\Business;
use App\BusinessLocation;
```

```
use App\Category;
use App\Contact;
use App\CustomerGroup;
use App\Media;
use App\Product;
use App\SellingPriceGroup;
use App\TaxRate;
use App\Transaction;
use App\TransactionSellLine;
use App\User;
use App\Utils\BusinessUtil;
use App\Utils\CashRegisterUtil;
use App\Utils\ContactUtil;
use App\Utils\ModuleUtil;
use App\Utils\DigitalUtil;
use App\Utils\NotificationUtil;
use App\Utils\AccountUtil;
use App\Utils\ProductUtil;
use App\Utils\TransactionUtil;
use App\Variation;
use App\VariationLocationDetails;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Storage;
use Yajra\DataTables\Facades\DataTables;
use Carbon\Carbon;

use App\Library\BimaSakti as BimaSakti;

class CashierController extends Controller
{
    protected $contactUtil;
    protected $productUtil;
    protected $businessUtil;
    protected $transactionUtil;
    protected $cashRegisterUtil;
    protected $moduleUtil;
    protected $notificationUtil;
    protected $accountUtil;

    public function __construct(
        ContactUtil $contactUtil,
        ProductUtil $productUtil,
        BusinessUtil $businessUtil,
        TransactionUtil $transactionUtil,
        CashRegisterUtil $cashRegisterUtil,
        ModuleUtil $moduleUtil,
        NotificationUtil $notificationUtil,
        AccountUtil $accountUtil
    ) {
        $this->contactUtil = $contactUtil;
        $this->productUtil = $productUtil;
        $this->businessUtil = $businessUtil;
        $this->transactionUtil = $transactionUtil;
        $this->cashRegisterUtil = $cashRegisterUtil;
        $this->moduleUtil = $moduleUtil;
        $this->notificationUtil = $notificationUtil;
        $this->accountUtil = $accountUtil;

        $this->dummyPaymentLine = ['method' => 'cash', 'amount' => 0, 'note' => '',
'card_transaction_number' => '', 'card_number' => '', 'card_type' => '', 'card_holder_name' =>
'', 'card_month' => '', 'card_year' => '', 'card_security' => '', 'cheque_number' => '',
'bank_account_number' => '',
        'is_return' => 0, 'transaction_no' => ''];
    }

    public function index()
    {
        if (!Auth::user()->can('sell.view') && !Auth::user()->can('sell.create')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = request()->session()->get('user.business_id');

        $business_locations = BusinessLocation::forDropdown($business_id, false);
        $customers = Contact::customersDropdown($business_id, false);

        $sales_representative = User::forDropdown($business_id, false, false, true);

        $is_cmsn_agent_enabled = request()->session()->get('business.sales_cmsn_agnt');
        $commission_agents = [];
```

```
        if (!empty($is_cmsn_agent_enabled)) {
            $commission_agents = User::forDropdown($business_id, false, true, true);
        }

        //Service staff filter
        $service_staffs = null;
        if ($this->productUtil->isModuleEnabled('service_staff')) {
            $service_staffs = $this->productUtil->serviceStaffDropdown($business_id);
        }

        return view('sale_pos.index')->with(compact('business_locations', 'customers',
'sales_representative', 'is_cmsn_agent_enabled', 'commission_agents', 'service_staffs'));
    }

    public function create()
    {
        if (!Auth::user()->can('sell.create')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = request()->session()->get('user.business_id');

        //Check if subscribed or not, then check for users quota
        if (!$this->moduleUtil->isSubscribed($business_id)) {
            return $this->moduleUtil->expiredResponse(action('HomeController@index'));
        } elseif (!$this->moduleUtil->isQuotaAvailable('invoices', $business_id)) {
            return $this->moduleUtil->quotaExpiredResponse('invoices', $business_id,
action('SellPosController@index'));
        }

        //Check if there is a open register, if no then redirect to Create Register screen.
        if ($this->cashRegisterUtil->countOpenedRegister() == 0) {
            return redirect()->action('CashRegisterController@create');
        }

        $walk_in_customer = $this->contactUtil->getWalkInCustomer($business_id);

        $business_details = $this->businessUtil->getDetails($business_id);
        $taxes = TaxRate::forBusinessDropdown($business_id, true, true);
        $payment_types = $this->productUtil->payment_types();

        $payment_lines[] = $this->dummyPaymentLine;

        $business_locations = BusinessLocation::forDropdown($business_id, false, true);
        $bl_attributes = $business_locations['attributes'];
        $business_locations = $business_locations['locations'];

        $default_location = null;
        if (count($business_locations) == 1) {
            foreach ($business_locations as $id => $name) {
                $default_location = $id;
            }
        }

        //Shortcuts
        $shortcuts = json_decode($business_details->keyboard_shortcuts, true);
        $pos_settings = empty($business_details->pos_settings) ? $this->businessUtil-
>defaultPosSettings() : json_decode($business_details->pos_settings, true);

        $commsn_agnt_setting = $business_details->sales_cmsn_agnt;
        $commission_agent = [];
        if ($commsn_agnt_setting == 'user') {
            $commission_agent = User::forDropdown($business_id, false);
        } elseif ($commsn_agnt_setting == 'cmsn_agnt') {
            $commission_agent = User::saleCommissionAgentsDropdown($business_id, false);
        }

        //If brands, category are enabled then send else false.
        $categories = (request()->session()->get('business.enable_category') == 1) ?
Category::catAndSubCategories($business_id) : false;
        $brands = (request()->session()->get('business.enable_brand') == 1) ?
Brands::where('business_id', $business_id)
                    ->pluck('name', 'id')
                    ->prepend(__('lang_v1.all_brands'), 'all') : false;

        $change_return = $this->dummyPaymentLine;

        $types = [];
        if (Auth::user()->can('supplier.create')) {
            $types['supplier'] = __('report.supplier');
        }
```

```
        if (Auth::user()->can('customer.create')) {
            $types['customer'] = __('report.customer');
        }
        if (Auth::user()->can('supplier.create') && Auth::user()->can('customer.create')) {
            $types['both'] = __('lang_v1.both_supplier_customer');
        }
        $customer_groups = CustomerGroup::forDropdown($business_id);

        //Accounts
        $accounts = [];
        if ($this->moduleUtil->isModuleEnabled('account')) {
            $accounts = Account::forDropdown($business_id, true, false);
        }
        //Selling Price Group Dropdown
        $price_groups = SellingPriceGroup::forDropdown($business_id);

        return view('sale_pos.create')
            ->with(compact(
                'business_details',
                'taxes',
                'payment_types',
                'walk_in_customer',
                'payment_lines',
                'business_locations',
                'bl_attributes',
                'default_location',
                'shortcuts',
                'commission_agent',
                'categories',
                'brands',
                'pos_settings',
                'change_return',
                'types',
                'customer_groups',
                'accounts',
                'price_groups'
            ));
    }

    public function store(Request $request)
    {
        if (!Auth::user()->can('sell.create') && !Auth::user()->can('direct_sell.access')) {
            return $this->respondFailed('Unauthorized action.', 403);
        }

        $is_direct_sale = false;

        //Check if there is a open register, if no then redirect to Create Register screen.
        if (!$is_direct_sale && $this->cashRegisterUtil->countOpenedRegister() == 0) {
            return $this->respondFailed('No Open Register', 410);
        }

        try {
            $input = $request->except('_token');
            $input['location_id'] = $request->input('location_id') ? $request-
>input('location_id') : "";

            $customer = Contact::findOrFail($input['contact_id']);

            $business = Auth::user()->business->currency;
            $currency_details = (object) [
                'thousand_separator' => $business->thousand_separator,
                'decimal_separator' => $business->decimal_separator
            ];

            $input['is_quotation'] = 0;
            $input['status'] = "final";
            //status is send as quotation from Add sales screen.
            if ($input['status'] == 'quotation') {
                $input['status'] = 'draft';
                $input['is_quotation'] = 1;
            }

            if (!empty($input['products'])) {
                $business_id = Auth::user()->business_id;
                $accountId = $this->accountUtil->getBalanceAccount($business_id);
                $final_total = 0;
                $change_return = 0;
                foreach($input['products'] as $key => $product) {
                    $subTotal = 0;
                    $product['variation_id'] = $product['id'];
```

```
                    $variation = $this->getPurchaseLine($product['variation_id'],
$input['location_id']);
                    if($variation == null) return $this->respondFailed("Produk tidak ditemukan",
404);

                    $input['products'][$key]['product_id'] = $variation->product_id;
                    $input['products'][$key]['product_type'] = $variation->type;


                    // $input['products'][$key]['unit_price'] = $subTotal;
                    $input['products'][$key]['variation_id'] = $variation->variation_id;
                    $input['products'][$key]['enable_stock'] = $variation->enable_stock;
                    $input['products'][$key]['product_unit_id'] = $variation->product_unit_id;
                    $input['products'][$key]['sub_unit_id'] = $variation->sub_unit_id;
                    // $input['products'][$key]['unit_price_inc_tax'] = $variation-
>unit_price_inc_tax;
                    $input['products'][$key]['base_unit_multiplier'] = 1;
                    $input['products'][$key]['sell_line_note'] = "";
                    $input['products'][$key]['tax_id'] = null;
                    $input['products'][$key]['item_tax'] = "0.00";
                    $final_total += $product['unit_price_inc_tax'] * $product['quantity'];
                }

                $input['discount_type'] = 'percentage';
                $final_total = $final_total - ($final_total * $input['discount_amount']/100);



                foreach($input['payments'] as $payment) {
                    $change_return += $this->transactionUtil->num_uf($payment['amount']) -
$final_total;
                }

                if($change_return < 0)
                    return $this->respondFailed("Maaf uang anda tidak cukup untuk melakukan
pembayaran : " . $final_total . " - ".$input['discount_amount']." - ".$change_return);

                $input['final_total'] = $final_total;
                $input['change_return'] = $change_return;
                $input['price_group'] = 0;
                $input['discount_type_modal'] = "percentage";
                $input['discount_amount_modal'] = "0.00";
                $input['shipping_details'] = "";
                $input['shipping_charges'] = "0.00";
                $input['sell_price_tax'] = "includes";
                $input['recur_interval_type'] = "days";
                $input['is_suspend'] = 0;
                $input['shipping_charges_modal'] = "0.00";
                $input['rp_redeemed_modal'] = 0;
                $input['tax_calculation_amount'] = "0.00";
                $input['tax_rate_id'] = null;
                $input['rp_redeemed_amount'] = 0;
                $input['rp_redeemed'] = 0;
                $input['payment_type'] = 'cash';
                $input['payments'][0]['method'] = "cash";
                $input['payments'][0]['note'] = "";
                $input['payments'][0]['account_id'] = Account::where('outlet_id',
$input['location_id'])->first()->id;

                if($input['payment_type'] == 'tempo') {
                    //Check Customer credit limit
                    $is_credit_limit_exeeded = $this->transactionUtil-
>isCustomerCreditLimitExeeded($input);

                    if ($is_credit_limit_exeeded !== false) {
                        $credit_limit_amount = $this->transactionUtil-
>num_f($is_credit_limit_exeeded, true, $currency_details);
                        return $this->respondFailed("Customer Credit limit exceeded. Credit
Limit: ".$credit_limit_amount);
                    }
                }

                if(!$this->moduleUtil->isBalanceAvailable($business_id)) {
                    return $this->respondFailed("Maaf Saldo Kipa anda tidak mencukupi untuk
melakukan penjualan, ayo topup Saldo Kipa mu!");
                }

                if(!$this->moduleUtil->isPaidToday($business_id, $input['location_id'])){
                    //insert biaya pengeluaran as service cost
                    $bonusBalance = $this->accountUtil->getSubBalance($accountId, 'bonus');
                    $mainBalance = $this->accountUtil->getSubBalance($accountId);
```

```
                    $amount = 3850;

                    if($bonusBalance >= 3850) {
                        $this->transactionUtil->createExpenseTransaction($business_id,
$input['location_id'], $amount, $accountId, 'Pembayaran Biaya Layanan', 'saldo', 'bonus');
                    } else {
                        $useBonusBalance = ($bonusBalance > 0);

                        if($useBonusBalance){
                            $this->transactionUtil->createExpenseTransaction($business_id,
$input['location_id'], $bonusBalance, $accountId, 'Pembayaran Biaya Layanan', 'saldo', 'bonus');
                            $amount = 3850 - $bonusBalance;
                        }

                        $this->transactionUtil->createExpenseTransaction($business_id,
$input['location_id'], $amount, $accountId, 'Pembayaran Biaya Layanan', 'saldo', 'topup');
                    }
                }


                $user_id = Auth::id();

                $discount = ['discount_type' => $input['discount_type'],
                            'discount_amount' => $input['discount_amount']
                        ];
                $invoice_total = $this->productUtil->calculateInvoiceTotal($input['products'],
$input['tax_rate_id'], $discount);

                DB::beginTransaction();

                if (empty($request->input('transaction_date'))) {
                    $input['transaction_date'] = Carbon::now();
                } else {
                    $input['transaction_date'] = $this->productUtil->uf_date($request-
>input('transaction_date'), true);
                }
                if ($is_direct_sale) {
                    $input['is_direct_sale'] = 1;
                }

                //Set commission agent
                $input['commission_agent'] = !empty($request->input('commission_agent')) ?
$request->input('commission_agent') : null;
                $commsn_agnt_setting = Auth::user()->business->sales_cmsn_agnt;//$request-
>session()->get('business.sales_cmsn_agnt');

                if ($commsn_agnt_setting == 'logged_in_user') {
                    $input['commission_agent'] = $user_id;
                }

                if (isset($input['exchange_rate']) && $this->transactionUtil-
>num_uf($input['exchange_rate'], $currency_details) == 0) {
                    $input['exchange_rate'] = 1;
                }

                //Customer group details
                $contact_id = $request->get('contact_id', null);
                $cg = $this->contactUtil->getCustomerGroup($business_id, $contact_id);
                $input['customer_group_id'] = (empty($cg) || empty($cg->id)) ? null : $cg->id;

                //set selling price group id
                if ($request->has('price_group')) {
                    $input['selling_price_group_id'] = $request->input('price_group');
                }

                $input['is_suspend'] = isset($input['is_suspend']) && 1 == $input['is_suspend']
? 1 : 0;
                if ($input['is_suspend']) {
                    $input['sale_note'] = !empty($input['additional_notes']) ?
$input['additional_notes'] : null;
                }

                //Generate reference number
                if (!empty($input['is_recurring'])) {
                    //Update reference count
                    $ref_count = $this->transactionUtil->setAndGetReferenceCount('subscription');
                    $input['subscription_no'] = $this->transactionUtil-
>generateReferenceNumber('subscription', $ref_count);
                }

                if ($is_direct_sale) {
```

```
                    $input['invoice_scheme_id'] = $request->input('invoice_scheme_id');
                }

                $transaction = $this->transactionUtil->createSellTransaction($business_id,
$input, $invoice_total, $user_id);

                $this->transactionUtil->createOrUpdateSellLines($transaction, $input['products'],
$input['location_id']);

                if (!$is_direct_sale) {
                    //Add change return
                    $change_return = $this->dummyPaymentLine;
                    $change_return['amount'] = $input['change_return'];
                    $change_return['type'] = 'debit';
                    $change_return['is_return'] = 1;
                    $change_return['account_id'] = $input['payments'][0]['account_id'];
                    $input['payments'][] = $change_return;
                }

                if (!$transaction->is_suspend && !empty($input['payments'])) {
                    $this->transactionUtil->createOrUpdatePaymentLines($transaction,
$input['payments']);
                }

                $update_transaction = false;
                if ($this->transactionUtil->isModuleEnabled('tables')) {
                    $transaction->res_table_id = request()->get('res_table_id');
                    $update_transaction = true;
                }
                if ($this->transactionUtil->isModuleEnabled('service_staff')) {
                    $transaction->res_waiter_id = request()->get('res_waiter_id');
                    $update_transaction = true;
                }
                if ($update_transaction) {
                    $transaction->save();
                }

                //Check for final and do some processing.
                if ($input['status'] == 'final') {
                    //update product stock
                    foreach ($input['products'] as $product) {
                        $product['variation_id'] = $product['id'];
                        $decrease_qty = $this->productUtil
                                    ->num_uf($product['quantity']);
                        if (!empty($product['base_unit_multiplier'])) {
                            $decrease_qty = $decrease_qty * $product['base_unit_multiplier'];
                        }

                        if ($product['enable_stock']) {
                            $this->productUtil->decreaseProductQuantity(
                                $product['product_id'],
                                $product['variation_id'],
                                $input['location_id'],
                                $decrease_qty
                            );
                        }

                        if ($product['product_type'] == 'combo') {
                            //Decrease quantity of combo as well.
                            $this->productUtil
                                ->decreaseProductQuantityCombo(
                                    $product['combo'],
                                    $input['location_id']
                                );
                        }
                    }

                    //Add payments to Cash Register
                    if (!$is_direct_sale && !$transaction->is_suspend &&
!empty($input['payments'])) {
                        $this->cashRegisterUtil->addSellPayments($transaction,
$input['payments']);
                    }
                    //Update payment status
                    $this->transactionUtil->updatePaymentStatus($transaction->id, $transaction-
>final_total);

                    if (Auth::user()->business->enable_rp == 1) {
                        $redeemed = !empty($input['rp_redeemed']) ? $input['rp_redeemed'] : 0;
                        $this->transactionUtil->updateCustomerRewardPoints($contact_id,
$transaction->rp_earned, 0, $redeemed);
```

```
                    }

                    //Allocate the quantity from purchase and add mapping of
                    //purchase & sell lines in
                    //transaction_sell_lines_purchase_lines table
                    $business_details = $this->businessUtil->getDetails($business_id);
                    $pos_settings = empty($business_details->pos_settings) ? $this->businessUtil-
>defaultPosSettings() : json_decode($business_details->pos_settings, true);

                    $business = ['id' => $business_id,
                                 'accounting_method' => Auth::user()->business-
>accounting_method,//$request->session()->get('business.accounting_method'),
                                 'location_id' => $input['location_id'],
                                 'pos_settings' => $pos_settings
                                ];
                    $this->transactionUtil->mapPurchaseSell($business, $transaction->sell_lines,
'purchase');

                    //Auto send notification
                    $this->notificationUtil->autoSendNotification($business_id, 'new_sale',
$transaction, $transaction->contact);
                }

                //Set Module fields
                if (!empty($input['has_module_data'])) {
                    $this->moduleUtil->getModuleData('after_sale_saved', ['transaction' =>
$transaction, 'input' => $input]);
                }

                Media::uploadMedia($business_id, $transaction, $request, 'documents');

                $neraca = $this->transactionUtil->getBalanceSheet();
                $transaction->neraca = $neraca;

                DB::commit();

                activity('transaction')
                    ->causedBy(Auth::user())
                    ->performedOn($transaction)
                    ->withProperties($transaction)
                    ->log('created');

                $data = [
                    'id' => $transaction->id,
                    'final_total' => $final_total,
                    'invoice_no' => $transaction->invoice_no,
                    'charge_return' => $input['change_return'],
                    'discount_amount' => (float)$input['discount_amount']
                ];

                return $this->respondSuccess($data, "Pembayaran Berhasil");
            } else {
                return $this->respondFailed();
            }
        } catch (\Exception $e) {
            DB::rollBack();

            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());
            $msg = '';

            if (get_class($e) == \App\Exceptions\PurchaseSellMismatch::class) {
                $msg = $e->getMessage();
            }
            return $this->respondFailed($e->getMessage());
        }
    }

    public function storeRemote(Request $request)
    {

        $is_direct_sale = false;

        try {
            $input = $request->all();
            $input['location_id'] = $request->input('location_id') ? $request-
>input('location_id') : "";

            $customer = Contact::findOrFail($input['contact_id']);

            $business = Auth::user()->business->currency;
```

```
            $currency_details = (object) [
                'thousand_separator' => $business->thousand_separator,
                'decimal_separator' => $business->decimal_separator
            ];

            $input['is_quotation'] = 0;
            $input['status'] = "final";
            if ($input['data']['STATUS'] == '00') {
                $keterangan = strtolower($input['data']['KET']);
                if(strpos($keterangan, 'sedang diproses') !== false) {
                    $input['digital_status'] = "pending";
                } else {
                    $input['digital_status'] = "success";
                }
            } else {
                $input['digital_status'] = "pending";
            }

            $input['id_transaction_digital'] = $input['data']['REF2'];

            if (!empty($input['products'])) {
                $business_id = Auth::user()->business_id;
                $final_total = 0;
                $change_return = 0;
                foreach($input['products'] as $key => $product) {
                    $variation = $this->getPurchaseLine($product['variation_id'],
$input['location_id']);
                    if($variation == null) return $this->respondFailed("Produk tidak ditemukan",
404);
                    $input['products'][$key]['product_id'] = $variation->product_id;
                    $input['products'][$key]['product_type'] = $variation->type;
                    $input['products'][$key]['unit_price'] = $request->input('final_total_sell')
* $product['quantity'];
                    $input['products'][$key]['variation_id'] = $variation->variation_id;
                    $input['products'][$key]['enable_stock'] = $variation->enable_stock;
                    $input['products'][$key]['product_unit_id'] = $variation->product_unit_id;
                    $input['products'][$key]['sub_unit_id'] = $variation->sub_unit_id;
                    $input['products'][$key]['unit_price_inc_tax'] = $request-
>input('final_total_sell');
                    $input['products'][$key]['base_unit_multiplier'] = 1;
                    $input['products'][$key]['sell_line_note'] = "";
                    $input['products'][$key]['tax_id'] = null;
                    $input['products'][$key]['item_tax'] = "0.00";
                    $final_total += ($product['quantity'] * $request->input('final_total_sell'));
                }

                $input['discount_type'] = 'percentage';
                $final_total = $final_total - ($final_total * $input['discount_amount']/100);

                if($customer->customer_group) {
                    $final_total = $final_total - ($final_total * $customer->customer_group-
>amount/100);
                }

                foreach($input['payments'] as $payment) {
                    $change_return += $this->transactionUtil->num_uf($payment['amount']) -
$final_total;
                }

                $input['final_total'] = $final_total;
                $input['change_return'] = $change_return;
                $input['price_group'] = 0;
                $input['discount_type_modal'] = "percentage";
                $input['discount_amount_modal'] = "0.00";
                $input['shipping_details'] = "";
                $input['shipping_charges'] = "0.00";
                $input['sell_price_tax'] = "includes";
                $input['recur_interval_type'] = "days";
                $input['is_suspend'] = 0;
                $input['shipping_charges_modal'] = "0.00";
                $input['rp_redeemed_modal'] = 0;
                $input['tax_calculation_amount'] = "0.00";
                $input['tax_rate_id'] = null;
                $input['rp_redeemed_amount'] = 0;
                $input['rp_redeemed'] = 0;
                $input['payment_type'] = 'cash';
                $input['additional_notes'] = $request->input('additional_notes');
                $input['payments'][0]['method'] = "cash";
                $input['payments'][0]['note'] = "";
                $input['payments'][0]['account_id'] = Account::where('outlet_id',
$input['location_id'])->first()->id;
```

```
                $user_id = Auth::id();

                $discount = ['discount_type' => $input['discount_type'],
                                'discount_amount' => $input['discount_amount']
                            ];
                $invoice_total = $this->productUtil->calculateInvoiceTotal($input['products'],
$input['tax_rate_id'], $discount);

                DB::beginTransaction();

                if (empty($request->input('transaction_date'))) {
                    $input['transaction_date'] =  Carbon::now();
                } else {
                    $input['transaction_date'] = $this->productUtil->uf_date($request-
>input('transaction_date'), true);
                }

                //Set commission agent
                $input['commission_agent'] = !empty($request->input('commission_agent')) ?
$request->input('commission_agent') : null;
                $commsn_agnt_setting = Auth::user()->business->sales_cmsn_agnt;//$request-
>session()->get('business.sales_cmsn_agnt');

                if ($commsn_agnt_setting == 'logged_in_user') {
                    $input['commission_agent'] = $user_id;
                }

                if (isset($input['exchange_rate']) && $this->transactionUtil-
>num_uf($input['exchange_rate'], $currency_details) == 0) {
                    $input['exchange_rate'] = 1;
                }

                //Customer group details
                $contact_id = $request->get('contact_id', null);
                $cg = $this->contactUtil->getCustomerGroup($business_id, $contact_id);
                $input['customer_group_id'] = (empty($cg) || empty($cg->id)) ? null : $cg->id;

                //set selling price group id
                if ($request->has('price_group')) {
                    $input['selling_price_group_id'] = $request->input('price_group');
                }

                $input['is_suspend'] = isset($input['is_suspend']) && 1 == $input['is_suspend']
? 1 : 0;
                if ($input['is_suspend']) {
                    $input['sale_note'] = !empty($input['additional_notes']) ?
$input['additional_notes'] : null;
                }

                //Generate reference number
                if (!empty($input['is_recurring'])) {
                    //Update reference count
                    $ref_count = $this->transactionUtil->setAndGetReferenceCount('subscription');
                    $input['subscription_no'] = $this->transactionUtil-
>generateReferenceNumber('subscription', $ref_count);
                }

                if ($is_direct_sale) {
                    $input['invoice_scheme_id'] = $request->input('invoice_scheme_id');
                }

                // create transaction
                $transaction = $this->transactionUtil->createSellTransaction($business_id,
$input, $invoice_total, $user_id);

                $this->transactionUtil->createOrUpdateSellLines($transaction, $input['products'],
$input['location_id']);

                if (!$is_direct_sale) {
                    //Add change return
                    $change_return = $this->dummyPaymentLine;
                    $change_return['amount'] = $input['change_return'];
                    $change_return['type'] = 'debit';
                    $change_return['is_return'] = 1;
                    $change_return['account_id'] = $input['payments'][0]['account_id'];
                    $input['payments'][] = $change_return;
                }

                if (!$transaction->is_suspend && !empty($input['payments'])) {
```

```
                    $this->transactionUtil->createOrUpdatePaymentLines($transaction,
$input['payments']);
                }

                $update_transaction = false;
                if ($this->transactionUtil->isModuleEnabled('tables')) {
                    $transaction->res_table_id = request()->get('res_table_id');
                    $update_transaction = true;
                }

                if ($this->transactionUtil->isModuleEnabled('service_staff')) {
                    $transaction->res_waiter_id = request()->get('res_waiter_id');
                    $update_transaction = true;
                }

                if ($update_transaction) {
                    $transaction->save();
                }

                //Check for final and do some processing.
                if ($input['status'] == 'final') {
                    //update product stock
                    foreach ($input['products'] as $product) {
                        $decrease_qty = $this->productUtil
                            ->num_uf($product['quantity']);
                        if (!empty($product['base_unit_multiplier'])) {
                            $decrease_qty = $decrease_qty * $product['base_unit_multiplier'];
                        }

                        if ($product['enable_stock']) {
                            $this->productUtil->decreaseProductQuantity(
                                $product['product_id'],
                                $product['variation_id'],
                                $input['location_id'],
                                $decrease_qty
                            );
                        }

                        if ($product['product_type'] == 'combo') {
                            //Decrease quantity of combo as well.
                            $this->productUtil
                                ->decreaseProductQuantityCombo(
                                    $product['combo'],
                                    $input['location_id']
                                );
                        }
                    }

                    //Add payments to Cash Register
                    if (!$is_direct_sale && !$transaction->is_suspend &&
!empty($input['payments'])) {
                        $this->cashRegisterUtil->addSellPayments($transaction,
$input['payments']);
                    }
                    //Update payment status
                    $this->transactionUtil->updatePaymentStatus($transaction->id, $transaction-
>final_total);

                    if (Auth::user()->business->enable_rp == 1) {
                        $redeemed = !empty($input['rp_redeemed']) ? $input['rp_redeemed'] : 0;
                        $this->transactionUtil->updateCustomerRewardPoints($contact_id,
$transaction->rp_earned, 0, $redeemed);
                    }

                    //Allocate the quantity from purchase and add mapping of
                    //purchase & sell lines in
                    //transaction_sell_lines_purchase_lines table
                    $business_details = $this->businessUtil->getDetails($business_id);
                    $pos_settings = empty($business_details->pos_settings) ? $this->businessUtil-
>defaultPosSettings() : json_decode($business_details->pos_settings, true);

                    $business = ['id' => $business_id,
                                 'accounting_method' => Auth::user()->business-
>accounting_method,//$request->session()->get('business.accounting_method'),
                                 'location_id' => $input['location_id'],
                                 'pos_settings' => $pos_settings
                                ];
                    $this->transactionUtil->mapPurchaseSell($business, $transaction->sell_lines,
'purchase', true, null, $request->input('modal_satuan'));

                    //Auto send notification
```

```
                $this->notificationUtil->autoSendNotification($business_id, 'new_sale',
$transaction, $transaction->contact);
            }

            //Set Module fields
            if (!empty($input['has_module_data'])) {
                $this->moduleUtil->getModuleData('after_sale_saved', ['transaction' =>
$transaction, 'input' => $input]);
            }

            Media::uploadMedia($business_id, $transaction, $request, 'documents');

            $neraca = $this->transactionUtil->getBalanceSheet();
            $transaction->neraca = $neraca;

            DB::commit();

            activity('transaction')
                ->causedBy(Auth::user())
                ->performedOn($transaction)
                ->withProperties($transaction)
                ->log('created');

            $data = [
                'id' => $transaction->id,
                'final_total' => $final_total,
                'invoice_no' => $transaction->invoice_no,
                'charge_return' => $input['change_return']
            ];

            $product = Product::find($product['product_id']);

            if ($product->product_type == 'digital') {
                $data['digital_status'] = $transaction->digital_status;
                $data['result'] = $input['data'];
                $data['notes'] = $input['sale_note'];
            }

            return $this->respondSuccess($data, "Pembayaran Berhasil");
        } else {
            return $this->respondFailed();
        }
    } catch (\Exception $e) {
        DB::rollBack();

        \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());
        $msg = '';

        if (get_class($e) == \App\Exceptions\PurchaseSellMismatch::class) {
            $msg = $e->getMessage();
        }
        return $this->respondFailed($e->getMessage());
    }
}

public function inquiry(Request $request)
{
    try {
        $business = Auth::user()->business;

        $sku = $request->input('sku');
        $msisdn = $request->input('msisdn');
        $digital_type = $request->input('digital_type', 'pascabayar');

        $product = Product::where('business_id', $business->id)
                    ->where('product_type', 'digital')
                    ->where('digital_type', $digital_type)
                    ->where('sku', $sku)
                    ->firstOrFail();

        $client = new BimaSakti(env('BIMASAKTI_ID'), env('BIMASAKTI_PIN'));

        if(strpos($sku, 'PLN') !== false) {
            $response = $client->inquiryPLN($product->product_custom_field1, "", $msisdn);
        } else {
            $response = $client->inquiry($product->product_custom_field1,$msisdn);
        }

        if($response["success"]) {
```

```
                // is dpp < adm ?
                $fee = (int)$product->variations[0]->default_sell_price;

                $responseReal = [
                    'product_id' => $product->id,
                    'msisdn' => $msisdn,
                    'fee' => $fee,
                    'inquiry' => $response["data"]
                ];

                return $this->respondSuccess($responseReal, "Tagihan ditemukan");
            } else {
                return $this->respondFailed(json_encode($response));
            }
        } catch(\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());
            return $this->respondFailed('Sepertinya ada yang salah');
        }
    }

    private function receiptContent(
        $business_id,
        $location_id,
        $transaction_id,
        $printer_type = null,
        $is_package_slip = false,
        $from_pos_screen = true
    ) {
        $output = ['is_enabled' => false,
                    'print_type' => 'browser',
                    'html_content' => null,
                    'printer_config' => [],
                    'data' => []
                ];


        $business_details = $this->businessUtil->getDetails($business_id);
        $location_details = BusinessLocation::find($location_id);

        if ($from_pos_screen && $location_details->print_receipt_on_invoice != 1) {
            return $output;
        }
        //Check if printing of invoice is enabled or not.
        //If enabled, get print type.
        $output['is_enabled'] = true;

        $invoice_layout = $this->businessUtil->invoiceLayout($business_id, $location_id,
$location_details->invoice_layout_id);

        //Check if printer setting is provided.
        $receipt_printer_type = is_null($printer_type) ? $location_details->receipt_printer_type
: $printer_type;

        $receipt_details = $this->transactionUtil->getReceiptDetails($transaction_id,
$location_id, $invoice_layout, $business_details, $location_details, $receipt_printer_type);

        $currency_details = [
            'symbol' => $business_details->currency_symbol,
            'thousand_separator' => $business_details->thousand_separator,
            'decimal_separator' => $business_details->decimal_separator,
        ];
        $receipt_details->currency = $currency_details;

        if ($is_package_slip) {
            $output['html_content'] = view('sale_pos.receipts.packing_slip',
compact('receipt_details'))->render();
            return $output;
        }
        //If print type browser - return the content, printer - return printer config data, and
invoice format config
        if ($receipt_printer_type == 'printer') {
            $output['print_type'] = 'printer';
            $output['printer_config'] = $this->businessUtil->printerConfig($business_id,
$location_details->printer_id);
            $output['data'] = $receipt_details;
        } else {
            $layout = !empty($receipt_details->design) ? 'sale_pos.receipts.' . $receipt_details-
>design : 'sale_pos.receipts.classic';

            $output['html_content'] = view($layout, compact('receipt_details'))->render();
```

```
        }

        return $output;
    }

    public function show($id, Request $request)
    {
        $business_id = Auth::user()->business_id;
        $location_id = $request->input('location_id');

        $business = $this->businessUtil->getDetails($business_id);
        $outlet = BusinessLocation::find($location_id);

        $invoice_layout = $this->businessUtil->invoiceLayout($business_id, $location_id, $outlet->invoice_layout_id);
        $receipt_details = $this->transactionUtil->getReceiptDetails($id, $location_id, $invoice_layout, $business, $outlet, $outlet->receipt_printer_type);
        $currency_details = [
            'symbol' => $business->currency_symbol,
            'thousand_separator' => $business->thousand_separator,
            'decimal_separator' => $business->decimal_separator,
        ];
        $receipt_details->currency = $currency_details;
        $receipt_details->operator_name = Auth::user()->first_name;

        return $this->respondSuccess($receipt_details);
    }

    public function edit($id)
    {
        if (!Auth::user()->can('sell.update')) {
            abort(403, 'Unauthorized action.');
        }

        //Check if the transaction can be edited or not.
        $edit_days = request()->session()->get('business.transaction_edit_days');
        if (!$this->transactionUtil->canBeEdited($id, $edit_days)) {
            return back()
                ->with('status', ['success' => 0,
                    'msg' => __('messages.transaction_edit_not_allowed', ['days' => $edit_days])]);
        }

        //Check if there is a open register, if no then redirect to Create Register screen.
        if ($this->cashRegisterUtil->countOpenedRegister() == 0) {
            return redirect()->action('CashRegisterController@create');
        }

        //Check if return exist then not allowed
        if ($this->transactionUtil->isReturnExist($id)) {
            return back()->with('status', ['success' => 0,
                    'msg' => __('lang_v1.return_exist')]);
        }

        $business_id = request()->session()->get('user.business_id');
        $walk_in_customer = $this->contactUtil->getWalkInCustomer($business_id);

        $business_details = $this->businessUtil->getDetails($business_id);
        $taxes = TaxRate::forBusinessDropdown($business_id, true, true);
        $payment_types = $this->productUtil->payment_types();

        $transaction = Transaction::where('business_id', $business_id)
                            ->where('type', 'sell')
                            ->findorfail($id);

        $location_id = $transaction->location_id;
        $location_printer_type = BusinessLocation::find($location_id)->receipt_printer_type;

        $sell_details = TransactionSellLine::
                        join(
                            'products AS p',
                            'transaction_sell_lines.product_id',
                            '=',
                            'p.id'
                        )
                        ->join(
                            'variations AS variations',
                            'transaction_sell_lines.variation_id',
                            '=',
                            'variations.id'
                        )
```

```
                        ->join(
                            'product_variations AS pv',
                            'variations.product_variation_id',
                            '=',
                            'pv.id'
                        )
                        ->leftjoin('variation_location_details AS vld', function ($join) use
($location_id) {
                            $join->on('variations.id', '=', 'vld.variation_id')
                                ->where('vld.location_id', '=', $location_id);
                        })
                        ->leftjoin('units', 'units.id', '=', 'p.unit_id')
                        ->where('transaction_sell_lines.transaction_id', $id)
                        ->select(
                            DB::raw("IF(pv.is_dummy = 0, CONCAT(p.name, ' (', pv.name,
':',variations.name, ')'), p.name) AS product_name"),
                            'p.id as product_id',
                            'p.enable_stock',
                            'p.name as product_actual_name',
                            'p.type as product_type',
                            'pv.name as product_variation_name',
                            'pv.is_dummy as is_dummy',
                            'variations.name as variation_name',
                            'variations.sub_sku',
                            'p.barcode_type',
                            'p.enable_sr_no',
                            'variations.id as variation_id',
                            'units.short_name as unit',
                            'units.allow_decimal as unit_allow_decimal',
                            'transaction_sell_lines.tax_id as tax_id',
                            'transaction_sell_lines.item_tax as item_tax',
                            'transaction_sell_lines.unit_price as default_sell_price',
                            'transaction_sell_lines.unit_price_before_discount as
unit_price_before_discount',
                            'transaction_sell_lines.unit_price_inc_tax as sell_price_inc_tax',
                            'transaction_sell_lines.id as transaction_sell_lines_id',
                            'transaction_sell_lines.quantity as quantity_ordered',
                            'transaction_sell_lines.sell_line_note as sell_line_note',
                            'transaction_sell_lines.parent_sell_line_id',
                            'transaction_sell_lines.lot_no_line_id',
                            'transaction_sell_lines.line_discount_type',
                            'transaction_sell_lines.line_discount_amount',
                            'transaction_sell_lines.res_service_staff_id',
                            'units.id as unit_id',
                            'transaction_sell_lines.sub_unit_id',
                            DB::raw('vld.qty_available + transaction_sell_lines.quantity AS
qty_available')
                        )
                        ->get();
        if (!empty($sell_details)) {
            foreach ($sell_details as $key => $value) {

                //If modifier or combo sell line then unset
                if (!empty($sell_details[$key]->parent_sell_line_id)) {
                    unset($sell_details[$key]);
                } else {
                    if ($transaction->status != 'final') {
                        $actual_qty_avlbl = $value->qty_available - $value->quantity_ordered;
                        $sell_details[$key]->qty_available = $actual_qty_avlbl;
                        $value->qty_available = $actual_qty_avlbl;
                    }

                    $sell_details[$key]->formatted_qty_available = $this->productUtil-
>num_f($value->qty_available, false, null, true);

                    //Add available lot numbers for dropdown to sell lines
                    $lot_numbers = [];
                    if (request()->session()->get('business.enable_lot_number') == 1 ||
request()->session()->get('business.enable_product_expiry') == 1) {
                        $lot_number_obj = $this->transactionUtil-
>getLotNumbersFromVariation($value->variation_id, $business_id, $location_id);
                        foreach ($lot_number_obj as $lot_number) {
                            //If lot number is selected added ordered quantity to lot quantity
available
                            if ($value->lot_no_line_id == $lot_number->purchase_line_id) {
                                $lot_number->qty_available += $value->quantity_ordered;
                            }

                            $lot_number->qty_formated = $this->productUtil->num_f($lot_number-
>qty_available);
                            $lot_numbers[] = $lot_number;
```

```
                        }
                    }
                    $sell_details[$key]->lot_numbers = $lot_numbers;

                    if (!empty($value->sub_unit_id)) {
                        $value = $this->productUtil->changeSellLineUnit($business_id, $value);
                        $sell_details[$key] = $value;
                    }

                    $sell_details[$key]->formatted_qty_available = $this->productUtil-
>num_f($value->qty_available, false, null, true);

                    if ($this->transactionUtil->isModuleEnabled('modifiers')) {
                        //Add modifier details to sel line details
                        $sell_line_modifiers = TransactionSellLine::where('parent_sell_line_id',
$sell_details[$key]->transaction_sell_lines_id)
                            ->where('children_type', 'modifier')
                            ->get();
                        $modifiers_ids = [];
                        if (count($sell_line_modifiers) > 0) {
                            $sell_details[$key]->modifiers = $sell_line_modifiers;
                            foreach ($sell_line_modifiers as $sell_line_modifier) {
                                $modifiers_ids[] = $sell_line_modifier->variation_id;
                            }
                        }
                        $sell_details[$key]->modifiers_ids = $modifiers_ids;

                        //add product modifier sets for edit
                        $this_product = Product::find($sell_details[$key]->product_id);
                        if (count($this_product->modifier_sets) > 0) {
                            $sell_details[$key]->product_ms = $this_product->modifier_sets;
                        }
                    }

                    //Get details of combo items
                    if ($sell_details[$key]->product_type == 'combo') {
                        $sell_line_combos = TransactionSellLine::where('parent_sell_line_id',
$sell_details[$key]->transaction_sell_lines_id)
                            ->where('children_type', 'combo')
                            ->get()
                            ->toArray();
                        if (!empty($sell_line_combos)) {
                            $sell_details[$key]->combo_products = $sell_line_combos;
                        }

                        //calculate quantity available if combo product
                        $combo_variations = [];
                        foreach ($sell_line_combos as $combo_line) {
                            $combo_variations[] = [
                                'variation_id' => $combo_line['variation_id'],
                                'quantity' => $combo_line['quantity'] / $sell_details[$key]-
>quantity_ordered,
                                'unit_id' => null
                            ];
                        }
                        $sell_details[$key]->qty_available =
                        $this->productUtil->calculateComboQuantity($location_id,
$combo_variations);

                        if ($transaction->status == 'final') {
                            $sell_details[$key]->qty_available = $sell_details[$key]-
>qty_available + $sell_details[$key]->quantity_ordered;
                        }

                        $sell_details[$key]->formatted_qty_available = $this->productUtil-
>num_f($sell_details[$key]->qty_available, false, null, true);
                    }
                }
            }
        }

        $payment_lines = $this->transactionUtil->getPaymentDetails($id);
        //If no payment lines found then add dummy payment line.
        if (empty($payment_lines)) {
            $payment_lines[] = $this->dummyPaymentLine;
        }

        $shortcuts = json_decode($business_details->keyboard_shortcuts, true);
        $pos_settings = empty($business_details->pos_settings) ? $this->businessUtil-
>defaultPosSettings() : json_decode($business_details->pos_settings, true);
```

```
        $commsn_agnt_setting = $business_details->sales_cmsn_agnt;
        $commission_agent = [];
        if ($commsn_agnt_setting == 'user') {
            $commission_agent = User::forDropdown($business_id, false);
        } elseif ($commsn_agnt_setting == 'cmsn_agnt') {
            $commission_agent = User::saleCommissionAgentsDropdown($business_id, false);
        }

        //If brands, category are enabled then send else false.
        $categories = (request()->session()->get('business.enable_category') == 1) ?
Category::catAndSubCategories($business_id) : false;
        $brands = (request()->session()->get('business.enable_brand') == 1) ?
Brands::where('business_id', $business_id)
                    ->pluck('name', 'id')
                    ->prepend(__('lang_v1.all_brands'), 'all') : false;

        $change_return = $this->dummyPaymentLine;

        $types = [];
        if (Auth::user()->can('supplier.create')) {
            $types['supplier'] = __('report.supplier');
        }
        if (Auth::user()->can('customer.create')) {
            $types['customer'] = __('report.customer');
        }
        if (Auth::user()->can('supplier.create') && Auth::user()->can('customer.create')) {
            $types['both'] = __('lang_v1.both_supplier_customer');
        }
        $customer_groups = CustomerGroup::forDropdown($business_id);

        //Accounts
        $accounts = [];
        if ($this->moduleUtil->isModuleEnabled('account')) {
            $accounts = Account::forDropdown($business_id, true, false);
        }
        //Selling Price Group Dropdown
        $price_groups = SellingPriceGroup::forDropdown($business_id);

        $waiters = [];
        if ($this->productUtil->isModuleEnabled('service_staff') &&
!empty($pos_settings['inline_service_staff'])) {
            $waiters_enabled = true;
            $waiters = $this->productUtil->serviceStaffDropdown($business_id);
        }
        $redeem_details = [];
        if (request()->session()->get('business.enable_rp') == 1) {
            $redeem_details = $this->transactionUtil->getRewardRedeemDetails($business_id,
$transaction->contact_id);

            $redeem_details['points'] += $transaction->rp_redeemed;
            $redeem_details['points'] -= $transaction->rp_earned;
        }

        $edit_discount = Auth::user()->can('edit_product_discount_from_pos_screen');
        $edit_price = Auth::user()->can('edit_product_price_from_pos_screen');

        return view('sale_pos.edit')
            ->with(compact('business_details', 'taxes', 'payment_types', 'walk_in_customer',
'sell_details', 'transaction', 'payment_lines', 'location_printer_type', 'shortcuts',
'commission_agent', 'categories', 'pos_settings', 'change_return', 'types', 'customer_groups',
'brands', 'accounts', 'price_groups', 'waiters', 'redeem_details', 'edit_price',
'edit_discount'));
    }

    public function update(Request $request, $id)
    {
        if (!Auth::user()->can('sell.update') && !Auth::user()->can('direct_sell.access')) {
            return $this->respondFailed('Unauthorized action.', 403);
        }

        try {
            $input = $request->except('_token');

            //status is send as quotation from edit sales screen.
            $input['is_quotation'] = 0;
            if ($input['status'] == 'quotation') {
                $input['status'] = 'draft';
                $input['is_quotation'] = 1;
            }

            $is_direct_sale = false;
```

```
            if (!empty($input['products'])) {
                //Get transaction value before updating.
                $transaction_before = Transaction::find($id);
                $status_before =  $transaction_before->status;
                $rp_earned_before = $transaction_before->rp_earned;
                $rp_redeemed_before = $transaction_before->rp_redeemed;

                if ($transaction_before->is_direct_sale == 1) {
                    $is_direct_sale = true;
                }

                //Check Customer credit limit
                $is_credit_limit_exeeded = $this->transactionUtil-
>isCustomerCreditLimitExeeded($input, $id);

                if ($is_credit_limit_exeeded !== false) {
                    $credit_limit_amount = $this->transactionUtil-
>num_f($is_credit_limit_exeeded, true);
                    return $this->respondFailed("Customer Credit limit exceeded. Credit Limit:
".$credit_limit_amount);
                }

                //Check if there is a open register, if no then redirect to Create Register
screen.
                if (!$is_direct_sale && $this->cashRegisterUtil->countOpenedRegister() == 0) {
                    return $this->respondFailed('No Open Register', 410);
                }

                $business_id = Auth::user()->business_id;
                $user_id = Auth::id();
                $commsn_agnt_setting = Auth::user()->business->sales_cmsn_agnt;

                $discount = ['discount_type' => $input['discount_type'],
                                'discount_amount' => $input['discount_amount']
                            ];
                $invoice_total = $this->productUtil->calculateInvoiceTotal($input['products'],
$input['tax_rate_id'], $discount);

                if (!empty($request->input('transaction_date'))) {
                    $input['transaction_date'] = $this->productUtil->uf_date($request-
>input('transaction_date'), true);
                }

                $input['commission_agent'] = !empty($request->input('commission_agent')) ?
$request->input('commission_agent') : null;
                if ($commsn_agnt_setting == 'logged_in_user') {
                    $input['commission_agent'] = $user_id;
                }

                if (isset($input['exchange_rate']) && $this->transactionUtil-
>num_uf($input['exchange_rate']) == 0) {
                    $input['exchange_rate'] = 1;
                }

                //Customer group details
                $contact_id = $request->get('contact_id', null);
                $cg = $this->contactUtil->getCustomerGroup($business_id, $contact_id);
                $input['customer_group_id'] = (empty($cg) || empty($cg->id)) ? null : $cg->id;

                //set selling price group id
                if ($request->has('price_group')) {
                    $input['selling_price_group_id'] = $request->input('price_group');
                }

                $input['is_suspend'] = isset($input['is_suspend']) && 1 == $input['is_suspend']
? 1 : 0;
                if ($input['is_suspend']) {
                    $input['sale_note'] = !empty($input['additional_notes']) ?
$input['additional_notes'] : null;
                }

                if ($is_direct_sale && $status_before == 'draft') {
                    $input['invoice_scheme_id'] = $request->input('invoice_scheme_id');
                }

                //Begin transaction
                DB::beginTransaction();

                $transaction = $this->transactionUtil->updateSellTransaction($id, $business_id,
$input, $invoice_total, $user_id);
```

```
                //Update Sell lines
                $deleted_lines = $this->transactionUtil->createOrUpdateSellLines($transaction,
$input['products'], $input['location_id'], true, $status_before);

                //Update update lines
                if (!$is_direct_sale && !$transaction->is_suspend) {
                    //Add change return
                    $change_return = $this->dummyPaymentLine;
                    $change_return['amount'] = $input['change_return'];
                    $change_return['is_return'] = 1;
                    if (!empty($input['change_return_id'])) {
                        $change_return['id'] = $input['change_return_id'];
                    }
                    $input['payment'][] = $change_return;

                    $this->transactionUtil->createOrUpdatePaymentLines($transaction,
$input['payment']);

                    //Update cash register
                    $this->cashRegisterUtil->updateSellPayments($status_before, $transaction,
$input['payment']);
                }

                if (Auth::user()->business->enable_rp == 1) {
                    $this->transactionUtil->updateCustomerRewardPoints($contact_id, $transaction-
>rp_earned, $rp_earned_before, $transaction->rp_redeemed, $rp_redeemed_before);
                }

                //Update payment status
                $this->transactionUtil->updatePaymentStatus($transaction->id, $transaction-
>final_total);

                //Update product stock
                $this->productUtil->adjustProductStockForInvoice($status_before, $transaction,
$input);

                //Allocate the quantity from purchase and add mapping of
                //purchase & sell lines in
                //transaction_sell_lines_purchase_lines table
                $business_details = $this->businessUtil->getDetails($business_id);
                $pos_settings = empty($business_details->pos_settings) ? $this->businessUtil-
>defaultPosSettings() : json_decode($business_details->pos_settings, true);

                $business = ['id' => $business_id,
                                'accounting_method' => Auth::user()->business->accounting_method,
                                'location_id' => $input['location_id'],
                                'pos_settings' => $pos_settings
                            ];
                $this->transactionUtil->adjustMappingPurchaseSell($status_before, $transaction,
$business, $deleted_lines);

                if ($this->transactionUtil->isModuleEnabled('tables')) {
                    $transaction->res_table_id = request()->get('res_table_id');
                    $transaction->save();
                }
                if ($this->transactionUtil->isModuleEnabled('service_staff')) {
                    $transaction->res_waiter_id = request()->get('res_waiter_id');
                    $transaction->save();
                }
                $log_properties = [];
                if (isset($input['repair_completed_on'])) {
                    $completed_on = !empty($input['repair_completed_on']) ? $this-
>transactionUtil->uf_date($input['repair_completed_on'], true) : null;
                    if ($transaction->repair_completed_on != $completed_on) {
                        $log_properties['completed_on_from'] = $transaction->repair_completed_on;
                        $log_properties['completed_on_to'] = $completed_on;
                    }
                }

                //Set Module fields
                if (!empty($input['has_module_data'])) {
                    $this->moduleUtil->getModuleData('after_sale_saved', ['transaction' =>
$transaction, 'input' => $input]);
                }

                if (!empty($input['update_note'])) {
                    $log_properties['update_note'] = $input['update_note'];
                }

                Media::uploadMedia($business_id, $transaction, $request, 'documents');
```

```
                $neraca = $this->transactionUtil->getBalanceSheet();
                $transaction->neraca = $neraca;

                DB::commit();

                activity('transaction')
                    ->causedBy(Auth::user())
                    ->performedOn($transaction)
                    ->withProperties($transaction)
                    ->log('updated');

                $msg = '';
                $receipt = '';

                if ($input['status'] == 'draft' && $input['is_quotation'] == 0) {
                    $msg = "Draft added";
                } elseif ($input['status'] == 'draft' && $input['is_quotation'] == 1) {
                    $msg = "Quotation updated";
                    if (!$is_direct_sale) {
                        $receipt = $this->receiptContent($business_id, $input['location_id'],
$transaction->id);
                    } else {
                        $receipt = '';
                    }
                } elseif ($input['status'] == 'final') {
                    $msg = "POS Sale Updated";
                    if (!$is_direct_sale && !$transaction->is_suspend) {
                        $receipt = $this->receiptContent($business_id, $input['location_id'],
$transaction->id);
                    } else {
                        $receipt = '';
                    }
                }

                return $this->respondSuccess(['receipt' => $receipt], $msg);
            } else {
                return $this->respondFailed();
            }
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed($e->getMessage());
        }
    }

    public function destroy($id)
    {
        if (!Auth::user()->can('sell.delete')) {
            return $this->respondFailed('Unauthorized action.', 403);
        }

        try {
            //Check if return exist then not allowed
            if ($this->transactionUtil->isReturnExist($id)) {
                return $this->respondFailed("Return exist", 403);
            }

            $business_id = Auth::user()->business_id;
            $transaction = Transaction::where('business_id', $business_id)
                        ->where('type', 'sell')
                        ->with(['sell_lines'])
                        ->findOrFail($id);

            //Begin transaction
            DB::beginTransaction();

            if (!empty($transaction)) {
                //If status is draft direct delete transaction
                if ($transaction->status == 'draft') {
                    $transaction->delete();
                } else {
                    $deleted_sell_lines = $transaction->sell_lines;
                    $deleted_sell_lines_ids = $deleted_sell_lines->pluck('id')->toArray();
                    $this->transactionUtil->deleteSellLines(
                        $deleted_sell_lines_ids,
                        $transaction->location_id
                    );
```

```
                    $this->transactionUtil->updateCustomerRewardPoints($transaction->contact_id,
0, $transaction->rp_earned, 0, $transaction->rp_redeemed);

                    $transaction->status = 'draft';
                    $business = ['id' => $business_id,
                            'accounting_method' => Auth::user()->business->accounting_method,
                            'location_id' => $transaction->location_id
                        ];

                    $this->transactionUtil->adjustMappingPurchaseSell('final', $transaction,
$business, $deleted_sell_lines_ids);

                    //Delete Cash register transactions
                    $transaction->cash_register_payments()->delete();

                    $transaction->delete();
                }
            }

            //Delete account transactions
            AccountTransaction::where('transaction_id', $transaction->id)->delete();

            $neraca = $this->transactionUtil->getBalanceSheet();
            $transaction->neraca = $neraca;

            DB::commit();

            activity('transaction')
                ->causedBy(Auth::user())
                ->performedOn($transaction)
                ->withProperties($transaction)
                ->log('deleted');
            return $this->respondSuccess(null, "Sales Deleted successfully", 204);
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return $this->respondFailed($e->getMessage());
        }
    }

    public function getProductRow($variation_id, $location_id)
    {
        $output = [];

        try {
            $row_count = request()->get('product_row');
            $row_count = $row_count + 1;
            $is_direct_sell = false;
            if (request()->get('is_direct_sell') == 'true') {
                $is_direct_sell = true;
            }

            $business_id = request()->session()->get('user.business_id');

            $business_details = $this->businessUtil->getDetails($business_id);
            $pos_settings = empty($business_details->pos_settings) ? $this->businessUtil-
>defaultPosSettings() : json_decode($business_details->pos_settings, true);

            $check_qty = !empty($pos_settings['allow_overselling']) ? false : true;
            $product = $this->productUtil->getDetailsFromVariation($variation_id, $business_id,
$location_id, $check_qty);
            $product->formatted_qty_available = $this->productUtil->num_f($product-
>qty_available, false, null, true);

            $sub_units = $this->productUtil->getSubUnits($business_id, $product->unit_id, false,
$product->product_id);

            //Get customer group and change the price accordingly
            $customer_id = request()->get('customer_id', null);
            $cg = $this->contactUtil->getCustomerGroup($business_id, $customer_id);
            $percent = (empty($cg) || empty($cg->amount)) ? 0 : $cg->amount;
            $product->default_sell_price = $product->default_sell_price + ($percent * $product-
>default_sell_price / 100);
            $product->sell_price_inc_tax = $product->sell_price_inc_tax + ($percent * $product-
>sell_price_inc_tax / 100);

            $tax_dropdown = TaxRate::forBusinessDropdown($business_id, true, true);

            $enabled_modules = $this->transactionUtil->allModulesEnabled();
```

```
            //Get lot number dropdown if enabled
            $lot_numbers = [];
            if (request()->session()->get('business.enable_lot_number') == 1 || request()-
>session()->get('business.enable_product_expiry') == 1) {
                $lot_number_obj = $this->transactionUtil-
>getLotNumbersFromVariation($variation_id, $business_id, $location_id, true);
                foreach ($lot_number_obj as $lot_number) {
                    $lot_number->qty_formated = $this->productUtil->num_f($lot_number-
>qty_available);
                    $lot_numbers[] = $lot_number;
                }
            }
            $product->lot_numbers = $lot_numbers;

            $price_group = request()->input('price_group');
            if (!empty($price_group)) {
                $variation_group_prices = $this->productUtil-
>getVariationGroupPrice($variation_id, $price_group, $product->tax_id);

                if (!empty($variation_group_prices['price_inc_tax'])) {
                    $product->sell_price_inc_tax = $variation_group_prices['price_inc_tax'];
                    $product->default_sell_price = $variation_group_prices['price_exc_tax'];
                }
            }

            $output['success'] = true;

            $waiters = null;
            if ($this->productUtil->isModuleEnabled('service_staff') &&
!empty($pos_settings['inline_service_staff'])) {
                $waiters_enabled = true;
                $waiters = $this->productUtil->serviceStaffDropdown($business_id, $location_id);
            }

            if (request()->get('type') == 'sell-return') {
                $output['html_content'] =  view('sell_return.partials.product_row')
                            ->with(compact('product', 'row_count', 'tax_dropdown',
'enabled_modules', 'sub_units'))
                            ->render();
            } else {
                $is_cg = !empty($cg->id) ? true : false;
                $is_pg = !empty($price_group) ? true : false;
                $discount = $this->productUtil->getProductDiscount($product, $business_id,
$location_id, $is_cg, $is_pg);

                if ($is_direct_sell) {
                    $edit_discount = Auth::user()->can('edit_product_discount_from_sale_screen');
                    $edit_price = Auth::user()->can('edit_product_price_from_sale_screen');
                } else {
                    $edit_discount = Auth::user()->can('edit_product_discount_from_pos_screen');
                    $edit_price = Auth::user()->can('edit_product_price_from_pos_screen');
                }

                $output['html_content'] =  view('sale_pos.product_row')
                            ->with(compact('product', 'row_count', 'tax_dropdown',
'enabled_modules', 'pos_settings', 'sub_units', 'discount', 'waiters', 'edit_discount',
'edit_price'))
                            ->render();
            }

            $output['enable_sr_no'] = $product->enable_sr_no;

            if ($this->transactionUtil->isModuleEnabled('modifiers')  && !$is_direct_sell) {
                $this_product = Product::where('business_id', $business_id)
                                        ->find($product->product_id);
                if (count($this_product->modifier_sets) > 0) {
                    $product_ms = $this_product->modifier_sets;
                    $output['html_modifier'] =
view('restaurant.product_modifier_set.modifier_for_product')
                        ->with(compact('product_ms', 'row_count'))->render();
                }
            }
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            $output['success'] = false;
            $output['msg'] = __('lang_v1.item_out_of_stock');
        }
```

```
        return $output;
    }

    public function getPaymentRow(Request $request)
    {
        $business_id = request()->session()->get('user.business_id');

        $row_index = $request->input('row_index');
        $removable = true;
        $payment_types = $this->productUtil->payment_types();

        $payment_line = $this->dummyPaymentLine;

        //Accounts
        $accounts = [];
        if ($this->moduleUtil->isModuleEnabled('account')) {
            $accounts = Account::forDropdown($business_id, true, false);
        }

        return view('sale_pos.partials.payment_row')
            ->with(compact('payment_types', 'row_index', 'removable', 'payment_line',
'accounts'));
    }

    public function getRecentTransactions(Request $request)
    {
        $business_id = Auth::user()->business_id;
        $user_id = Auth::id();
        $transaction_status = $request->get('status');

        $register = $this->cashRegisterUtil->getCurrentCashRegister($user_id);

        $query = Transaction::where('transactions.business_id', $business_id)
                        ->where('transactions.type', 'sell')
                        ->where('transactions.is_direct_sale', 0)
                        ->leftJoin(
                            'transactions AS SR',
                            'transactions.id',
                            '=',
                            'SR.return_parent_id'
                        );

        if ($transaction_status == 'quotation') {
            $query->where('transactions.status', 'draft')
                ->where('is_quotation', 1);
        } elseif ($transaction_status == 'draft') {
            $query->where('transactions.status', 'draft')
                ->where('is_quotation', 0);
        } else {
            $query->where('transactions.status', $transaction_status);
        }

        $location_id = '';
        if($request->input('location_id')) {
            $location_id = $request->input('location_id');
            if($location_id != -1) {
                $query->where('transactions.location_id', $location_id);
            } else {
                $location_id = '';
            }
        }


        if($request->input('type') == "digital") {
            $query->whereNotNull('transactions.digital_status');
        } else {
            $query->whereNull('transactions.digital_status');
        }


        $transactions = $query->orderBy('transactions.created_at', 'desc')
                            ->groupBy('transactions.id')
                            ->select([
                                'transactions.id',
                                'transactions.invoice_no',
                                'transactions.transaction_date',
                                'transactions.final_total',
                                'transactions.payment_status',
                                'transactions.digital_status',
                                'transactions.additional_notes',
                                'transactions.discount_amount',
```

```
                                'transactions.contact_id', //unset
                                'transactions.location_id', //unset
                                'transactions.created_by', //unset
                                DB::raw('IF((COUNT(SR.id) > 0), true, false) as is_return')
                            ])->with(['contact', 'sell_lines' => function($q) use ($location_id)
{
                                $q->leftJoin('products', 'transaction_sell_lines.product_id',
'=', 'products.id');
                                $q->leftJoin('variations', 'transaction_sell_lines.variation_id',
'=', 'variations.id');
                                $q->leftJoin('brands', 'products.brand_id', '=', 'brands.id');
                                // $q->leftJoin('variation_location_details', 'variations.id',
'=', 'variation_location_details.variation_id');
                                // $q->where('variation_location_details.location_id', '=',
'transactions.location_id');
                                $q->select([
                                    'products.id as product_id',
                                    'products.name as name',
                                    'products.type as type',
                                    'products.image as image',
                                    'products.enable_stock as enable_stock',
                                    'products.product_description as product_description',
                                    'brands.name as brand',
                                    'variations.id as id',
                                    'variations.name as variation',
                                    DB::raw('CONVERT(IFNULL(transaction_sell_lines.quantity, 0),
INTEGER) as quantity'),
                                    // DB::raw('CONVERT(IFNULL(
variation_location_details.qty_available, 0), INTEGER) + quantity as qty_available'),
                                    'transaction_sell_lines.unit_price_inc_tax as selling_price',
                                    'variations.sub_sku as sub_sku',
                                    'transaction_sell_lines.id as sell_line_id',
                                    'transaction_id'
                                ]);
                            },'payment_lines']);

        $start = 0;
        $limit = $request->input('limit') ? $request->input('limit') : 10000;
        $paged = $request->input('page') ? $request->input('page') : 1;

        $start = ($paged - 1) * $limit;
        $transactions = $transactions->offset($start)->limit($limit)->get();

        // dd($transactions);

        foreach($transactions as $key => $transaction) {
            $cg = $this->contactUtil->getCustomerGroup($business_id, $transaction->contact_id);
            $percent = (empty($cg) || empty($cg->amount)) ? 0 : $cg->amount;

            $products = [];
            $transactions[$key]->customer_group_id = $transaction->customer_group_id ?? 0;
            $transactions[$key]->total_paid = 0;
            $transactions[$key]->spg_name = $transaction->contact->price_group != null ?
$transaction->contact->price_group->name : "-";
            $transactions[$key]->outlet_name = $transaction->location->name;
            $transactions[$key]->sales_person_name = $transaction->sales_person->first_name . " "
. $transaction->sales_person->last_name;
            // $transactions[$key]->contact->customer_group_id = $transaction->contact-
>customer_group_id ?? 0;
            // $transactions[$key]->contact->selling_price_group_id = $transaction->contact-
>selling_price_group_id ?? 0;
            foreach($transaction->sell_lines as $sell_line){
                $sell_line->discount_group = $percent;
                $qtyAvail = VariationLocationDetails::where('variation_id', $sell_line->id)-
>where('location_id', $transaction->location_id)->first()['qty_available'];
                $sell_line->qty_available = $qtyAvail == "" ? "0" : $qtyAvail;
                $products[] = $sell_line;
            }

            foreach($transaction->payment_lines as $payment_line){
                if($payment_line->card_type == "credit")
                    $transactions[$key]->total_paid += $payment_line->amount;
            }

            $transactions[$key]['products'] = $products;
            unset($transactions[$key]->sell_lines);
            unset($transactions[$key]->payment_lines);
            unset($transactions[$key]->sales_person);
            unset($transactions[$key]->location);
            unset($transactions[$key]->contact);
        }
```

```
        return $this->respondArray($transactions);
    }

    public function printInvoice(Request $request, $transaction_id)
    {
        if (request()->ajax()) {
            try {
                $output = ['success' => 0,
                        'msg' => trans("messages.something_went_wrong")
                        ];

                $business_id = $request->session()->get('user.business_id');

                $transaction = Transaction::where('business_id', $business_id)
                                ->where('id', $transaction_id)
                                ->with(['location'])
                                ->first();

                if (empty($transaction)) {
                    return $output;
                }

                $printer_type = 'browser';
                if (!empty(request()->input('check_location')) && request()->
>input('check_location') == true) {
                    $printer_type = $transaction->location->receipt_printer_type;
                }

                $is_package_slip = !empty($request->input('package_slip')) ? true : false;

                $receipt = $this->receiptContent($business_id, $transaction->location_id,
$transaction_id, $printer_type, $is_package_slip, false);

                if (!empty($receipt)) {
                    $output = ['success' => 1, 'receipt' => $receipt];
                }
            } catch (\Exception $e) {
                \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" .
$e->getMessage());

                $output = ['success' => 0,
                        'msg' => trans("messages.something_went_wrong")
                        ];
            }

            return $output;
        }
    }

    public function getProductSuggestion(Request $request)
    {
        $location_id = $request->input('location_id') ? $request->input('location_id') : "";
        $term = $request->get('name');
        $check_qty = true;
        $business_id = Auth::user()->business_id;
        DB::enableQueryLog();
        $products = Variation::join('products as p', 'variations.product_id', '=', 'p.id')
            ->leftjoin(
                'variation_location_details AS VLD',
                function ($join) use ($location_id) {
                    $join->on('variations.id', '=', 'VLD.variation_id');

                    //Include Location
                    if (!empty($location_id)) {
                        $join->where(function ($query) use ($location_id) {
                            $query->where('VLD.location_id', '=', $location_id);
                            //Check null to show products even if no quantity is available in a
location.
                            //TODO: Maybe add a settings to show product not available at a
location or not.
                            $query->orWhereNull('VLD.location_id');
                        });
                        ;
                    }
                }
            )
            ->leftJoin('brands', 'p.brand_id', '=', 'brands.id')
            ->where('variations.is_active', 1)
            ->where('p.business_id', $business_id)
            ->where('p.type', '!=', 'modifier')
```

```
            ->where('p.is_inactive', 0)
            ->where('p.not_for_selling', 0);
            //Include search
        if (!empty($term)) {
            $products->where(function ($query) use ($term) {
                $query->where('p.name', 'like', '%' . $term .'%');
                $query->orWhere('sku', 'like', '%' . $term .'%');
                $query->orWhere('sub_sku', 'like', '%' . $term .'%');
            });
        }

        $brand_name = $request->input('brand', null);
        if($brand_name) {
            $brand = Brands::where('business_id', $business_id)
                        ->where('name', 'like', '%' . $brand_name . '%')
                        ->first();
            if($brand)
                $products->where('p.brand_id', $brand->id);
            else
                $products->where('p.brand_id', -1);
        }

        $category_name = $request->input('category', null);
        if($category_name) {
            $category = Category::where('business_id', $business_id)
                        ->where('name', 'like', '%' . $category_name . '%')
                        ->first();
            if($category)
                $products->where('p.sub_category_id', $category->id);
        }

        $category_id = $request->input('category_id', null);
        if ($category_id) {
            $products->where('p.category_id', $category_id);
            $sub_category_id = $request->input('sub_category_id', null);
            if ($sub_category_id) {
                $products->where('p.sub_category_id', $sub_category_id);
            }
        }

        $product_type = $request->input('product_type', null);
        if ($product_type == 'digital') {
            $products->where('p.product_type', $product_type);
        } else {
            $products->where('p.product_type', '!=', 'digital');
        }

        $digital_type = $request->input('digital_type', null);
        if ($digital_type) {
            $products->where('p.digital_type', $digital_type);
        }

        if($request->input('selected_products')) {
            $selected_products = $request->input('selected_products');
            $products->where(function($q) use ($selected_products) {
                $q->orWhereIn('variations.id', explode(',', $selected_products));
            });
        }

        $products = $products->select(
            'p.id as product_id',
            'p.name',
            'p.type',
            'p.image',
            'p.enable_stock',
            'p.product_description',
            'brands.name as brand',
            'variations.id',
            'variations.name as variation',
            DB::raw('CONVERT(IFNULL( VLD.qty_available, 0), INTEGER) as qty_available'),
            'variations.default_sell_price as selling_price',
            'variations.sub_sku'
        )->orderBy('selling_price', 'asc');

        $customer_id = $request->get('customer_id', null);
        $customer = Contact::find($customer_id);
        $cg = $this->contactUtil->getCustomerGroup($business_id, $customer_id);
        $percent = (empty($cg) || empty($cg->amount)) ? 0 : $cg->amount;

        if($customer != null)
            $price_group = $customer->selling_price_group_id;
```

```
        $start = 0;
        $limit = $request->input('limit') ? $request->input('limit') : 25;
        $paged = $request->input('page') ? $request->input('page') : 1;

        $start = ($paged - 1) * $limit;

        $products = $products->offset($start)->limit($limit)->get();

        foreach($products as $key => $product) {
            $products[$key]->qty_available = ($product->qty_available == null) ? 0 : $product-
>qty_available;
            $products[$key]->image = ($product->image == null) ? '' : env('AWS_CDN_URL') .
'/img/' . $product->image;
            $products[$key]->discount_group = $percent;
            if (!empty($price_group)) {
                $variation_group_prices = $this->productUtil->getVariationGroupPrice($product-
>id, $price_group, null);
                $spgPrice = $variation_group_prices['price_inc_tax'];
                if (!empty($spgPrice) && $spgPrice > 0) {
                        $products[$key]->selling_price_group =
$variation_group_prices['price_inc_tax'];
                } else {
                    $products[$key]->selling_price_group = $products[$key]->selling_price;
                }

                $products[$key]->selling_price_final = ($products[$key]->selling_price_group ??
0) - ($percent * $products[$key]->selling_price_group / 100);
            } else {
                $products[$key]->selling_price_group = $products[$key]->selling_price;
                $products[$key]->selling_price_final = $products[$key]->selling_price - ($percent
* $products[$key]->selling_price / 100);
                    // $products[$key]->selling_price_final = 0;
            }
        }

        return $this->respondArray($products);
    }

    public function showInvoiceUrl($id)
    {
        if (!Auth::user()->can('sell.update')) {
            abort(403, 'Unauthorized action.');
        }

        if (request()->ajax()) {
            $business_id = request()->session()->get('user.business_id');
            $transaction = Transaction::where('business_id', $business_id)
                                ->findorfail($id);
            $url = $this->transactionUtil->getInvoiceUrl($id, $business_id);

            return view('sale_pos.partials.invoice_url_modal')
                    ->with(compact('transaction', 'url'));
        }
    }

    public function showInvoice($token)
    {
        $transaction = Transaction::where('invoice_token', $token)->with(['business'])->first();

        if (!empty($transaction)) {
            $receipt = $this->receiptContent($transaction->business_id, $transaction-
>location_id, $transaction->id, 'browser');

            $title = $transaction->business->name . ' | ' . $transaction->invoice_no;
            return view('sale_pos.partials.show_invoice')
                    ->with(compact('receipt', 'title'));
        } else {
            die(__("messages.something_went_wrong"));
        }
    }

    public function listSubscriptions()
    {
        if (!Auth::user()->can('sell.view') && !Auth::user()->can('direct_sell.access')) {
            abort(403, 'Unauthorized action.');
        }

        if (request()->ajax()) {
```

```
            $business_id = request()->session()->get('user.business_id');

            $sells = Transaction::leftJoin('contacts', 'transactions.contact_id', '=',
'contacts.id')
                ->leftJoin('transaction_payments as tp', 'transactions.id', '=',
'tp.transaction_id')
                ->join(
                    'business_locations AS bl',
                    'transactions.location_id',
                    '=',
                    'bl.id'
                )
                ->where('transactions.business_id', $business_id)
                ->where('transactions.type', 'sell')
                ->where('transactions.status', 'final')
                ->where('transactions.is_recurring', 1)
                ->select(
                    'transactions.id',
                    'transactions.transaction_date',
                    'transactions.is_direct_sale',
                    'transactions.invoice_no',
                    'contacts.name',
                    'transactions.subscription_no',
                    'bl.name as business_location',
                    'transactions.recur_parent_id',
                    'transactions.recur_stopped_on',
                    'transactions.is_recurring',
                    'transactions.recur_interval',
                    'transactions.recur_interval_type',
                    'transactions.recur_repetitions'
                )->with(['subscription_invoices']);


            $permitted_locations = Auth::user()->permitted_locations();
            if ($permitted_locations != 'all') {
                $sells->whereIn('transactions.location_id', $permitted_locations);
            }

            if (!empty(request()->start_date) && !empty(request()->end_date)) {
                $start = request()->start_date;
                $end =  request()->end_date;
                $sells->whereDate('transactions.transaction_date', '>=', $start)
                        ->whereDate('transactions.transaction_date', '<=', $end);
            }
            $datatable = Datatables::of($sells)
                ->addColumn(
                    'action',
                    function ($row) {
                        $html = '' ;

                        if ($row->is_recurring == 1 && Auth::user()->can("sell.update")) {
                            $link_text = !empty($row->recur_stopped_on) ?
__('lang_v1.start_subscription') : __('lang_v1.stop_subscription');
                            $link_class = !empty($row->recur_stopped_on) ? 'btn-success' : 'btn-
danger';

                            $html .= '<a href="' .
action('SellPosController@toggleRecurringInvoices', [$row->id]) . '"
class="toggle_recurring_invoice btn btn-xs ' . $link_class . '"><i class="fa fa-power-off"></i> '
. $link_text . '</a>';

                            if ($row->is_direct_sale == 0) {
                                $html .= '<a target="_blank" class="btn btn-xs btn-primary"
href="' . action('SellPosController@edit', [$row->id]) . '"><i class="glyphicon glyphicon-
edit"></i> ' . __("messages.edit") . '</a>';
                            } else {
                                $html .= '<a target="_blank" class="btn btn-xs btn-primary"
href="' . action('SellController@edit', [$row->id]) . '"><i class="glyphicon glyphicon-edit"></i>
' . __("messages.edit") . '</a>';
                            }
                        }

                        return $html;
                    }
                )
                ->removeColumn('id')
                ->editColumn('transaction_date', '{{@format_date($transaction_date)}}')
                ->editColumn('recur_interval', function ($row) {
                    $type = $row->recur_interval == 1 ? str_singular(__('lang_v1.' . $row-
>recur_interval_type)) : __('lang_v1.' . $row->recur_interval_type);
```

```
                    return $row->recur_interval . $type;
                })
                ->addColumn('subscription_invoices', function ($row) {
                    $invoices = [];
                    if (!empty($row->subscription_invoices)) {
                        $invoices = $row->subscription_invoices->pluck('invoice_no')->toArray();
                    }

                    $html = '';
                    $count = 0;
                    if (!empty($invoices)) {
                        $imploded_invoices = '<span class="label bg-info">' . implode('</span>,
<span class="label bg-info">', $invoices) . '</span>';
                        $count = count($invoices);
                        $html .= '<small>' . $imploded_invoices . '</small>';
                    }
                    if ($count > 0) {
                        $html .= '<br><small class="text-muted">' .
                    __('sale.total') . ': ' . $count . '</small>';
                    }

                    return $html;
                })
                ->addColumn('last_generated', function ($row) {
                    if (!empty($row->subscription_invoices)) {
                        $last_generated_date = $row->subscription_invoices->max('created_at');
                    }
                    return !empty($last_generated_date) ? $last_generated_date->diffForHumans() :
'';
                })
                ->addColumn('upcoming_invoice', function ($row) {
                    if (empty($row->recur_stopped_on)) {
                        $last_generated = !empty($row->subscription_invoices) ?
Carbon::parse($row->subscription_invoices->max('transaction_date')) : Carbon::parse($row-
>transaction_date);
                        if ($row->recur_interval_type == 'days') {
                            $upcoming_invoice = $last_generated->addDays($row->recur_interval);
                        } elseif ($row->recur_interval_type == 'months') {
                            $upcoming_invoice = $last_generated->addMonths($row->recur_interval);
                        } elseif ($row->recur_interval_type == 'years') {
                            $upcoming_invoice = $last_generated->addYears($row->recur_interval);
                        }
                    }
                    return !empty($upcoming_invoice) ? $this->transactionUtil-
>format_date($upcoming_invoice) : '';
                })
                ->rawColumns(['action', 'subscription_invoices'])
                ->make(true);

            return $datatable;
        }
        return view('sale_pos.subscriptions');
    }

    public function toggleRecurringInvoices($id)
    {
        if (!Auth::user()->can('sell.create')) {
            abort(403, 'Unauthorized action.');
        }

        try {
            $business_id = request()->session()->get('user.business_id');
            $transaction = Transaction::where('business_id', $business_id)
                            ->where('type', 'sell')
                            ->where('is_recurring', 1)
                            ->findorfail($id);

            if (empty($transaction->recur_stopped_on)) {
                $transaction->recur_stopped_on = Carbon::now();
            } else {
                $transaction->recur_stopped_on = null;
            }
            $transaction->save();

            $output = ['success' => 1,
                    'msg' => trans("lang_v1.updated_success")
                ];
        } catch (\Exception $e) {
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());
```

```
            $output = ['success' => 0,
                            'msg' => trans("messages.something_went_wrong")
                        ];
        }

        return $output;
    }

    public function getRewardDetails(Request $request)
    {
        if ($request->session()->get('business.enable_rp') != 1) {
            return '';
        }

        $business_id = request()->session()->get('user.business_id');

        $customer_id = $request->input('customer_id');

        $redeem_details = $this->transactionUtil->getRewardRedeemDetails($business_id,
$customer_id);

        return json_encode($redeem_details);
    }

    public function placeOrdersApi(Request $request)
    {
        try {
            $api_token = $request->header('API-TOKEN');
            $api_settings = $this->moduleUtil->getApiSettings($api_token);

            $business_id = $api_settings->business_id;
            $location_id = $api_settings->location_id;

            $input = $request->only(['products', 'customer_id', 'addresses']);

            //check if all stocks are available
            $variation_ids = [];
            foreach ($input['products'] as $product_data) {
                $variation_ids[] = $product_data['variation_id'];
            }

            $variations_details = $this->getVariationsDetails($business_id, $location_id,
$variation_ids);
            $is_valid = true;
            $error_messages = [];
            $sell_lines = [];
            $final_total = 0;
            foreach ($variations_details as $variation_details) {
                if ($variation_details->product->enable_stock == 1) {
                    if (empty($variation_details->variation_location_details[0]) ||
$variation_details->variation_location_details[0]->qty_available <
$input['products'][$variation_details->id]['quantity']) {
                        $is_valid = false;
                        $error_messages[] = 'Only ' . $variation_details-
>variation_location_details[0]->qty_available . ' ' . $variation_details->product->unit-
>short_name . ' of '. $input['products'][$variation_details->id]['product_name'] . ' available';
                    }
                }

                //Create product line array
                $sell_lines[] = [
                    'product_id' => $variation_details->product->id,
                    'unit_price_before_discount' => $variation_details->unit_price_inc_tax,
                    'unit_price' => $variation_details->unit_price_inc_tax,
                    'unit_price_inc_tax' => $variation_details->unit_price_inc_tax,
                    'variation_id' => $variation_details->id,
                    'quantity' => $input['products'][$variation_details->id]['quantity'],
                    'item_tax' => 0,
                    'enable_stock' => $variation_details->product->enable_stock,
                    'tax_id' => null,
                ];

                $final_total += ($input['products'][$variation_details->id]['quantity'] *
$variation_details->unit_price_inc_tax);
            }

            if (!$is_valid) {
                return $this->respond([
                    'success' => false,
                    'error_messages' => $error_messages
                ]);
```

```
            }

            $business = Business::find($business_id);
            $user_id = $business->owner_id;

            $business_data = [
                'id' => $business_id,
                'accounting_method' => $business->accounting_method,
                'location_id' => $location_id
            ];

            $customer = Contact::where('business_id', $business_id)
                            ->whereIn('type', ['customer', 'both'])
                            ->find($input['customer_id']);

            $order_data = [
                'business_id' => $business_id,
                'location_id' => $location_id,
                'contact_id' => $input['customer_id'],
                'final_total' => $final_total,
                'created_by' => $user_id,
                'status' => 'final',
                'payment_status' => 'due',
                'additional_notes' => '',
                'transaction_date' => Carbon::now(),
                'customer_group_id' => $customer->customer_group_id,
                'tax_rate_id' => null,
                'sale_note' => null,
                'commission_agent' => null,
                'order_addresses' => json_encode($input['addresses']),
                'products' => $sell_lines,
                'is_created_from_api' => 1,
                'discount_type' => 'fixed',
                'discount_amount' => 0
            ];

            $invoice_total = [
                'total_before_tax' => $final_total,
                'tax' => 0,
            ];

            DB::beginTransaction();

            $transaction = $this->transactionUtil->createSellTransaction($business_id,
$order_data, $invoice_total, $user_id, false);

            //Create sell lines
            $this->transactionUtil->createOrUpdateSellLines($transaction,
$order_data['products'], $order_data['location_id'], false, null, [], false);

            //update product stock
            foreach ($order_data['products'] as $product) {
                if ($product['enable_stock']) {
                    $this->productUtil->decreaseProductQuantity(
                        $product['product_id'],
                        $product['variation_id'],
                        $order_data['location_id'],
                        $product['quantity']
                    );
                }
            }

            $this->transactionUtil->mapPurchaseSell($business_data, $transaction->sell_lines,
'purchase');
            //Auto send notification
            $this->notificationUtil->autoSendNotification($business_id, 'new_sale', $transaction,
$transaction->contact);

            DB::commit();

            $receipt = $this->receiptContent($business_id, $transaction->location_id,
$transaction->id);

            $output = [
                'success' => 1,
                'transaction' => $transaction,
                'receipt' => $receipt
            ];
        } catch (\Exception $e) {
            DB::rollBack();
```

```
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());
            $msg = trans("messages.something_went_wrong");

            if (get_class($e) == \App\Exceptions\PurchaseSellMismatch::class) {
                $msg = $e->getMessage();
            }

            $output = ['success' => 0,
                    'error_messages' => [$msg]
                ];
        }

        return $this->respond($output);
    }

    private function getVariationsDetails($business_id, $location_id, $variation_ids)
    {
        $variation_details = Variation::whereIn('id', $variation_ids)
                            ->with([
                                'product' => function ($q) use ($business_id) {
                                    $q->where('business_id', $business_id);
                                },
                                'product.unit',
                                'variation_location_details' => function ($q) use ($location_id)
{
                                    $q->where('location_id', $location_id);
                                }
                            ])->get();

        return $variation_details;
    }

    private function getPurchaseLine($variation_id, $location_id) {
        $business_id = Auth::user()->business_id;

        $product = Variation::join('products as p', 'variations.product_id', '=', 'p.id')
            ->leftjoin(
                'variation_location_details AS VLD',
                function ($join) use ($location_id) {
                    $join->on('variations.id', '=', 'VLD.variation_id');

                    //Include Location
                    if (!empty($location_id)) {
                        $join->where(function ($query) use ($location_id) {
                            $query->where('VLD.location_id', '=', $location_id);
                            //Check null to show products even if no quantity is available in a
location.
                            //TODO: Maybe add a settings to show product not available at a
location or not.
                            $query->orWhereNull('VLD.location_id');
                        });
                        ;
                    }
                }
            )
            ->where('p.business_id', $business_id)
            ->where('p.type', '!=', 'modifier')
            ->where('p.is_inactive', 0)
            ->where('p.not_for_selling', 0)
            ->where('variations.id', $variation_id);

        $product = $product->select(
            'p.id as product_id',
            'p.name',
            'p.type',
            'p.enable_stock',
            'p.unit_id as product_unit_id',
            'p.sub_unit_ids as sub_unit_id',
            'variations.id as variation_id',
            'variations.name as variation',
            DB::raw('CONVERT(IFNULL( VLD.qty_available, 0), UNSIGNED INTEGER) as qty_available'),
            'variations.default_sell_price as unit_price',
            'variations.sell_price_inc_tax as unit_price_inc_tax',
            'variations.sub_sku'
        )->first();

        return $product;
    }

    public function getAllCustomer() {
```

```
        $business_id = Auth::user()->business_id;
        $contacts = Contact::where('type', 'customer')
                    ->where('business_id', $business_id)
                    ->with(['customer_group' => function($q) {
                        $q->select(['id', 'name', 'amount', 'is_default', 'is_digital']);
                    }])
                    ->get();

        foreach($contacts as $key => $value) {
            $contacts[$key]->customer_group_id = $value->customer_group_id ? $value->customer_group_id : 0;
            $contacts[$key]->selling_price_group_id = $value->selling_price_group_id ? $value->selling_price_group_id : 0;
        }
        return $this->respondArray($contacts);
    }

    public function getCategories() {
        $business_id = Auth::user()->business_id;

            $default = [
                [
                    'name' => 'Produk Fisik',
                    'short_code' => 'ALL_PRODUCTS',
                    'id' => -1,
                    'parent_id' => 0,
                ],
                [
                    'name' => 'Produk Digital',
                    'short_code' => 'PRODUCT_DIGITAL',
                    'id' => -2,
                    'parent_id' => 0,
                ]
            ];

        $categories = Category::where('business_id', $business_id)
            ->where('is_digital', 0)
            ->where('parent_id', '')
            ->select([
                'name',
                'short_code',
                'id',
                DB::raw('CONVERT(IFNULL( parent_id, 0), UNSIGNED INTEGER) as parent_id'),
            ])
            ->orderBy('created_at', 'asc')
            ->get();

        foreach($categories as $category) {
            $default[] = $category;
        }

        return $this->respondArray($default);
    }
}
```

t. *Callback Controller*

```
<?php

namespace App\Http\Controllers;

use App\Transaction;
use App\Account;
use App\User;

use App\Utils\BusinessUtil;
use App\Utils\ContactUtil;
use App\Utils\ModuleUtil;
use App\Utils\ProductUtil;
use App\Utils\TransactionUtil;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use Auth;

class CallbackController extends Controller
{

    protected $productUtil;
    protected $transactionUtil;
    protected $contactUtil;
```

```
    protected $businessUtil;
    protected $moduleUtil;

    /**
     * Constructor
     *
     * @param ProductUtils $product
     * @return void
     */
    public function __construct(ProductUtil $productUtil, TransactionUtil $transactionUtil,
ContactUtil $contactUtil, BusinessUtil $businessUtil, ModuleUtil $moduleUtil)
    {
        $this->productUtil = $productUtil;
        $this->transactionUtil = $transactionUtil;
        $this->contactUtil = $contactUtil;
        $this->businessUtil = $businessUtil;
        $this->moduleUtil = $moduleUtil;
    }

    public function irs_callback(Request $request)
    {
        $input = $request->all();
        \Log::emergency(json_encode($input));

        if(!$input) {
            return $this->respondSuccess(null, "Test callback");
        }

        $status = $input['statuscode'];

        $sn = $input['sn'];
        $clientid = explode('-', $input['clientid']); // INV-{business_id}-{msisdn}-{refno}
        $type = $clientid[0];
        $refno = $clientid[1];
        $msisdn = $input['msisdn'];

        if($status == 1) {
            $sell = Transaction::where('additional_notes', $msisdn)
                ->where('type', 'sell')
                ->where('ref_no', 'PS'.$refno)
                ->first();

            $sell->update(['digital_status' => 'success']);

            $neraca = $this->transactionUtil->getBalanceSheet($sell->business_id);
            $sell->neraca = $neraca;

            DB::commit();

            activity('transaction')
                ->performedOn($sell)
                ->withProperties($sell)
                ->log('success');

            return $this->respondSuccess(null, "Transaksi sukses : ".$status);
        }

        if($type == 'INV') {
            $doReturnSell = $this->returnSell($refno, $msisdn);
            if($doReturnSell) {
                $doReturnPurchase = $this->returnPurchase($refno, $msisdn);
                if($doReturnPurchase) {
                    return $this->respondSuccess(null, "Transaksi Gagal, Berhasil me-retur
barang");
                } else {
                    return $this->respondFailed("Transaksi Gagal, Gagal me-retur barang.
CODE:PRCS");
                }
            } else {
                return $this->respondFailed("Transaksi Gagal, Gagal me-retur barang. CODE:SELL");
            }
        }
    }

    private function returnPurchase($trx_id, $msisdn)
    {
        try {
            $purchase = Transaction::where('additional_notes', $msisdn)
                        ->where('type', 'purchase')
                        ->where('ref_no', 'PB'.$trx_id)
                        ->with(['purchase_lines', 'purchase_lines.sub_unit'])
```

```
                        ->firstOrFail();

            $return_total = 0;

            DB::beginTransaction();

            foreach ($purchase->purchase_lines as $purchase_line) {
                $old_return_qty = $purchase_line->quantity_returned;

                $return_quantity = 1;

                $multiplier = 1;
                if (!empty($purchase_line->sub_unit->base_unit_multiplier)) {
                    $multiplier = $purchase_line->sub_unit->base_unit_multiplier;
                    $return_quantity = $return_quantity * $multiplier;
                }

                $purchase_line->quantity_returned = $return_quantity;
                $purchase_line->save();
                $return_total += $purchase_line->purchase_price * $purchase_line-
>quantity_returned;

                //Decrease quantity in variation location details
                if ($old_return_qty != $purchase_line->quantity_returned) {
                    $this->productUtil->decreaseProductQuantity(
                        $purchase_line->product_id,
                        $purchase_line->variation_id,
                        $purchase->location_id,
                        $purchase_line->quantity_returned,
                        $old_return_qty
                    );
                }
            }
            $return_total_inc_tax = $return_total; // +taxamount

            $return_transaction_data = [
                'total_before_tax' => $purchase->final_total,
                'final_total' => $purchase->final_total,
                'tax_amount' => 0,
                'tax_id' => null
            ];

            // $ref_count = $this->transactionUtil->setAndGetReferenceCount('purchase_return',
$purchase->business_id);
            // $return_transaction_data['ref_no'] = $this->transactionUtil-
>generateReferenceNumber('purchase_return', $ref_count, $purchase->business_id);
            $return_transaction_data['ref_no'] = time().'-'.$purchase->business_id.'-'.$trx_id;

            $return_transaction = Transaction::where('business_id', $purchase->business_id)
                                        ->where('type', 'purchase_return')
                                        ->where('return_parent_id', $purchase->id)
                                        ->first();

            if (!empty($return_transaction)) {
                $message = 'updated';
                $return_transaction->update($return_transaction_data);
            } else {
                $message = 'created';
                $return_transaction_data['business_id'] = $purchase->business_id;
                $return_transaction_data['location_id'] = $purchase->location_id;
                $return_transaction_data['type'] = 'purchase_return';
                $return_transaction_data['status'] = 'final';
                $return_transaction_data['contact_id'] = $purchase->contact_id;
                $return_transaction_data['transaction_date'] = date('Y-m-d h:i:s');
                $return_transaction_data['created_by'] = $purchase->created_by;
                $return_transaction_data['return_parent_id'] = $purchase->id;

                $return_transaction = Transaction::create($return_transaction_data);
            }

            $kipatronic = Account::where('business_id', $purchase->business_id)
                            ->where('accounts.type', 'saldo')
                            ->first();

            $payment = [
                [
                    'amount' => $purchase->final_total,
                    'method' => "cash",
                    'note' => 'Retur Penjualan Product Digital',
                    'account_id' => $kipatronic->id
                ]
```

```
            ];

            //Add Purchase payments
            $this->transactionUtil->createOrUpdatePaymentLines($return_transaction, $payment,
$purchase->business_id, $purchase->created_by, null, true, true);

            //update payment status
            $this->transactionUtil->updatePaymentStatus($return_transaction->id,
$return_transaction->final_total);

            $neraca = $this->transactionUtil->getBalanceSheet($purchase->business_id);
            $return_transaction->neraca = $neraca;

            DB::commit();

            activity('transaction')
                ->performedOn($return_transaction)
                ->withProperties($return_transaction)
                ->log($message);

            return true;
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            return false;
        }
    }

    private function returnSell($trx_id, $msisdn)
    {
        try {
            //Get parent sale
            $sell = Transaction::where('additional_notes', $msisdn)
                ->where('type', 'sell')
                ->where('ref_no', 'PS'.$trx_id)
                ->with(['sell_lines', 'sell_lines.sub_unit'])
                ->firstOrFail();

            $sell->update(['digital_status' => 'failed']);

            //Check if any sell return exists for the sale
            $sell_return = Transaction::where('business_id', $sell->business_id)
                ->where('type', 'sell_return')
                ->where('return_parent_id', $sell->id)
                ->first();

            $sell_return_data = [
                'transaction_date' => date('Y-m-d h:i:s'),
                'invoice_no' => '',
                'discount_type' => 'percentage',
                'discount_amount' => 0,
                'tax_id' => null,
                'tax_amount' => 0,
                'total_before_tax' => $sell->total_before_tax,
                'final_total' => $sell->final_total
            ];

            DB::beginTransaction();

            //Generate reference number
            if (empty($sell_return_data['invoice_no'])) {
                //Update reference count
                $ref_count = $this->productUtil->setAndGetReferenceCount('sell_return', $sell-
>business_id);
                $sell_return_data['invoice_no'] = $this->productUtil-
>generateReferenceNumber('sell_return', $ref_count, $sell->business_id);
            }

            if (empty($sell_return)) {
                $message = 'created';
                $sell_return_data['business_id'] = $sell->business_id;
                $sell_return_data['location_id'] = $sell->location_id;
                $sell_return_data['contact_id'] = $sell->contact_id;
                $sell_return_data['customer_group_id'] = $sell->customer_group_id;
                $sell_return_data['type'] = 'sell_return';
                $sell_return_data['status'] = 'final';
                $sell_return_data['created_by'] = $sell->created_by;
                $sell_return_data['return_parent_id'] = $sell->id;
                $sell_return = Transaction::create($sell_return_data);
```

```
            } else {
                $message = 'updated';
                $sell_return->update($sell_return_data);
            }

            if (User::find($sell->created_by)->business->enable_rp == 1 && !empty($sell->rp_earned)) {
                $is_reward_expired = $this->transactionUtil->isRewardExpired($sell->transaction_date, $sell->business_id);
                if (!$is_reward_expired) {
                    $diff = $sell->final_total - $sell_return->final_total;
                    $new_reward_point = $this->transactionUtil->calculateRewardPoints($sell->business_id, $diff);
                    $this->transactionUtil->updateCustomerRewardPoints($sell->contact_id, $new_reward_point, $sell->rp_earned);

                    $sell->rp_earned = $new_reward_point;
                    $sell->save();
                }
            }

            $payment = [
                [
                    'amount' => $sell_return->final_total,
                    'method' => "cash",
                    'note' => 'Retur Penjualan',
                    'account_id' => Account::where('outlet_id', $sell->location_id)->first()->id

                ]
            ];

            //Add Purchase payments
            $this->transactionUtil->createOrUpdatePaymentLines($sell_return, $payment, $sell->business_id, $sell->created_by, true, true);

            //Update payment status
            $this->transactionUtil->updatePaymentStatus($sell_return->id, $sell_return->final_total);

            foreach ($sell->sell_lines as $sell_line) {
                $quantity = 1;

                $quantity_before = $this->transactionUtil->num_f($sell_line->quantity_returned);
                $quantity_formated = $this->transactionUtil->num_f($quantity);

                $sell_line->quantity_returned = $quantity;
                $sell_line->save();

                //update quantity sold in corresponding purchase lines
                $this->transactionUtil->updateQuantitySoldFromSellLine($sell_line, $quantity_formated, $quantity_before);

                // Update quantity in variation location details
                $this->productUtil->updateProductQuantity($sell_return->location_id, $sell_line->product_id, $sell_line->variation_id, $quantity_formated, $quantity_before);

            }

            $neraca = $this->transactionUtil->getBalanceSheet($sell->business_id);
            $sell_return->neraca = $neraca;
            $sell->neraca = $neraca;

            DB::commit();

            activity('transaction')
                ->performedOn($sell_return)
                ->withProperties($sell_return)
                ->log($message);

            activity('transaction')
                ->performedOn($sell)
                ->withProperties($sell)
                ->log('failed');

            return true;
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e->getMessage());

            return false;
```

```
        }
    }
}
```

u. *Report Controller*

```
<?php

namespace App\Http\Controllers;

use App\Account;
use App\Brands;
use App\BusinessLocation;
use App\CashRegister;
use App\Category;

use App\Contact;
use App\CustomerGroup;

use App\Business;

use App\ExpenseCategory;
use App\Product;
use App\PurchaseLine;
use App\Restaurant\ResTable;
use App\SellingPriceGroup;
use App\Transaction;
use App\TransactionPayment;
use App\TransactionSellLine;
use App\TransactionSellLinesPurchaseLines;
use App\Unit;
use App\User;
use App\Utils\ModuleUtil;
use App\Utils\ProductUtil;
use App\Utils\TransactionUtil;
use App\Utils\CashRegisterUtil;
use App\Utils\AccountUtil;
use App\Variation;
use App\VariationLocationDetails;
use Charts;
use Datatables;
use DB;
use Illuminate\Http\Request;
use Auth;

class ReportController extends Controller
{
    protected $transactionUtil;
    protected $productUtil;
    protected $moduleUtil;
    protected $cashRegisterUtil;
    protected $accountUtil;

    public function __construct(TransactionUtil $transactionUtil, ProductUtil $productUtil,
ModuleUtil $moduleUtil, CashRegisterUtil $cashRegisterUtil, AccountUtil $accountUtil)
    {
        $this->transactionUtil = $transactionUtil;
        $this->productUtil = $productUtil;
        $this->moduleUtil = $moduleUtil;
        $this->cashRegisterUtil = $cashRegisterUtil;
        $this->accountUtil = $accountUtil;
    }

    public function getReportPenjualan(Request $request)
    {
        if (!Auth::user()->can('profit_loss_report.view')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;

        $show_manufacturing_data = false;

        $location_id = '';

        if ($request->input('location_id') && $request->input('location_id') > 0) {
            $location_id = $request->input('location_id');
        }
        //Return the details in ajax call
        if($request->input('date_filter') == 'harian') {
            $start_date = \Carbon\Carbon::now()->startOfDay()->format('Y-m-d');
```

```
            $end_date = \Carbon\Carbon::now()->endOfDay()->format('Y-m-d');
        } else if($request->input('date_filter') == 'bulanan'){
            $start_date = \Carbon\Carbon::now()->startOfMonth()->format('Y-m-d');
            $end_date = \Carbon\Carbon::now()->endOfMonth()->format('Y-m-d');
        } else if($request->input('date_filter') == 'tahunan'){
            $start_date = \Carbon\Carbon::now()->startOfYear()->format('Y-m-d');
            $end_date = \Carbon\Carbon::now()->endOfYear()->format('Y-m-d');
        } else {
            $start_date = \Carbon\Carbon::now()->startOfDay()->format('Y-m-d');
            $end_date = \Carbon\Carbon::now()->endOfDay()->format('Y-m-d');
        }
        // $location_id = $request->get('location_id');


        //For Opening stock date should be 1 day before
        $day_before_start_date = \Carbon\Carbon::createFromFormat('Y-m-d', $start_date)-
>subDay()->format('Y-m-d');
        //Get Opening stock
        $opening_stock = $this->transactionUtil->getOpeningClosingStock($business_id,
$day_before_start_date, $location_id, true);

        //Get Closing stock
        $closing_stock = $this->transactionUtil->getOpeningClosingStock(
            $business_id,
            $end_date,
            $location_id
        );

        //Get Purchase details
        $purchase_details = $this->transactionUtil->getPurchaseTotals(
            $business_id,
            $start_date,
            $end_date,
            $location_id
        );

        //Get Sell details
        $sell_details = $this->transactionUtil->getSellTotals(
            $business_id,
            $start_date,
            $end_date,
            $location_id
        );

        $transaction_types = [
            'purchase_return', 'sell_return', 'expense', 'stock_adjustment', 'sell_transfer',
'purchase_transfer', 'purchase', 'sell', 'income'
        ];

        $show_total_payroll = false;

        $transaction_totals = $this->transactionUtil->getTransactionTotals(
            $business_id,
            $transaction_types,
            $start_date,
            $end_date,
            $location_id
        );

        $total_expense = $this->transactionUtil->getTotalBonusExpense($business_id, $location_id,
$start_date, $end_date);

        $saldoKipaId = $this->accountUtil->getBalanceAccount($business_id);
        $saldoBonus = $this->accountUtil->getSubBalance($saldoKipaId, 'bonus');

        if($saldoBonus >= 0) {
            $transaction_totals['total_expense'] -= $total_expense;
        }

        $gross_profit = $this->transactionUtil->getGrossProfit(
            $business_id,
            $start_date,
            $end_date,
            $location_id
        );

        $pokokPembelian = $this->transactionUtil->getPokokPembelian(
            $business_id,
            $start_date,
            $end_date,
            $location_id
```

```
        );

        $data = [];

        //Manufacturing module data.
        if ($show_manufacturing_data) {
            $data['total_production_cost'] = $this->transactionUtil->getTotalProductionCost(
                $business_id,
                $start_date,
                $end_date,
                $location_id
                );
        } else {
            $data['total_production_cost'] = 0;
        }

        $total_transfer_shipping_charges =
$transaction_totals['total_transfer_shipping_charges'];

        //Add total sell shipping charges to $total_transfer_shipping_charges
        if (!empty($sell_details['total_shipping_charges'])) {
            $total_transfer_shipping_charges += $sell_details['total_shipping_charges'];
        }
        //Add total purchase shipping charges to $total_transfer_shipping_charges
        if (!empty($purchase_details['total_shipping_charges'])) {
            $total_transfer_shipping_charges += $purchase_details['total_shipping_charges'];
        }

        //Discounts
        $total_purchase_discount = $transaction_totals['total_purchase_discount'];
        $total_sell_discount = $transaction_totals['total_sell_discount'];
        $total_reward_amount = $transaction_totals['total_reward_amount'];

        $data['opening_stock'] = !empty($opening_stock) ? $opening_stock : 0;
        $data['closing_stock'] = !empty($closing_stock) ? $closing_stock : 0;
        $data['total_purchase'] = !empty($purchase_details['total_purchase_exc_tax']) ?
$purchase_details['total_purchase_exc_tax'] : 0;
        $data['total_sell'] = !empty($sell_details['total_sell_exc_tax']) ?
$sell_details['total_sell_exc_tax'] : 0;
        $data['count_sell'] = !empty($sell_details['count_sell']) ? $sell_details['count_sell'] :
0;
        $data['count_product_sold'] = !empty($sell_details['count_product_sold']) ?
$sell_details['count_product_sold'] : 0;
        $data['total_expense'] =  $transaction_totals['total_expense'];
        $data['total_income'] =  $transaction_totals['total_income'];
        $data['pokok_pembelian'] = !empty($pokokPembelian) ? $pokokPembelian : 0;

        $total_payroll = 0;
        if ($show_total_payroll) {
            $data['total_payroll'] =  $transaction_totals['total_payroll'];
            $total_payroll = $transaction_totals['total_payroll'];
        }

        $data['total_adjustment'] = $transaction_totals['total_adjustment'];
        $data['total_stok_masuk'] = $transaction_totals['total_purchase_transfer'];
        $data['total_stok_keluar'] = $transaction_totals['total_sell_transfer'];

        $data['closing_stock'] = $data['closing_stock'] - $data['total_adjustment'];

        $data['total_recovered'] = $transaction_totals['total_recovered'];

        $data['total_transfer_shipping_charges'] = $total_transfer_shipping_charges;

        $data['total_purchase_discount'] = !empty($total_purchase_discount) ?
$total_purchase_discount : 0;
        $data['total_sell_discount'] = !empty($total_sell_discount) ? $total_sell_discount : 0;

        $data['total_reward_amount'] = !empty($total_reward_amount) ? $total_reward_amount : 0;

        $data['total_purchase_return'] = $transaction_totals['total_purchase_return_exc_tax'];

        $data['total_sell_return'] = $transaction_totals['total_sell_return_exc_tax'];

        // $data['closing_stock'] = $data['closing_stock'] - $data['total_sell_return'];

        $data['closing_stock'] = $data['opening_stock'] + $data['total_purchase'] -
$data['pokok_pembelian'] - $data['total_purchase_return'] - $data['total_adjustment'];

        $data['net_profit'] = $data['total_sell'] - $data['total_sell_return'] -
$data['total_sell_discount'] - $data['total_reward_amount'] -
```

```
                                $data['total_expense'] - $data['total_production_cost'] -
$total_payroll + $data['total_recovered'] - $data['total_adjustment'] -
                                $data['total_transfer_shipping_charges'] +
$data['total_purchase_discount']
                                    - $data['pokok_pembelian'] + $data['total_income'];

        // dd($data);
        $data['gross_profit'] = round((int)$gross_profit);
        $data['net_profit'] = round($data['net_profit']);

        $accounts = Account::leftjoin('account_transactions as AT', function ($join) {
                        $join->on('AT.account_id', '=', 'accounts.id');
                        $join->whereNull('AT.deleted_at');
                    })
                    ->leftJoin('account_categories as AC', function ($join) {
                        $join->on('AC.id', '=', 'accounts.account_categories_id');
                    })
                    ->where('business_id', $business_id)
                    ->where('accounts.account_categories_id', 1)
                    ->select(['accounts.name', 'accounts.account_number',
                        'AC.name as category',DB::raw("CONVERT(IFNULL( SUM(IF(AT.type='credit',
amount, -1*amount)), 0), INTEGER) as balance")])
                    ->groupBy('accounts.id')->get();

        $data['accounts'] = $accounts;

        // dd($data);

        return $this->respondSuccess($data);
    }

    /**
     * Shows profit\loss of a business
     *
     * @return \Illuminate\Http\Response
     */
    public function getProfitLoss(Request $request)
    {
        if (!Auth::user()->can('profit_loss_report.view')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;

        $show_manufacturing_data = false;

        $start_date = $request->get('start_date');
        $end_date = $request->get('end_date');
        // $location_id = $request->get('location_id');
        $location_id = '';
        if ($request->input('location_id')) {
            $location_id = $request->input('location_id');
        }

        //For Opening stock date should be 1 day before
        $day_before_start_date = \Carbon\Carbon::createFromFormat('Y-m-d', $start_date)-
>subDay()->format('Y-m-d');
        //Get Opening stock
        $opening_stock = $this->transactionUtil->getOpeningClosingStock($business_id,
$day_before_start_date, $location_id, true);

        //Get Closing stock
        $closing_stock = $this->transactionUtil->getOpeningClosingStock(
            $business_id,
            $end_date,
            $location_id
        );

        //Get Purchase details
        $purchase_details = $this->transactionUtil->getPurchaseTotals(
            $business_id,
            $start_date,
            $end_date,
            $location_id
        );

        //Get Sell details
        $sell_details = $this->transactionUtil->getSellTotals(
            $business_id,
            $start_date,
            $end_date,
```

```
            $location_id
        );

        $transaction_types = [
            'purchase_return', 'sell_return', 'expense', 'stock_adjustment', 'sell_transfer',
'purchase_transfer', 'purchase', 'sell'
        ];

        $show_total_payroll = false;

        $transaction_totals = $this->transactionUtil->getTransactionTotals(
            $business_id,
            $transaction_types,
            $start_date,
            $end_date,
            $location_id
        );

        $saldoKipaId = $this->accountUtil->getBalanceAccount($business_id);
        $saldoBonus = $this->accountUtil->getSubBalance($saldoKipaId, 'bonus');

        if($saldoBonus >= 0) {
            $transaction_totals['total_expense'] -= $total_expense;
        }

        $gross_profit = $this->transactionUtil->getGrossProfit(
            $business_id,
            $start_date,
            $end_date,
            $location_id
        );

        $pokokPembelian = $this->transactionUtil->getPokokPembelian(
            $business_id,
            $start_date,
            $end_date,
            $location_id
        );

        $data = [];

        //Manufacturing module data.
        if ($show_manufacturing_data) {
            $data['total_production_cost'] = $this->transactionUtil->getTotalProductionCost(
                $business_id,
                $start_date,
                $end_date,
                $location_id
                );
        } else {
            $data['total_production_cost'] = 0;
        }

        $total_transfer_shipping_charges =
$transaction_totals['total_transfer_shipping_charges'];

        //Add total sell shipping charges to $total_transfer_shipping_charges
        if (!empty($sell_details['total_shipping_charges'])) {
            $total_transfer_shipping_charges += $sell_details['total_shipping_charges'];
        }
        //Add total purchase shipping charges to $total_transfer_shipping_charges
        if (!empty($purchase_details['total_shipping_charges'])) {
            $total_transfer_shipping_charges += $purchase_details['total_shipping_charges'];
        }

        //Discounts
        $total_purchase_discount = $transaction_totals['total_purchase_discount'];
        $total_sell_discount = $transaction_totals['total_sell_discount'];
        $total_reward_amount = $transaction_totals['total_reward_amount'];

        $data['opening_stock'] = !empty($opening_stock) ? $opening_stock : 0;
        $data['closing_stock'] = !empty($closing_stock) ? $closing_stock : 0;
        $data['total_purchase'] = !empty($purchase_details['total_purchase_exc_tax']) ?
$purchase_details['total_purchase_exc_tax'] : 0;
        $data['total_sell'] = !empty($sell_details['total_sell_exc_tax']) ?
$sell_details['total_sell_exc_tax'] : 0;
        $data['total_expense'] =  $transaction_totals['total_expense'];
        $data['pokok_pembelian'] = !empty($pokokPembelian) ? $pokokPembelian : 0;

        $total_payroll = 0;
        if ($show_total_payroll) {
```

```
            $data['total_payroll'] =  $transaction_totals['total_payroll'];
            $total_payroll = $transaction_totals['total_payroll'];
        }

        $data['total_adjustment'] = $transaction_totals['total_adjustment'];
        $data['total_stok_masuk'] = $transaction_totals['total_purchase_transfer'];
        $data['total_stok_keluar'] = $transaction_totals['total_sell_transfer'];

        $data['closing_stock'] = $data['closing_stock'] - $data['total_adjustment'];

        $data['total_recovered'] = $transaction_totals['total_recovered'];

        $data['total_transfer_shipping_charges'] = $total_transfer_shipping_charges;

        $data['total_purchase_discount'] = !empty($total_purchase_discount) ?
$total_purchase_discount : 0;
        $data['total_sell_discount'] = !empty($total_sell_discount) ? $total_sell_discount : 0;

        $data['total_reward_amount'] = !empty($total_reward_amount) ? $total_reward_amount : 0;

        $data['total_purchase_return'] = $transaction_totals['total_purchase_return_exc_tax'];

        $data['total_sell_return'] = $transaction_totals['total_sell_return_exc_tax'];

        // $data['closing_stock'] = $data['closing_stock'] - $data['total_sell_return'];

        $data['closing_stock'] = $data['opening_stock'] + $data['total_purchase'] -
$data['pokok_pembelian'] - $data['total_purchase_return'] - $data['total_adjustment'];

        $data['net_profit'] = $data['total_sell'] - $data['total_sell_return'] -
$data['total_sell_discount'] - $data['total_reward_amount'] -
                                    $data['total_expense'] - $data['total_production_cost'] -
$total_payroll + $data['total_recovered'] - $data['total_adjustment'] -
                                $data['total_transfer_shipping_charges'] +
$data['total_purchase_discount']
                                        - $data['pokok_pembelian'];

        // dd($data);
        $data['gross_profit'] = $gross_profit;

        // dd($data);

        return $this->respondArray($data);
    }

    public function getDashboard(Request $request)
    {

        $business_id = Auth::user()->business_id;

        $show_manufacturing_data = false;

        if(Auth::user()->getRoleNameAttribute() == "Admin") {
            $start_date = \Carbon\Carbon::now()->startofMonth()->format('Y-m-d');
            $end_date = \Carbon\Carbon::now()->endOfMonth()->format('Y-m-d');
        } else {
            $start_date = \Carbon\Carbon::now()->format('Y-m-d');
            $end_date = $start_date;
        }

        // $location_id = $request->get('location_id');
        $location_id = null;
        if ($request->input('location_id')) {
            $location_id = $request->input('location_id');
        }

        //For Opening stock date should be 1 day before
        $day_before_start_date = \Carbon\Carbon::createFromFormat('Y-m-d', $start_date)-
>subDay()->format('Y-m-d');
        //Get Opening stock
        $opening_stock = $this->transactionUtil->getOpeningClosingStock($business_id,
$day_before_start_date, $location_id, true);

        //Get Closing stock
        $closing_stock = $this->transactionUtil->getOpeningClosingStock(
            $business_id,
            $end_date,
            $location_id
        );

        //Get Purchase details
```

```
        $purchase_details = $this->transactionUtil->getPurchaseTotals(
            $business_id,
            $start_date,
            $end_date,
            $location_id
        );

        //Get Sell details
        $sell_details = $this->transactionUtil->getSellTotals(
            $business_id,
            $start_date,
            $end_date,
            $location_id
        );

        $transaction_types = [
            'purchase_return', 'sell_return', 'expense', 'stock_adjustment', 'sell_transfer',
'purchase', 'sell'
        ];

        $show_total_payroll = false;

        $transaction_totals = $this->transactionUtil->getTransactionTotals(
            $business_id,
            $transaction_types,
            $start_date,
            $end_date,
            $location_id
        );

        $gross_profit = $this->transactionUtil->getGrossProfit(
            $business_id,
            $start_date,
            $end_date,
            $location_id
        );

        $data = [];

        $total_transfer_shipping_charges =
$transaction_totals['total_transfer_shipping_charges'];

        //Add total sell shipping charges to $total_transfer_shipping_charges
        if (!empty($sell_details['total_shipping_charges'])) {
            $total_transfer_shipping_charges += $sell_details['total_shipping_charges'];
        }
        //Add total purchase shipping charges to $total_transfer_shipping_charges
        if (!empty($purchase_details['total_shipping_charges'])) {
            $total_transfer_shipping_charges += $purchase_details['total_shipping_charges'];
        }

        //Discounts
        $total_purchase_discount = $transaction_totals['total_purchase_discount'];
        $total_sell_discount = $transaction_totals['total_sell_discount'];
        $total_reward_amount = $transaction_totals['total_reward_amount'];
        $data['total_sell'] = !empty($sell_details['total_sell_exc_tax']) ?
$sell_details['total_sell_exc_tax'] : 0;

        $wallet = Account::leftjoin('account_transactions as AT', function ($join) {
            $join->on('AT.account_id', '=', 'accounts.id');
            $join->whereNull('AT.deleted_at');
        })
                        ->where('business_id', $business_id)
                        ->where('accounts.type', 'saldo')
                        ->select(['name', 'account_number', 'is_default', 'is_digital',
'accounts.note', 'accounts.id',
                            'is_closed', DB::raw("SUM( IF(AT.type='credit', amount, -
1*amount) ) as balance")])
                        ->groupBy('accounts.id');

        $wallet = $wallet->first();

        if($wallet){
            $data['balance'] = ($wallet->balance == null) ? 0 : $wallet->balance;
        } else {
            $data['balance'] = 0;
        }

        $all_notifications = Auth::user()->notifications;
        $unread_notifications = $all_notifications->where('read_at', null);
        $total_unread = count($unread_notifications);
```

```
        $data['count_unread_notification'] = $total_unread;

        return $this->respondSuccess($data);
    }

    public function getPurchaseSell(Request $request)
    {
        if (!auth()->user()->can('purchase_n_sell_report.view')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = $request->session()->get('user.business_id');

        //Return the details in ajax call
        if ($request->ajax()) {
            $start_date = $request->get('start_date');
            $end_date = $request->get('end_date');

            // $location_id = $request->get('location_id');
            $location_id = '';
            if ($request->session()->has('business_location')) {
                $location_id = session('business_location')['id'];
            }

            $purchase_details = $this->transactionUtil->getPurchaseTotals($business_id,
$start_date, $end_date, $location_id);

            $sell_details = $this->transactionUtil->getSellTotals(
                $business_id,
                $start_date,
                $end_date,
                $location_id
            );

            $transaction_types = [
                'purchase_return', 'sell_return'
            ];

            $transaction_totals = $this->transactionUtil->getTransactionTotals(
                $business_id,
                $transaction_types,
                $start_date,
                $end_date,
                $location_id
            );

            $total_purchase_return_inc_tax =
$transaction_totals['total_purchase_return_inc_tax'];
            $total_sell_return_inc_tax = $transaction_totals['total_sell_return_inc_tax'];

            $difference = [
                'total' => $sell_details['total_sell_inc_tax'] + $total_sell_return_inc_tax -
$purchase_details['total_purchase_inc_tax'] - $total_purchase_return_inc_tax,
                'due' => $sell_details['invoice_due'] - $purchase_details['purchase_due']
            ];

            return ['purchase' => $purchase_details,
                    'sell' => $sell_details,
                    'total_purchase_return' => $total_purchase_return_inc_tax,
                    'total_sell_return' => $total_sell_return_inc_tax,
                    'difference' => $difference
                ];
        }

        $business_locations = BusinessLocation::forDropdown($business_id, true);

        return view('report.purchase_sell')
                    ->with(compact('business_locations'));
    }

    public function getCustomer(Request $request)
    {
        if (!Auth::user()->can('contacts_report.view')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;

        $contacts = Contact::where('contacts.business_id', $business_id)
                ->join('transactions AS t', 'contacts.id', '=', 't.contact_id')
```

```
                ->groupBy('contacts.id')
                ->select(
                    DB::raw("SUM(IF(t.type = 'purchase', final_total, 0)) as total_purchase"),
                    DB::raw("SUM(IF(t.type = 'purchase_return', final_total, 0)) as
total_purchase_return"),
                    DB::raw("SUM(IF(t.type = 'sell' AND t.status = 'final', final_total, 0)) as
total_invoice"),
                    DB::raw("SUM(IF(t.type = 'purchase', (SELECT SUM(amount) FROM
transaction_payments WHERE transaction_payments.transaction_id=t.id), 0)) as purchase_paid"),
                    DB::raw("SUM(IF(t.type = 'sell' AND t.status = 'final', (SELECT
SUM(IF(is_return = 1,-1*amount,amount)) FROM transaction_payments WHERE
transaction_payments.transaction_id=t.id), 0)) as invoice_received"),
                    DB::raw("SUM(IF(t.type = 'sell_return', (SELECT SUM(amount) FROM
transaction_payments WHERE transaction_payments.transaction_id=t.id), 0)) as sell_return_paid"),
                    DB::raw("SUM(IF(t.type = 'purchase_return', (SELECT SUM(amount) FROM
transaction_payments WHERE transaction_payments.transaction_id=t.id), 0)) as
purchase_return_received"),
                    DB::raw("SUM(IF(t.type = 'sell_return', final_total, 0)) as
total_sell_return"),
                    DB::raw("SUM(IF(t.type = 'opening_balance', final_total, 0)) as
opening_balance"),
                    DB::raw("SUM(IF(t.type = 'opening_balance', (SELECT SUM(IF(is_return = 1,-
1*amount,amount)) FROM transaction_payments WHERE transaction_payments.transaction_id=t.id), 0))
as opening_balance_paid"),
                    'contacts.supplier_business_name',
                    'contacts.name',
                    'contacts.id'
                );
        $permitted_locations = Auth::user()->permitted_locations();

        if ($permitted_locations != 'all') {
            $contacts->whereIn('t.location_id', $permitted_locations);
        }

        if (!empty($request->input('customer_group_id'))) {
            $contacts->where('contacts.customer_group_id', $request->input('customer_group_id'));
        }

        $start_date = $request->get('start_date');
        $end_date = $request->get('end_date');
        if (!empty($start_date) && !empty($end_date)) {
            $contacts->whereBetween(DB::raw('date(t.transaction_date)'), [$start_date,
$end_date]);
        }

        $contacts->where('contacts.type', 'customer');

        return $this->respondArray($contacts->get());
    }

    public function getSuppliers(Request $request)
    {
        if (!Auth::user()->can('contacts_report.view')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;

        $contacts = Contact::where('contacts.business_id', $business_id)
                ->join('transactions AS t', 'contacts.id', '=', 't.contact_id')
                ->groupBy('contacts.id')
                ->select(
                    DB::raw("SUM(IF(t.type = 'purchase', final_total, 0)) as total_purchase"),
                    DB::raw("SUM(IF(t.type = 'purchase_return', final_total, 0)) as
total_purchase_return"),
                    DB::raw("SUM(IF(t.type = 'sell' AND t.status = 'final', final_total, 0)) as
total_invoice"),
                    DB::raw("SUM(IF(t.type = 'purchase', (SELECT SUM(amount) FROM
transaction_payments WHERE transaction_payments.transaction_id=t.id), 0)) as purchase_paid"),
                    DB::raw("SUM(IF(t.type = 'sell' AND t.status = 'final', (SELECT
SUM(IF(is_return = 1,-1*amount,amount)) FROM transaction_payments WHERE
transaction_payments.transaction_id=t.id), 0)) as invoice_received"),
                    DB::raw("SUM(IF(t.type = 'sell_return', (SELECT SUM(amount) FROM
transaction_payments WHERE transaction_payments.transaction_id=t.id), 0)) as sell_return_paid"),
                    DB::raw("SUM(IF(t.type = 'purchase_return', (SELECT SUM(amount) FROM
transaction_payments WHERE transaction_payments.transaction_id=t.id), 0)) as
purchase_return_received"),
                    DB::raw("SUM(IF(t.type = 'sell_return', final_total, 0)) as
total_sell_return"),
                    DB::raw("SUM(IF(t.type = 'opening_balance', final_total, 0)) as
opening_balance"),
```

```
                    DB::raw("SUM(IF(t.type = 'opening_balance', (SELECT SUM(IF(is_return = 1,-
1*amount,amount)) FROM transaction_payments WHERE transaction_payments.transaction_id=t.id), 0))
as opening_balance_paid"),
                    'contacts.supplier_business_name',
                    'contacts.name',
                    'contacts.id'
        );

        $permitted_locations = Auth::user()->permitted_locations();

        if ($permitted_locations != 'all') {
            $contacts->whereIn('t.location_id', $permitted_locations);
        }

        if (!empty($request->input('customer_group_id'))) {
            $contacts->where('contacts.customer_group_id', $request->input('customer_group_id'));
        }

        $start_date = $request->get('start_date');
        $end_date = $request->get('end_date');
        if (!empty($start_date) && !empty($end_date)) {
            $contacts->whereBetween(DB::raw('date(t.transaction_date)'), [$start_date,
$end_date]);
        }

        $contacts->where('contacts.type', 'supplier');

        return $this->respondArray($contacts->get());
    }

    public function getStockReport(Request $request)
    {
        if (!Auth::user()->can('stock_report.view')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;
        $business = Business::findOrFail($business_id);
        $account_method = $business->accounting_method;

        $selling_price_groups = SellingPriceGroup::where('business_id', $business_id)
                                                 ->get();
        $allowed_selling_price_group = false;
        foreach ($selling_price_groups as $selling_price_group) {
            if (Auth::user()->can('selling_price_group.' . $selling_price_group->id)) {
                $allowed_selling_price_group = true;
                break;
            }
        }


        $query = Variation::join('products as p', 'p.id', '=', 'variations.product_id')
                ->join('units', 'p.unit_id', '=', 'units.id')
                ->leftjoin('variation_location_details as vld', 'variations.id', '=',
'vld.variation_id')
                ->join('product_variations as pv', 'variations.product_variation_id', '=',
'pv.id')
                ->where('p.business_id', $business_id)
                ->where('p.product_type', 'fisik')
                ->whereIn('p.type', ['single', 'variable']);

        $permitted_locations = Auth::user()->permitted_locations();
        $location_filter = '';

        if ($permitted_locations != 'all') {
            $query->whereIn('vld.location_id', $permitted_locations);

            $locations_imploded = implode(', ', $permitted_locations);
            $location_filter .= "AND transactions.location_id IN ($locations_imploded) ";
        }

        $location_id = null;
        if ($request->input('location_id')) {
            $location_id = $request->input('location_id');
        }
        $vld_location_filter = '';
        if (!empty($location_id)) {
            // $query->where('vld.location_id', $location_id);

            $location_filter .= "AND transactions.location_id=$location_id";
            $vld_location_filter = "AND variation_location_details.location_id=$location_id";
```

```
        }

        if (!empty($request->input('category_id'))) {
            $query->where('p.category_id', $request->input('category_id'));
        }
        if (!empty($request->input('sub_category_id'))) {
            $query->where('p.sub_category_id', $request->input('sub_category_id'));
        }
        if (!empty($request->input('brand_id'))) {
            $query->where('p.brand_id', $request->input('brand_id'));
        }
        if (!empty($request->input('unit_id'))) {
            $query->where('p.unit_id', $request->input('unit_id'));
        }

        $tax_id = $request->get('tax_id', null);
        if (!empty($tax_id)) {
            $query->where('p.tax', $tax_id);
        }

        $type = $request->get('type', null);
        if (!empty($type)) {
            $query->where('p.type', $type);
        }

        $only_mfg_products = $request->get('only_mfg_products', 0);
        if (!empty($only_mfg_products)) {
            $query->join('mfg_recipes as mr', 'mr.variation_id', '=', 'variations.id');
        }

        $products = $query->select(
            DB::raw("CONVERT(
                IFNULL(
                        (SELECT SUM(TSL.quantity) FROM transactions
                        JOIN purchase_lines AS TSL ON transactions.id=TSL.transaction_id
                        WHERE TSL.variation_id=variations.id $location_filter
                        ), 0
                    ), UNSIGNED INTEGER
                ) as total_stock"),
            DB::raw("CONVERT(
                        IFNULL(
                                (SELECT SUM(TSL.quantity - TSL.quantity_returned -
TSL.quantity_adjusted - TSL.quantity_sold) FROM transactions
                                JOIN purchase_lines AS TSL ON transactions.id=TSL.transaction_id
                                WHERE TSL.variation_id=variations.id $location_filter
                                ), 0
                            ), UNSIGNED INTEGER
                        ) as stock"),
            DB::raw("CONVERT(
                        IFNULL(
                                (SELECT SUM(TSL.quantity - TSL.quantity_returned) FROM
transactions
                                    JOIN transaction_sell_lines AS TSL ON
transactions.id=TSL.transaction_id
                                    WHERE transactions.status='final' AND
transactions.type='sell' $location_filter
                                        AND TSL.variation_id=variations.id
                                ), 0
                            ), UNSIGNED INTEGER
                        ) as total_sold"),
            DB::raw("CONVERT(
                        IFNULL(
                                (SELECT SUM(IF(transactions.type='sell_transfer', TSL.quantity,
0) ) FROM transactions
                                JOIN transaction_sell_lines AS TSL ON
transactions.id=TSL.transaction_id
                                WHERE transactions.status='final' AND
transactions.type='sell_transfer' $location_filter
                                AND (TSL.variation_id=variations.id)
                                ), 0
                            ), UNSIGNED INTEGER
                        ) as total_transfered"),
            DB::raw("CONVERT(
                        IFNULL(
                                (SELECT SUM(IF(transactions.type='stock_adjustment',
SAL.quantity, 0) ) FROM transactions
                                JOIN stock_adjustment_lines AS SAL ON
transactions.id=SAL.transaction_id
                                WHERE transactions.status='received' AND
transactions.type='stock_adjustment' $location_filter
                                AND (SAL.variation_id=variations.id)
```

```
                                ), 0
                            ), UNSIGNED INTEGER
                        ) as total_adjusted"),
            DB::raw("CONVERT(
                IFNULL(
                        (SELECT SUM(variation_location_details.qty_available) FROM
variation_location_details
                        WHERE variation_location_details.variation_id=variations.id
                        $vld_location_filter
                        ), 0
                    ), UNSIGNED INTEGER
                ) as stock_available"),
            // DB::raw("CONVERT(IFNULL(SUM(vld.qty_available), 0), UNSIGNED INTEGER) as stock"),
            DB::raw("CONVERT(IFNULL(p.alert_quantity, 0), UNSIGNED INTEGER) as alert_quantity"),
            'variations.sub_sku as sku',
            'variations.id as variation_id',
            'p.name as product',
            'p.image as product_image',
            'p.type',
            'p.id as product_id',
            'units.short_name as unit',
            'p.enable_stock as enable_stock',
            'pv.name as product_variation',
            'variations.name as variation_name'
        );

        $pl_query_string = $this->productUtil->get_pl_quantity_sum_string('PL');
        $products->addSelect(
            DB::raw("CONVERT(
                        IFNULL(
                                (SELECT COALESCE(SUM(PL.quantity - ($pl_query_string)), 0) FROM
transactions
                                JOIN purchase_lines AS PL ON transactions.id=PL.transaction_id
                                WHERE transactions.status='received' AND
transactions.type='production_purchase' $location_filter
                                AND (PL.variation_id=variations.id)
                                ), 0
                            ), UNSIGNED INTEGER
                        ) as total_mfg_stock")
        );

        if($request->input('unit_price') == 0) {
            $products->addSelect(DB::raw('CONVERT(IFNULL(variations.default_purchase_price, 0),
UNSIGNED INTEGER) as unit_price'));
        } else {
            $products->addSelect(DB::raw('CONVERT(IFNULL(variations.sell_price_inc_tax, 0),
UNSIGNED INTEGER) as unit_price'));
        }

        $products->groupBy('variations.id');

        $start = 0;
        $limit = $request->input('limit') ? $request->input('limit') : 20;
        $paged = $request->input('page') ? $request->input('page') : 1;

        $start = ($paged - 1) * $limit;
        $products = $products->offset($start)->limit($limit)->get();

        foreach($products as $key => $product) {
            $products[$key]->product_image = $product->product_image ? env('AWS_CDN_URL') .
'/img/' . $product->product_image : '';
        }

        return $this->respondArray($products);
    }

    public function getStockDetails(Request $request)
    {
        //Return the details in ajax call
        if ($request->ajax()) {
            $business_id = $request->session()->get('user.business_id');
            $product_id = $request->input('product_id');
            $query = Product::leftjoin('units as u', 'products.unit_id', '=', 'u.id')
                ->join('variations as v', 'products.id', '=', 'v.product_id')
                ->join('product_variations as pv', 'pv.id', '=', 'v.product_variation_id')
                ->leftjoin('variation_location_details as vld', 'v.id', '=', 'vld.variation_id')
                ->where('products.business_id', $business_id)
                ->where('products.id', $product_id)
                ->whereNull('v.deleted_at');

            $permitted_locations = auth()->user()->permitted_locations();
```

```
            $location_filter = '';
            if ($permitted_locations != 'all') {
                $query->whereIn('vld.location_id', $permitted_locations);
                $locations_imploded = implode(', ', $permitted_locations);
                $location_filter .= "AND transactions.location_id IN ($locations_imploded) ";
            }

            if (!empty($request->input('location_id'))) {
                $location_id = $request->input('location_id');

                $query->where('vld.location_id', $location_id);

                $location_filter .= "AND transactions.location_id=$location_id";
            }

            $product_details =  $query->select(
                'products.name as product',
                'u.short_name as unit',
                'pv.name as product_variation',
                'v.name as variation',
                'v.sub_sku as sub_sku',
                'v.sell_price_inc_tax',
                DB::raw("SUM(vld.qty_available) as stock"),
                DB::raw("(SELECT SUM(IF(transactions.type='sell', TSL.quantity -
TSL.quantity_returned, -1* TPL.quantity) ) FROM transactions
                        LEFT JOIN transaction_sell_lines AS TSL ON
transactions.id=TSL.transaction_id

                        LEFT JOIN purchase_lines AS TPL ON transactions.id=TPL.transaction_id

                        WHERE transactions.status='final' AND transactions.type='sell'
$location_filter
                        AND (TSL.variation_id=v.id OR TPL.variation_id=v.id)) as total_sold"),
                DB::raw("(SELECT SUM(IF(transactions.type='sell_transfer', TSL.quantity, 0) )
FROM transactions
                        LEFT JOIN transaction_sell_lines AS TSL ON
transactions.id=TSL.transaction_id
                        WHERE transactions.status='final' AND transactions.type='sell_transfer'
$location_filter
                        AND (TSL.variation_id=v.id)) as total_transfered"),
                DB::raw("(SELECT SUM(IF(transactions.type='stock_adjustment', SAL.quantity, 0) )
FROM transactions
                        LEFT JOIN stock_adjustment_lines AS SAL ON
transactions.id=SAL.transaction_id
                        WHERE transactions.status='received' AND
transactions.type='stock_adjustment' $location_filter
                        AND (SAL.variation_id=v.id)) as total_adjusted")

            )
                        ->groupBy('v.id')
                        ->get();

            return view('report.stock_details')
                        ->with(compact('product_details'));
        }
    }

    public function getTaxReport(Request $request)
    {
        if (!auth()->user()->can('tax_report.view')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = $request->session()->get('user.business_id');

        //Return the details in ajax call
        if ($request->ajax()) {
            $start_date = $request->get('start_date');
            $end_date = $request->get('end_date');
            // $location_id = $request->get('location_id');
            $location_id = '';
            if ($request->session()->has('business_location')) {
                $location_id = session('business_location')['id'];
            }

            $input_tax_details = $this->transactionUtil->getInputTax($business_id, $start_date,
$end_date, $location_id);

            $input_tax = view('report.partials.tax_details')->with(['tax_details' =>
$input_tax_details])->render();
```

```
            $output_tax_details = $this->transactionUtil->getOutputTax($business_id, $start_date,
$end_date, $location_id);

            $output_tax = view('report.partials.tax_details')->with(['tax_details' =>
$output_tax_details])->render();

            return ['input_tax' => $input_tax,
                    'output_tax' => $output_tax,
                    'tax_diff' => $output_tax_details['total_tax'] -
$input_tax_details['total_tax']
                ];
        }

        $business_locations = BusinessLocation::forDropdown($business_id, true);

        return view('report.tax_report')
                    ->with(compact('business_locations'));
    }

    public function getTrendingProducts(Request $request)
    {
        if (!Auth::user()->can('trending_product_report.view')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;
        $filters = $request->only(['category', 'sub_category', 'brand', 'unit', 'limit',
'location_id']);
        // $location_id = $request->input('location_id');
        $filters['location_id'] = null;
        if ($request->input('location_id')) {
            $location_id = $request->input('location_id');
            $filters['location_id'] = $request->input('location_id');
        }

        $date_range = $request->input('date_range');

        if (!empty($date_range)) {
            $date_range_array = explode('~', $date_range);
            $filters['start_date'] = $this->transactionUtil->uf_date(trim($date_range_array[0]));
            $filters['end_date'] = $this->transactionUtil->uf_date(trim($date_range_array[1]));
        }

        $products = $this->productUtil->getTrendingProducts($business_id, $filters);

        // $values = [];
        // $labels = [];
        // foreach ($products as $product) {
        //     $values[] = $product->total_unit_sold;
        //     $labels[] = $product->product . ' (' . $product->unit . ')';
        // }

        return $this->respondArray($products);
    }

    public function getExpenseReport(Request $request)
    {
        if (!Auth::user()->can('expense_report.view')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;
        $filters = $request->only(['category', 'location_id']);
        $filters['location_id'] = null;
        if ($request->input('location_id')) {
            $location_id = $request->input('location_id');
            $filters['location_id'] = $request->input('location_id');
        }

        $date_range = $request->input('date_range');

        if (!empty($date_range)) {
            $date_range_array = explode('~', $date_range);
            $filters['start_date'] = $this->transactionUtil->uf_date(trim($date_range_array[0]));
            $filters['end_date'] = $this->transactionUtil->uf_date(trim($date_range_array[1]));
        } else {
            $filters['start_date'] = \Carbon\Carbon::now()->startOfMonth()->format('Y-m-d');
            $filters['end_date'] = \Carbon\Carbon::now()->endOfMonth()->format('Y-m-d');
        }

        $expenses = $this->transactionUtil->getExpenseReport($business_id, $filters);
```

```
        return $this->respondArray($expenses);
    }

    public function getStockAdjustmentReport(Request $request)
    {
        if (!Auth::user()->can('stock_report.view')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;

        $query =  Transaction::where('business_id', $business_id)
                ->where('type', 'stock_adjustment');

        //Check for permitted locations of a user
        $permitted_locations = Auth::user()->permitted_locations();
        if ($permitted_locations != 'all') {
            $query->whereIn('location_id', $permitted_locations);
        }

        $start_date = $request->get('start_date');
        $end_date = $request->get('end_date');
        if (!empty($start_date) && !empty($end_date)) {
            $query->whereBetween(DB::raw('date(transaction_date)'), [$start_date, $end_date]);
        }

        // $location_id = $request->get('location_id');
        $location_id = null;
        if ($request->input("location_id")) {
            $request->input("location_id");
        }
        if (!empty($location_id)) {
            $query->where('location_id', $location_id);
        }

        $stock_adjustment_details = $query->select(
            DB::raw("SUM(final_total) as total_amount"),
            DB::raw("SUM(total_amount_recovered) as total_recovered"),
            DB::raw("SUM(IF(adjustment_type = 'normal', final_total, 0)) as total_normal"),
            DB::raw("SUM(IF(adjustment_type = 'abnormal', final_total, 0)) as total_abnormal")
        )->first();

        return $this->respondArray($stock_adjustment_details);
    }

    public function getRegisterReport(Request $request)
    {
        if (!Auth::user()->can('register_report.view')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }
        $business_id = Auth::user()->business_id;

        $registers = CashRegister::join(
            'users as u',
            'u.id',
            '=',
            'cash_registers.user_id'
            )
            ->where('cash_registers.business_id', $business_id)
            ->select(
                'cash_registers.*',
                DB::raw("CONCAT(COALESCE(surname, ''), ' ', COALESCE(first_name, ''), ' ',
COALESCE(last_name, ''), '<br>', COALESCE(email, '')) as user_name")
            );

        if (!empty($request->input('user_id'))) {
            $registers->where('cash_registers.user_id', $request->input('user_id'));
        }

        if (!empty($request->input('status'))) {
            $registers->where('cash_registers.status', $request->input('status'));
        }

        return $this->respondArray($registers->get());
    }

    public function getSalesRepresentativeReport(Request $request)
    {
        if (!auth()->user()->can('sales_representative.view')) {
            abort(403, 'Unauthorized action.');
```

```
        }

        $business_id = $request->session()->get('user.business_id');

        $users = User::allUsersDropdown($business_id, false);
        $business_locations = BusinessLocation::forDropdown($business_id, true);

        return view('report.sales_representative')
                ->with(compact('users', 'business_locations'));
    }

    public function getSalesRepresentativeTotalExpense(Request $request)
    {
        if (!auth()->user()->can('sales_representative.view')) {
            abort(403, 'Unauthorized action.');
        }

        if ($request->ajax()) {
            $business_id = $request->session()->get('user.business_id');

            $filters = $request->only(['expense_for', 'location_id', 'start_date', 'end_date']);

            $total_expense = $this->transactionUtil->getExpenseReport($business_id, $filters,
'total');

            return $total_expense;
        }
    }

    public function getSalesRepresentativeTotalSell(Request $request)
    {
        if (!auth()->user()->can('sales_representative.view')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = $request->session()->get('user.business_id');

        //Return the details in ajax call
        if ($request->ajax()) {
            $start_date = $request->get('start_date');
            $end_date = $request->get('end_date');

            $location_id = $request->get('location_id');
            $created_by = $request->get('created_by');

            $sell_details = $this->transactionUtil->getSellTotals($business_id, $start_date,
$end_date, $location_id, $created_by);

            //Get Sell Return details
            $transaction_types = [
                'sell_return'
            ];
            $sell_return_details = $this->transactionUtil->getTransactionTotals(
                $business_id,
                $transaction_types,
                $start_date,
                $end_date,
                $location_id,
                $created_by
            );

            $total_sell_return = !empty($sell_return_details['total_sell_return_exc_tax']) ?
$sell_return_details['total_sell_return_exc_tax'] : 0;
            $total_sell = $sell_details['total_sell_exc_tax'] - $total_sell_return;

            return [
                'total_sell_exc_tax' => $sell_details['total_sell_exc_tax'],
                'total_sell_return_exc_tax' => $total_sell_return,
                'total_sell' => $total_sell
            ];
        }
    }

    public function getSalesRepresentativeTotalCommission(Request $request)
    {
        if (!auth()->user()->can('sales_representative.view')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = $request->session()->get('user.business_id');
```

```
        //Return the details in ajax call
        if ($request->ajax()) {
            $start_date = $request->get('start_date');
            $end_date = $request->get('end_date');

            $location_id = $request->get('location_id');
            $commission_agent = $request->get('commission_agent');

            $sell_details = $this->transactionUtil->getTotalSellCommission($business_id,
$start_date, $end_date, $location_id, $commission_agent);

            //Get Commision
            $commission_percentage = User::find($commission_agent)->cmmsn_percent;
            $total_commission = $commission_percentage *
$sell_details['total_sales_with_commission'] / 100;

            return ['total_sales_with_commission' =>
                        $sell_details['total_sales_with_commission'],
                    'total_commission' => $total_commission,
                    'commission_percentage' => $commission_percentage
                ];
        }
    }

    public function getStockExpiryReport(Request $request)
    {
        if (!auth()->user()->can('stock_report.view')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = $request->session()->get('user.business_id');

        //TODO:: Need to display reference number and edit expiry date button

        //Return the details in ajax call
        if ($request->ajax()) {
            $query = PurchaseLine::leftjoin(
                'transactions as t',
                'purchase_lines.transaction_id',
                '=',
                't.id'
            )
                            ->leftjoin(
                                'products as p',
                                'purchase_lines.product_id',
                                '=',
                                'p.id'
                            )
                            ->leftjoin(
                                'variations as v',
                                'purchase_lines.variation_id',
                                '=',
                                'v.id'
                            )
                            ->leftjoin(
                                'product_variations as pv',
                                'v.product_variation_id',
                                '=',
                                'pv.id'
                            )
                            ->leftjoin('business_locations as l', 't.location_id', '=', 'l.id')
                            ->leftjoin('units as u', 'p.unit_id', '=', 'u.id')
                            ->where('t.business_id', $business_id)
                            //->whereNotNull('p.expiry_period')
                            //->whereNotNull('p.expiry_period_type')
                            //->whereNotNull('exp_date')
                            ->where('p.enable_stock', 1);
            // ->whereRaw('purchase_lines.quantity > purchase_lines.quantity_sold +
quantity_adjusted + quantity_returned');

            $permitted_locations = auth()->user()->permitted_locations();

            if ($permitted_locations != 'all') {
                $query->whereIn('t.location_id', $permitted_locations);
            }

            if (!empty($request->input('location_id'))) {
                $location_id = $request->input('location_id');
                $query->where('t.location_id', $location_id);
            }
```

```
            if (!empty($request->input('category_id'))) {
                $query->where('p.category_id', $request->input('category_id'));
            }
            if (!empty($request->input('sub_category_id'))) {
                $query->where('p.sub_category_id', $request->input('sub_category_id'));
            }
            if (!empty($request->input('brand_id'))) {
                $query->where('p.brand_id', $request->input('brand_id'));
            }
            if (!empty($request->input('unit_id'))) {
                $query->where('p.unit_id', $request->input('unit_id'));
            }
            if (!empty($request->input('exp_date_filter'))) {
                $query->whereDate('exp_date', '<=', $request->input('exp_date_filter'));
            }

            $only_mfg_products = request()->get('only_mfg_products', 0);
            if (!empty($only_mfg_products)) {
                $query->where('t.type', 'production_purchase');
            }

            $report = $query->select(
                'p.name as product',
                'p.sku',
                'p.type as product_type',
                'v.name as variation',
                'pv.name as product_variation',
                'l.name as location',
                'mfg_date',
                'exp_date',
                'u.short_name as unit',
                DB::raw("SUM(COALESCE(quantity, 0) - COALESCE(quantity_sold, 0) -
COALESCE(quantity_adjusted, 0) - COALESCE(quantity_returned, 0)) as stock_left"),
                't.ref_no',
                't.id as transaction_id',
                'purchase_lines.id as purchase_line_id',
                'purchase_lines.lot_number'
            )
            ->groupBy('purchase_lines.exp_date')
            ->groupBy('purchase_lines.lot_number');

            return Datatables::of($report)
                ->editColumn('name', function ($row) {
                    if ($row->product_type == 'variable') {
                        return $row->product . ' - ' .
                        $row->product_variation . ' - ' . $row->variation;
                    } else {
                        return $row->product;
                    }
                })
                ->editColumn('mfg_date', function ($row) {
                    if (!empty($row->mfg_date)) {
                        return $this->productUtil->format_date($row->mfg_date);
                    } else {
                        return '--';
                    }
                })
                ->editColumn('ref_no', function ($row) {
                    return '<button type="button" data-href="' .
action('PurchaseController@show', [$row->transaction_id])
                        . '" class="btn btn-link btn-modal" data-container=".view_modal"  >'
. $row->ref_no . '</button>';
                })
                ->editColumn('stock_left', function ($row) {
                    return '<span data-is_quantity="true" class="display_currency stock_left"
data-currency_symbol=false data-orig-value="' . $row->stock_left . '" data-unit="' . $row->unit .
'" >' . $row->stock_left . '</span> ' . $row->unit;
                })
                ->addColumn('edit', function ($row) {
                    $html =  '<button type="button" class="btn btn-primary btn-xs
stock_expiry_edit_btn" data-transaction_id="' . $row->transaction_id . '" data-
purchase_line_id="' . $row->purchase_line_id . '"> <i class="fa fa-edit"></i> ' .
__("messages.edit") .
                    '</button>';

                    if (!empty($row->exp_date)) {
                        $carbon_exp = \Carbon::createFromFormat('Y-m-d', $row->exp_date);
                        $carbon_now = \Carbon::now();
                        if ($carbon_now->diffInDays($carbon_exp, false) < 0) {
```

```
                            $html .=  ' <button type="button" class="btn btn-warning btn-xs
remove_from_stock_btn" data-href="' . action('StockAdjustmentController@removeExpiredStock',
[$row->purchase_line_id]) . '"> <i class="fa fa-trash"></i> ' . __("lang_v1.remove_from_stock") .
                                '</button>';
                            }
                        }

                        return $html;
                    })
                    ->rawColumns(['exp_date', 'ref_no', 'edit', 'stock_left'])
                    ->make(true);
        }

        $categories = Category::where('business_id', $business_id)
                            ->where('parent_id', 0)
                            ->pluck('name', 'id');
        $brands = Brands::where('business_id', $business_id)
                            ->pluck('name', 'id');
        $units = Unit::where('business_id', $business_id)
                            ->pluck('short_name', 'id');
        $business_locations = BusinessLocation::forDropdown($business_id, true);
        $view_stock_filter = [
            \Carbon::now()->subDay()->format('Y-m-d') => __('report.expired'),
            \Carbon::now()->addWeek()->format('Y-m-d') => __('report.expiring_in_1_week'),
            \Carbon::now()->addDays(15)->format('Y-m-d') => __('report.expiring_in_15_days'),
            \Carbon::now()->addMonth()->format('Y-m-d') => __('report.expiring_in_1_month'),
            \Carbon::now()->addMonths(3)->format('Y-m-d') => __('report.expiring_in_3_months'),
            \Carbon::now()->addMonths(6)->format('Y-m-d') => __('report.expiring_in_6_months'),
            \Carbon::now()->addYear()->format('Y-m-d') => __('report.expiring_in_1_year')
        ];

        return view('report.stock_expiry_report')
                ->with(compact('categories', 'brands', 'units', 'business_locations',
'view_stock_filter'));
    }

    public function getStockExpiryReportEditModal(Request $request, $purchase_line_id)
    {
        if (!auth()->user()->can('stock_report.view')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = $request->session()->get('user.business_id');

        //Return the details in ajax call
        if ($request->ajax()) {
            $purchase_line = PurchaseLine::join(
                'transactions as t',
                'purchase_lines.transaction_id',
                '=',
                't.id'
            )
                                ->join(
                                    'products as p',
                                    'purchase_lines.product_id',
                                    '=',
                                    'p.id'
                                )
                                ->where('purchase_lines.id', $purchase_line_id)
                                ->where('t.business_id', $business_id)
                                ->select(['purchase_lines.*', 'p.name', 't.ref_no'])
                                ->first();

            if (!empty($purchase_line)) {
                if (!empty($purchase_line->exp_date)) {
                    $purchase_line->exp_date = date('m/d/Y', strtotime($purchase_line-
>exp_date));
                }
            }

            return view('report.partials.stock_expiry_edit_modal')
                ->with(compact('purchase_line'));
        }
    }

    public function updateStockExpiryReport(Request $request)
    {
        if (!auth()->user()->can('stock_report.view')) {
            abort(403, 'Unauthorized action.');
        }
```

```
        try {
            $business_id = $request->session()->get('user.business_id');

            //Return the details in ajax call
            if ($request->ajax()) {
                DB::beginTransaction();

                $input = $request->only(['purchase_line_id', 'exp_date']);

                $purchase_line = PurchaseLine::join(
                    'transactions as t',
                    'purchase_lines.transaction_id',
                    '=',
                    't.id'
                )
                                    ->join(
                                        'products as p',
                                        'purchase_lines.product_id',
                                        '=',
                                        'p.id'
                                    )
                                    ->where('purchase_lines.id', $input['purchase_line_id'])
                                    ->where('t.business_id', $business_id)
                                    ->select(['purchase_lines.*', 'p.name', 't.ref_no'])
                                    ->first();

                if (!empty($purchase_line) && !empty($input['exp_date'])) {
                    $purchase_line->exp_date = $this->productUtil->uf_date($input['exp_date']);
                    $purchase_line->save();
                }

                DB::commit();

                $output = ['success' => 1,
                            'msg' => __('lang_v1.updated_succesfully')
                        ];
            }
        } catch (\Exception $e) {
            DB::rollBack();
            \Log::emergency("File:" . $e->getFile(). "Line:" . $e->getLine(). "Message:" . $e-
>getMessage());

            $output = ['success' => 0,
                            'msg' => __('messages.something_went_wrong')
                        ];
        }

        return $output;
    }

    public function getCustomerGroup(Request $request)
    {
        if (!auth()->user()->can('contacts_report.view')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = $request->session()->get('user.business_id');

        if ($request->ajax()) {
            $query = Transaction::leftjoin('customer_groups AS CG',
'transactions.customer_group_id', '=', 'CG.id')
                        ->where('transactions.business_id', $business_id)
                        ->where('transactions.type', 'sell')
                        ->where('transactions.status', 'final')
                        ->groupBy('transactions.customer_group_id')
                        ->select(DB::raw("SUM(final_total) as total_sell"), 'CG.name');

            $group_id = $request->get('customer_group_id', null);
            if (!empty($group_id)) {
                $query->where('transactions.customer_group_id', $group_id);
            }

            $permitted_locations = auth()->user()->permitted_locations();
            if ($permitted_locations != 'all') {
                $query->whereIn('transactions.location_id', $permitted_locations);
            }

            // $location_id = $request->get('location_id', null);
            $location_id = null;
            if ($request->session()->has('business_location')) {
                $location_id = session('business_location')['id'];
```

```
            }
            if (!empty($location_id)) {
                $query->where('transactions.location_id', $location_id);
            }

            $start_date = $request->get('start_date');
            $end_date = $request->get('end_date');

            if (!empty($start_date) && !empty($end_date)) {
                $query->whereBetween(DB::raw('date(transaction_date)'), [$start_date,
$end_date]);
            }


            return Datatables::of($query)
                ->editColumn('total_sell', function ($row) {
                    return '<span class="display_currency" data-currency_symbol = true>' . $row-
>total_sell . '</span>';
                })
                ->rawColumns(['total_sell'])
                ->make(true);
        }

        $customer_group = CustomerGroup::forDropdown($business_id, false, true);
        $business_locations = BusinessLocation::forDropdown($business_id, true);

        return view('report.customer_group')
            ->with(compact('customer_group', 'business_locations'));
    }

    public function getproductPurchaseReport(Request $request)
    {
        if (!Auth::user()->can('purchase_n_sell_report.view')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;
        $variation_id = $request->get('variation_id', null);
        $query = PurchaseLine::join(
            'transactions as t',
            'purchase_lines.transaction_id',
            '=',
            't.id'
                )
                ->join(
                    'variations as v',
                    'purchase_lines.variation_id',
                    '=',
                    'v.id'
                )
                ->join('product_variations as pv', 'v.product_variation_id', '=', 'pv.id')
                ->join('contacts as c', 't.contact_id', '=', 'c.id')
                ->join('products as p', 'pv.product_id', '=', 'p.id')
                ->leftjoin('units as u', 'p.unit_id', '=', 'u.id')
                ->where('t.business_id', $business_id)
                ->where('t.type', 'purchase')
                ->select(
                    'p.name as product_name',
                    'p.type as product_type',
                    'pv.name as product_variation',
                    'v.name as variation_name',
                    'c.name as supplier',
                    't.id as transaction_id',
                    't.ref_no',
                    't.transaction_date as transaction_date',
                    'purchase_lines.purchase_price_inc_tax as unit_purchase_price',
                    DB::raw('(purchase_lines.quantity - purchase_lines.quantity_returned) as
purchase_qty'),
                    'purchase_lines.quantity_adjusted',
                    'u.short_name as unit',
                    DB::raw('((purchase_lines.quantity - purchase_lines.quantity_returned -
purchase_lines.quantity_adjusted) * purchase_lines.purchase_price_inc_tax) as subtotal')
                )
                ->groupBy('purchase_lines.id');

        if (!empty($variation_id)) {
            $query->where('purchase_lines.variation_id', $variation_id);
        }

        $start_date = $request->get('start_date');
        $end_date = $request->get('end_date');
```

```
        if (!empty($start_date) && !empty($end_date)) {
            $query->whereBetween(DB::raw('date(transaction_date)'), [$start_date, $end_date]);
        }

        $permitted_locations = Auth::user()->permitted_locations();
        if ($permitted_locations != 'all') {
            $query->whereIn('t.location_id', $permitted_locations);
        }

        // $location_id = $request->get('location_id', null);
        $location_id = null;
        if ($request->input('location_id')) {
            $location_id = $request->input('location_id');
        }
        if (!empty($location_id)) {
            $query->where('t.location_id', $location_id);
        }

        $supplier_id = $request->get('supplier_id', null);
        if (!empty($supplier_id)) {
            $query->where('t.contact_id', $supplier_id);
        }

        return $this->respondArray($query->get());
    }

    public function getproductSellReport(Request $request)
    {
        if (!Auth::user()->can('purchase_n_sell_report.view')) {
            return $this->respondFailed("Tidak diizinkan");
        }

        $business_id = Auth::user()->business_id;
        $variation_id = $request->get('variation_id', null);
        $query = TransactionSellLine::join(
            'transactions as t',
            'transaction_sell_lines.transaction_id',
            '=',
            't.id'
        )
            ->join(
                'variations as v',
                'transaction_sell_lines.variation_id',
                '=',
                'v.id'
            )
            ->join('product_variations as pv', 'v.product_variation_id', '=', 'pv.id')
            ->join('contacts as c', 't.contact_id', '=', 'c.id')
            ->join('products as p', 'pv.product_id', '=', 'p.id')
            ->leftjoin('tax_rates', 'transaction_sell_lines.tax_id', '=', 'tax_rates.id')
            ->leftjoin('units as u', 'p.unit_id', '=', 'u.id')
            ->where('t.business_id', $business_id)
            ->where('t.type', 'sell')
            ->where('t.status', 'final')
            ->select(
                'p.name as product_name',
                'p.type as product_type',
                'pv.name as product_variation',
                'v.name as variation_name',
                'c.name as customer',
                't.id as transaction_id',
                't.invoice_no',
                't.transaction_date as transaction_date',
                'transaction_sell_lines.unit_price_before_discount as unit_price',
                DB::raw('(transaction_sell_lines.quantity -
transaction_sell_lines.quantity_returned) as sell_qty'),
                'transaction_sell_lines.line_discount_type as discount_type',
                'transaction_sell_lines.line_discount_amount as discount_amount',
                'transaction_sell_lines.item_tax',
                'u.short_name as unit',
                DB::raw('((transaction_sell_lines.quantity -
transaction_sell_lines.quantity_returned) * transaction_sell_lines.unit_price_inc_tax) as
subtotal')
            )
            ->groupBy('transaction_sell_lines.id');

        if (!empty($variation_id)) {
            $query->where('transaction_sell_lines.variation_id', $variation_id);
        }
        $start_date = $request->get('start_date');
        $end_date = $request->get('end_date');
```

```
        if (!empty($start_date) && !empty($end_date)) {
            $query->whereBetween(DB::raw('date(transaction_date)'), [$start_date, $end_date]);
        }

        $permitted_locations = Auth::user()->permitted_locations();
        if ($permitted_locations != 'all') {
            $query->whereIn('t.location_id', $permitted_locations);
        }
        // $location_id = $request->get('location_id', null);
        $location_id = null;
        if ($request->input('location_id')) {
            $location_id = $request->input('location_id');
        }
        if (!empty($location_id)) {
            $query->where('t.location_id', $location_id);
        }

        $customer_id = $request->get('customer_id', null);
        if (!empty($customer_id)) {
            $query->where('t.contact_id', $customer_id);
        }

        return $this->respondArray($query->get());
    }

    public function getLotReport(Request $request)
    {
        if (!auth()->user()->can('stock_report.view')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = $request->session()->get('user.business_id');

        //Return the details in ajax call
        if ($request->ajax()) {
            $query = Product::where('products.business_id', $business_id)
                    ->leftjoin('units', 'products.unit_id', '=', 'units.id')
                    ->join('variations as v', 'products.id', '=', 'v.product_id')
                    ->join('purchase_lines as pl', 'v.id', '=', 'pl.variation_id')
                    ->leftjoin(
                        'transaction_sell_lines_purchase_lines as tspl',
                        'pl.id',
                        '=',
                        'tspl.purchase_line_id'
                    )
                    ->join('transactions as t', 'pl.transaction_id', '=', 't.id');

            $permitted_locations = auth()->user()->permitted_locations();
            $location_filter = 'WHERE ';

            if ($permitted_locations != 'all') {
                $query->whereIn('t.location_id', $permitted_locations);

                $locations_imploded = implode(', ', $permitted_locations);
                $location_filter = " LEFT JOIN transactions as t2 on pls.transaction_id=t2.id
WHERE t2.location_id IN ($locations_imploded) AND ";
            }

            // $location_id = $request->input('location_id');
            $location_id = null;
            if ($request->session()->has('business_location')) {
                $location_id = session('business_location')['id'];
            }
            if (!empty($location_id)) {
                $query->where('t.location_id', $location_id);

                $location_filter = "LEFT JOIN transactions as t2 on pls.transaction_id=t2.id
WHERE t2.location_id=$location_id AND ";
            }

            if (!empty($request->input('category_id'))) {
                $query->where('products.category_id', $request->input('category_id'));
            }

            if (!empty($request->input('sub_category_id'))) {
                $query->where('products.sub_category_id', $request->input('sub_category_id'));
            }

            if (!empty($request->input('brand_id'))) {
                $query->where('products.brand_id', $request->input('brand_id'));
            }
```

```
            if (!empty($request->input('unit_id'))) {
                $query->where('products.unit_id', $request->input('unit_id'));
            }

            $only_mfg_products = request()->get('only_mfg_products', 0);
            if (!empty($only_mfg_products)) {
                $query->where('t.type', 'production_purchase');
            }

            $products = $query->select(
                'products.name as product',
                'v.name as variation_name',
                'sub_sku',
                'pl.lot_number',
                'pl.exp_date as exp_date',
                DB::raw("( COALESCE((SELECT SUM(quantity - quantity_returned) from purchase_lines
as pls $location_filter variation_id = v.id AND lot_number = pl.lot_number), 0) -
                    SUM(COALESCE((tspl.quantity - tspl.qty_returned), 0))) as stock"),
                DB::raw("COALESCE(SUM(IF(tspl.sell_line_id IS NULL, 0, (tspl.quantity -
tspl.qty_returned)) ), 0) as total_sold"),
                DB::raw("COALESCE(SUM(IF(tspl.stock_adjustment_line_id IS NULL, 0, tspl.quantity
) ), 0) as total_adjusted"),
                'products.type',
                'units.short_name as unit'
            )
            ->whereNotNull('pl.lot_number')
            ->groupBy('v.id')
            ->groupBy('pl.lot_number');

            return Datatables::of($products)
                ->editColumn('stock', function ($row) {
                    $stock = $row->stock ? $row->stock : 0 ;
                    return '<span data-is_quantity="true" class="display_currency total_stock"
data-currency_symbol=false data-orig-value="' . (float)$stock . '" data-unit="' . $row->unit . '"
>' . (float)$stock . '</span> ' . $row->unit;
                })
                ->editColumn('product', function ($row) {
                    if ($row->variation_name != 'DUMMY') {
                        return $row->product . ' (' . $row->variation_name . ')';
                    } else {
                        return $row->product;
                    }
                })
                ->editColumn('total_sold', function ($row) {
                    if ($row->total_sold) {
                        return '<span data-is_quantity="true" class="display_currency total_sold"
data-currency_symbol=false data-orig-value="' . (float)$row->total_sold . '" data-unit="' . $row-
>unit . '" >' . (float)$row->total_sold . '</span> ' . $row->unit;
                    } else {
                        return '0' . ' ' . $row->unit;
                    }
                })
                ->editColumn('total_adjusted', function ($row) {
                    if ($row->total_adjusted) {
                        return '<span data-is_quantity="true" class="display_currency
total_adjusted" data-currency_symbol=false data-orig-value="' . (float)$row->total_adjusted . '"
data-unit="' . $row->unit . '" >' . (float)$row->total_adjusted . '</span> ' . $row->unit;
                    } else {
                        return '0' . ' ' . $row->unit;
                    }
                })
                ->editColumn('exp_date', function ($row) {
                    if (!empty($row->exp_date)) {
                        $carbon_exp = \Carbon::createFromFormat('Y-m-d', $row->exp_date);
                        $carbon_now = \Carbon::now();
                        if ($carbon_now->diffInDays($carbon_exp, false) >= 0) {
                            return $this->productUtil->format_date($row->exp_date) .
'<br><small>( <span class="time-to-now">' . $row->exp_date . '</span> )</small>';
                        } else {
                            return $this->productUtil->format_date($row->exp_date) . '   <span
class="label label-danger no-print">' . __('report.expired') . '</span><span
class="print_section">' . __('report.expired') . '</span><br><small>( <span class="time-from-
now">' . $row->exp_date . '</span> )</small>';
                        }
                    } else {
                        return '--';
                    }
                })
                ->removeColumn('unit')
                ->removeColumn('id')
```

```
                ->removeColumn('variation_name')
                ->rawColumns(['exp_date', 'stock', 'total_sold', 'total_adjusted'])
                ->make(true);
        }

        $categories = Category::where('business_id', $business_id)
                            ->where('parent_id', 0)
                            ->pluck('name', 'id');
        $brands = Brands::where('business_id', $business_id)
                            ->pluck('name', 'id');
        $units = Unit::where('business_id', $business_id)
                            ->pluck('short_name', 'id');
        $business_locations = BusinessLocation::forDropdown($business_id, true);

        return view('report.lot_report')
            ->with(compact('categories', 'brands', 'units', 'business_locations'));
    }

    public function purchasePaymentReport(Request $request)
    {
        if (!Auth::user()->can('purchase_n_sell_report.view')) {
            return $this->respondFailed("Tidak diizinkan");
        }

        $business_id = Auth::user()->business_id;
        $supplier_id = $request->get('supplier_id', null);
        $contact_filter1 = !empty($supplier_id) ? "AND t.contact_id=$supplier_id" : '';
        $contact_filter2 = !empty($supplier_id) ? "AND transactions.contact_id=$supplier_id" :
'';

        // $location_id = $request->get('location_id', null);
        $location_id = null;
        if ($request->input('location_id')) {
            $location_id = $request->input('location_id');
        }
        $parent_payment_query_part = empty($location_id) ? "AND transaction_payments.parent_id IS
NULL" : "";

        $query = TransactionPayment::leftjoin('transactions as t', function ($join) use
($business_id) {
            $join->on('transaction_payments.transaction_id', '=', 't.id')
                ->where('t.business_id', $business_id)
                ->whereIn('t.type', ['purchase', 'opening_balance']);
        })
            ->where('transaction_payments.business_id', $business_id)
            ->where(function ($q) use ($business_id, $contact_filter1, $contact_filter2,
$parent_payment_query_part) {
                $q->whereRaw("(transaction_payments.transaction_id IS NOT NULL AND t.type IN
('purchase', 'opening_balance')  $parent_payment_query_part $contact_filter1)")
                    ->orWhereRaw("EXISTS(SELECT * FROM transaction_payments as tp JOIN
transactions ON tp.transaction_id = transactions.id WHERE transactions.type IN ('purchase',
'opening_balance') AND transactions.business_id = $business_id AND
tp.parent_id=transaction_payments.id $contact_filter2)");
            })

            ->select(
                DB::raw("IF(transaction_payments.transaction_id IS NULL,
                            (SELECT c.name FROM transactions as ts
                            JOIN contacts as c ON ts.contact_id=c.id
                            WHERE ts.id=(
                                    SELECT tps.transaction_id FROM transaction_payments as tps
                                    WHERE tps.parent_id=transaction_payments.id LIMIT 1
                                )
                            ),
                            (SELECT c.name FROM transactions as ts JOIN
                                contacts as c ON ts.contact_id=c.id
                                WHERE ts.id=t.id
                            )
                        ) as supplier"),
                'transaction_payments.amount',
                'method',
                'paid_on',
                'transaction_payments.payment_ref_no',
                'transaction_payments.document',
                't.ref_no',
                't.id as transaction_id',
                'cheque_number',
                'card_transaction_number',
                'bank_account_number',
                'transaction_no',
                'transaction_payments.id as DT_RowId'
```

```
            )
            ->groupBy('transaction_payments.id');

        $start_date = $request->get('start_date');
        $end_date = $request->get('end_date');
        if (!empty($start_date) && !empty($end_date)) {
            $query->whereBetween(DB::raw('date(paid_on)'), [$start_date, $end_date]);
        }

        $permitted_locations = Auth::user()->permitted_locations();
        if ($permitted_locations != 'all') {
            $query->whereIn('t.location_id', $permitted_locations);
        }

        if (!empty($location_id)) {
            $query->where('t.location_id', $location_id);
        }

        return $this->respondArray($query->get());
    }

    public function sellPaymentReport(Request $request)
    {
        if (!Auth::user()->can('purchase_n_sell_report.view')) {
            return $this->respondFailed("Tidak diizinkan", 403);
        }

        $business_id = Auth::user()->business_id;
        $customer_id = $request->get('supplier_id', null);
        $contact_filter1 = !empty($customer_id) ? "AND t.contact_id=$customer_id" : '';
        $contact_filter2 = !empty($customer_id) ? "AND transactions.contact_id=$customer_id" :
'';

        // $location_id = $request->get('location_id', null);
        $location_id = null;
        if ($request->input('location_id')) {
            $location_id = $request->input('location_id');
        }
        $parent_payment_query_part = empty($location_id) ? "AND transaction_payments.parent_id IS
NULL" : "";

        $query = TransactionPayment::leftjoin('transactions as t', function ($join) use
($business_id) {
            $join->on('transaction_payments.transaction_id', '=', 't.id')
                ->where('t.business_id', $business_id)
                ->whereIn('t.type', ['sell', 'opening_balance']);
        })
            ->leftjoin('contacts as c', 't.contact_id', '=', 'c.id')
            ->where('transaction_payments.business_id', $business_id)
            ->where(function ($q) use ($business_id, $contact_filter1, $contact_filter2,
$parent_payment_query_part) {
                $q->whereRaw("(transaction_payments.transaction_id IS NOT NULL AND t.type IN
('sell', 'opening_balance') $parent_payment_query_part $contact_filter1)")
                    ->orWhereRaw("EXISTS(SELECT * FROM transaction_payments as tp JOIN
transactions ON tp.transaction_id = transactions.id WHERE transactions.type IN ('sell',
'opening_balance') AND transactions.business_id = $business_id AND
tp.parent_id=transaction_payments.id $contact_filter2)");
            })
            ->select(
                DB::raw("IF(transaction_payments.transaction_id IS NULL,
                            (SELECT c.name FROM transactions as ts
                            JOIN contacts as c ON ts.contact_id=c.id
                            WHERE ts.id=(
                                    SELECT tps.transaction_id FROM transaction_payments as tps
                                    WHERE tps.parent_id=transaction_payments.id LIMIT 1
                                )
                            ),
                            (SELECT c.name FROM transactions as ts JOIN
                                contacts as c ON ts.contact_id=c.id
                                WHERE ts.id=t.id
                            )
                        ) as customer"),
                'transaction_payments.amount',
                'method',
                'paid_on',
                'transaction_payments.payment_ref_no',
                'transaction_payments.document',
                'transaction_payments.transaction_no',
                't.invoice_no',
                't.id as transaction_id',
                'cheque_number',
```

```
                'card_transaction_number',
                'bank_account_number',
                'transaction_payments.id as DT_RowId'
            )
            ->groupBy('transaction_payments.id');

        $start_date = $request->get('start_date');
        $end_date = $request->get('end_date');
        if (!empty($start_date) && !empty($end_date)) {
            $query->whereBetween(DB::raw('date(paid_on)'), [$start_date, $end_date]);
        }

        $permitted_locations = Auth::user()->permitted_locations();
        if ($permitted_locations != 'all') {
            $query->whereIn('t.location_id', $permitted_locations);
        }

        if (!empty($location_id)) {
            $query->where('t.location_id', $location_id);
        }

        if (!empty($request->get('payment_types'))) {
            $query->where('transaction_payments.method', $request->get('payment_types'));
        }

        return $this->respondArray($query->get());
    }

    public function getTableReport(Request $request)
    {
        if (!auth()->user()->can('purchase_n_sell_report.view')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = $request->session()->get('user.business_id');

        if ($request->ajax()) {
            $query = ResTable::leftjoin('transactions AS T', 'T.res_table_id', '=',
'res_tables.id')
                        ->where('T.business_id', $business_id)
                        ->where('T.type', 'sell')
                        ->where('T.status', 'final')
                        ->groupBy('res_tables.id')
                        ->select(DB::raw("SUM(final_total) as total_sell"), 'res_tables.name as
table');

            $location_id = $request->get('location_id', null);
            if (!empty($location_id)) {
                $query->where('T.location_id', $location_id);
            }

            $start_date = $request->get('start_date');
            $end_date = $request->get('end_date');

            if (!empty($start_date) && !empty($end_date)) {
                $query->whereBetween(DB::raw('date(transaction_date)'), [$start_date,
$end_date]);
            }

            return Datatables::of($query)
                ->editColumn('total_sell', function ($row) {
                    return '<span class="display_currency" data-currency_symbol="true">' . $row-
>total_sell . '</span>';
                })
                ->rawColumns(['total_sell'])
                ->make(true);
        }

        $business_locations = BusinessLocation::forDropdown($business_id, true);

        return view('report.table_report')
            ->with(compact('business_locations'));
    }

    public function getServiceStaffReport(Request $request)
    {
        if (!auth()->user()->can('sales_representative.view')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = $request->session()->get('user.business_id');
```

```
        $business_locations = BusinessLocation::forDropdown($business_id, true);

        $waiters = $this->transactionUtil->serviceStaffDropdown($business_id);

        return view('report.service_staff_report')
            ->with(compact('business_locations', 'waiters'));
    }

    public function getproductSellGroupedReport(Request $request)
    {
        if (!auth()->user()->can('purchase_n_sell_report.view')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = $request->session()->get('user.business_id');
        $location_id = $request->get('location_id', null);

        $vld_str = '';
        if (!empty($location_id)) {
            $vld_str = "AND vld.location_id=$location_id";
        }

        if ($request->ajax()) {
            $variation_id = $request->get('variation_id', null);
            $query = TransactionSellLine::join(
                'transactions as t',
                'transaction_sell_lines.transaction_id',
                '=',
                't.id'
            )
                ->join(
                    'variations as v',
                    'transaction_sell_lines.variation_id',
                    '=',
                    'v.id'
                )
                ->join('product_variations as pv', 'v.product_variation_id', '=', 'pv.id')
                ->join('products as p', 'pv.product_id', '=', 'p.id')
                ->leftjoin('units as u', 'p.unit_id', '=', 'u.id')
                ->where('t.business_id', $business_id)
                ->where('t.type', 'sell')
                ->where('t.status', 'final')
                ->select(
                    'p.name as product_name',
                    'p.enable_stock',
                    'p.type as product_type',
                    'pv.name as product_variation',
                    'v.name as variation_name',
                    't.id as transaction_id',
                    't.transaction_date as transaction_date',
                    DB::raw('DATE_FORMAT(t.transaction_date, "%Y-%m-%d") as formated_date'),
                    DB::raw("(SELECT SUM(vld.qty_available) FROM variation_location_details as
vld WHERE vld.variation_id=v.id $vld_str) as current_stock"),
                    DB::raw('SUM(transaction_sell_lines.quantity -
transaction_sell_lines.quantity_returned) as total_qty_sold'),
                    'u.short_name as unit',
                    DB::raw('SUM((transaction_sell_lines.quantity -
transaction_sell_lines.quantity_returned) * transaction_sell_lines.unit_price_inc_tax) as
subtotal')
                )
                ->groupBy('v.id')
                ->groupBy('formated_date');

            if (!empty($variation_id)) {
                $query->where('transaction_sell_lines.variation_id', $variation_id);
            }
            $start_date = $request->get('start_date');
            $end_date = $request->get('end_date');
            if (!empty($start_date) && !empty($end_date)) {
                $query->whereBetween(DB::raw('date(transaction_date)'), [$start_date,
$end_date]);
            }

            $permitted_locations = auth()->user()->permitted_locations();
            if ($permitted_locations != 'all') {
                $query->whereIn('t.location_id', $permitted_locations);
            }

            if (!empty($location_id)) {
                $query->where('t.location_id', $location_id);
```

```
            }

            $customer_id = $request->get('customer_id', null);
            if (!empty($customer_id)) {
                $query->where('t.contact_id', $customer_id);
            }

            return Datatables::of($query)
                ->editColumn('product_name', function ($row) {
                    $product_name = $row->product_name;
                    if ($row->product_type == 'variable') {
                        $product_name .= ' - ' . $row->product_variation . ' - ' . $row-
>variation_name;
                    }

                    return $product_name;
                })
                ->editColumn('transaction_date', '{{@format_date($formated_date)}}')
                ->editColumn('total_qty_sold', function ($row) {
                    return '<span data-is_quantity="true" class="display_currency sell_qty" data-
currency_symbol=false data-orig-value="' . (float)$row->total_qty_sold . '" data-unit="' . $row-
>unit . '" >' . (float) $row->total_qty_sold . '</span> ' .$row->unit;
                    })
                ->editColumn('current_stock', function ($row) {
                    if ($row->enable_stock) {
                        return '<span data-is_quantity="true" class="display_currency
current_stock" data-currency_symbol=false data-orig-value="' . (float)$row->current_stock . '"
data-unit="' . $row->unit . '" >' . (float) $row->current_stock . '</span> ' .$row->unit;
                    } else {
                        return '';
                    }
                })
                 ->editColumn('subtotal', function ($row) {
                     return '<span class="display_currency row_subtotal" data-currency_symbol =
true data-orig-value="' . $row->subtotal . '">' . $row->subtotal . '</span>';
                 })

                ->rawColumns(['current_stock', 'subtotal', 'total_qty_sold'])
                ->make(true);
        }
    }

    public function productStockDetails()
    {
        if (!auth()->user()->can('report.stock_details')) {
            abort(403, 'Unauthorized action.');
        }

        $business_id = request()->session()->get('user.business_id');

        $stock_details = [];
        $location = null;
        $total_stock_calculated = 0;
        if (!empty(request()->input('location_id'))) {
            $variation_id = request()->get('variation_id', null);
            $location_id = request()->input('location_id');

            $location = BusinessLocation::where('business_id', $business_id)
                                        ->where('id', $location_id)
                                        ->first();

            $query = Variation::leftjoin('products as p', 'p.id', '=', 'variations.product_id')
                    ->leftjoin('units', 'p.unit_id', '=', 'units.id')
                    ->leftjoin('variation_location_details as vld', 'variations.id', '=',
'vld.variation_id')
                    ->leftjoin('product_variations as pv', 'variations.product_variation_id',
'=', 'pv.id')
                    ->where('p.business_id', $business_id)
                    ->where('vld.location_id', $location_id);
            if (!is_null($variation_id)) {
                $query->where('variations.id', $variation_id);
            }

            $stock_details = $query->select(
                DB::raw("(SELECT SUM(COALESCE(TSL.quantity, 0)) FROM transactions
                    LEFT JOIN transaction_sell_lines AS TSL ON
transactions.id=TSL.transaction_id
                    WHERE transactions.status='final' AND transactions.type='sell' AND
transactions.location_id=$location_id
                    AND TSL.variation_id=variations.id) as total_sold"),
                DB::raw("(SELECT SUM(COALESCE(TSL.quantity_returned, 0)) FROM transactions
```

```
                        LEFT JOIN transaction_sell_lines AS TSL ON
transactions.id=TSL.transaction_id
                        WHERE transactions.status='final' AND transactions.type='sell' AND
transactions.location_id=$location_id
                        AND TSL.variation_id=variations.id) as total_sell_return"),
                DB::raw("(SELECT SUM(COALESCE(TSL.quantity,0)) FROM transactions
                        LEFT JOIN transaction_sell_lines AS TSL ON
transactions.id=TSL.transaction_id
                        WHERE transactions.status='final' AND transactions.type='sell_transfer'
AND transactions.location_id=$location_id
                        AND TSL.variation_id=variations.id) as total_sell_transfered"),
                DB::raw("(SELECT SUM(COALESCE(PL.quantity,0)) FROM transactions
                        LEFT JOIN purchase_lines AS PL ON transactions.id=PL.transaction_id
                        WHERE transactions.status='received' AND
transactions.type='purchase_transfer' AND transactions.location_id=$location_id
                        AND PL.variation_id=variations.id) as total_purchase_transfered"),
                DB::raw("(SELECT SUM(COALESCE(SAL.quantity, 0)) FROM transactions
                        LEFT JOIN stock_adjustment_lines AS SAL ON
transactions.id=SAL.transaction_id
                        WHERE transactions.status='received' AND
transactions.type='stock_adjustment' AND transactions.location_id=$location_id
                        AND SAL.variation_id=variations.id) as total_adjusted"),
                DB::raw("(SELECT SUM(COALESCE(PL.quantity, 0)) FROM transactions
                        LEFT JOIN purchase_lines AS PL ON transactions.id=PL.transaction_id
                        WHERE transactions.status='received' AND transactions.type='purchase' AND
transactions.location_id=$location_id
                        AND PL.variation_id=variations.id) as total_purchased"),
                DB::raw("(SELECT SUM(COALESCE(PL.quantity_returned, 0)) FROM transactions
                        LEFT JOIN purchase_lines AS PL ON transactions.id=PL.transaction_id
                        WHERE transactions.status='received' AND transactions.type='purchase' AND
transactions.location_id=$location_id
                        AND PL.variation_id=variations.id) as total_purchase_return"),
                DB::raw("(SELECT SUM(COALESCE(PL.quantity, 0)) FROM transactions
                        LEFT JOIN purchase_lines AS PL ON transactions.id=PL.transaction_id
                        WHERE transactions.status='received' AND
transactions.type='opening_stock' AND transactions.location_id=$location_id
                        AND PL.variation_id=variations.id) as total_opening_stock"),
                DB::raw("(SELECT SUM(COALESCE(PL.quantity, 0)) FROM transactions
                        LEFT JOIN purchase_lines AS PL ON transactions.id=PL.transaction_id
                        WHERE transactions.status='received' AND
transactions.type='production_purchase' AND transactions.location_id=$location_id
                        AND PL.variation_id=variations.id) as total_manufactured"),
                DB::raw("(SELECT SUM(COALESCE(TSL.quantity, 0)) FROM transactions
                        LEFT JOIN transaction_sell_lines AS TSL ON
transactions.id=TSL.transaction_id
                        WHERE transactions.status='final' AND transactions.type='production_sell'
AND transactions.location_id=$location_id
                        AND TSL.variation_id=variations.id) as total_ingredients_used"),
                DB::raw("SUM(vld.qty_available) as stock"),
                'variations.sub_sku as sub_sku',
                'p.name as product',
                'p.id as product_id',
                'p.type',
                'p.sku as sku',
                'units.short_name as unit',
                'p.enable_stock as enable_stock',
                'variations.sell_price_inc_tax as unit_price',
                'pv.name as product_variation',
                'variations.name as variation_name',
                'variations.id as variation_id'
            )
            ->groupBy('variations.id')
            ->get();

            foreach ($stock_details as $index => $row) {
                $total_sold = $row->total_sold ?: 0;
                $total_sell_return = $row->total_sell_return ?: 0;
                $total_sell_transfered = $row->total_sell_transfered ?: 0;

                $total_purchase_transfered = $row->total_purchase_transfered ?: 0;
                $total_adjusted = $row->total_adjusted ?: 0;
                $total_purchased = $row->total_purchased ?: 0;
                $total_purchase_return = $row->total_purchase_return ?: 0;
                $total_opening_stock = $row->total_opening_stock ?: 0;
                $total_manufactured = $row->total_manufactured ?: 0;
                $total_ingredients_used = $row->total_ingredients_used ?: 0;

                $total_stock_calculated = $total_opening_stock + $total_purchased +
$total_purchase_transfered + $total_sell_return + $total_manufactured
                - ($total_sold + $total_sell_transfered + $total_adjusted +
$total_purchase_return + $total_ingredients_used);
```

```
                $stock_details[$index]->total_stock_calculated = $total_stock_calculated;
            }
        }

        $business_locations = BusinessLocation::forDropdown($business_id);
        return view('report.product_stock_details')
            ->with(compact('stock_details', 'business_locations', 'location'));
    }

    public function adjustProductStock()
    {
        if (!auth()->user()->can('report.stock_details')) {
            abort(403, 'Unauthorized action.');
        }

        if (!empty(request()->input('variation_id'))
            && !empty(request()->input('location_id'))
            && !empty(request()->input('stock'))) {
            $business_id = request()->session()->get('user.business_id');

            $vld = VariationLocationDetails::leftjoin(
                'business_locations as bl',
                'bl.id',
                '=',
                'variation_location_details.location_id'
            )
                    ->where('variation_location_details.location_id', request()-
>input('location_id'))
                        ->where('variation_id', request()->input('variation_id'))
                        ->where('bl.business_id', $business_id)
                        ->select('variation_location_details.*')
                        ->first();

            if (!empty($vld)) {
                $vld->qty_available = request()->input('stock');
                $vld->save();
            }
        }

        return redirect()->back()->with(['status' => [
                'success' => 1,
                'msg' => __('lang_v1.updated_succesfully')
            ]]);
    }

    public function serviceStaffLineOrders()
    {
        $business_id = request()->session()->get('user.business_id');

        $query = TransactionSellLine::leftJoin('transactions as t', 't.id', '=',
'transaction_sell_lines.transaction_id')
                ->leftJoin('variations as v', 'transaction_sell_lines.variation_id', '=', 'v.id')
                ->leftJoin('products as p', 'v.product_id', '=', 'p.id')
                ->leftJoin('units as u', 'p.unit_id', '=', 'u.id')
                ->leftJoin('product_variations as pv', 'v.product_variation_id', '=', 'pv.id')
                ->leftJoin('users as ss', 'ss.id', '=',
'transaction_sell_lines.res_service_staff_id')
                ->leftjoin(
                    'business_locations AS bl',
                    't.location_id',
                    '=',
                    'bl.id'
                )
                ->where('t.business_id', $business_id)
                ->where('t.type', 'sell')
                ->where('t.status', 'final')
                ->whereNotNull('transaction_sell_lines.res_service_staff_id');


        if (!empty(request()->service_staff_id)) {
            $query->where('transaction_sell_lines.res_service_staff_id', request()-
>service_staff_id);
        }

        if (request()->has('location_id')) {
            $location_id = request()->get('location_id');
            if (!empty($location_id)) {
                $query->where('t.location_id', $location_id);
            }
        }
```

```
        if (!empty(request()->start_date) && !empty(request()->end_date)) {
            $start = request()->start_date;
            $end =  request()->end_date;
            $query->whereDate('t.transaction_date', '>=', $start)
                        ->whereDate('t.transaction_date', '<=', $end);
        }

        $query->select(
            'p.name as product_name',
            'p.type as product_type',
            'v.name as variation_name',
            'pv.name as product_variation_name',
            'u.short_name as unit',
            't.id as transaction_id',
            'bl.name as business_location',
            't.transaction_date',
            't.invoice_no',
            'transaction_sell_lines.quantity',
            'transaction_sell_lines.unit_price_before_discount',
            'transaction_sell_lines.line_discount_type',
            'transaction_sell_lines.line_discount_amount',
            'transaction_sell_lines.item_tax',
            'transaction_sell_lines.unit_price_inc_tax',
            DB::raw('CONCAT(COALESCE(ss.first_name, ""), COALESCE(ss.last_name, "")) as
service_staff')
        );

        $datatable = Datatables::of($query)
            ->editColumn('product_name', function ($row) {
                $name = $row->product_name;
                if ($row->product_type == 'variable') {
                    $name .= ' - ' . $row->product_variation_name . ' - ' . $row->variation_name;
                }
                return $name;
            })
            ->editColumn(
                'unit_price_inc_tax',
                '<span class="display_currency unit_price_inc_tax" data-currency_symbol="true"
data-orig-value="{{$unit_price_inc_tax}}">{{$unit_price_inc_tax}}</span>'
            )
            ->editColumn(
                'item_tax',
                '<span class="display_currency item_tax" data-currency_symbol="true" data-orig-
value="{{$item_tax}}">{{$item_tax}}</span>'
            )
            ->editColumn(
                'quantity',
                '<span class="display_currency quantity" data-unit="{{$unit}}" data-
currency_symbol="false" data-orig-value="{{$quantity}}">{{$quantity}}</span> {{$unit}}'
            )
            ->editColumn(
                'unit_price_before_discount',
                '<span class="display_currency unit_price_before_discount" data-
currency_symbol="true" data-orig-
value="{{$unit_price_before_discount}}">{{$unit_price_before_discount}}</span>'
            )
            ->addColumn(
                'total',
                '<span class="display_currency total" data-currency_symbol="true" data-orig-
value="{{$unit_price_inc_tax * $quantity}}">{{$unit_price_inc_tax * $quantity}}</span>'
            )
            ->editColumn(
                'line_discount_amount',
                function ($row) {
                    $discount = !empty($row->line_discount_amount) ? $row->line_discount_amount :
0;

                    if (!empty($discount) && $row->line_discount_type == 'percentage') {
                        $discount = $row->unit_price_before_discount * ($discount / 100);
                    }

                    return '<span class="display_currency total-discount" data-
currency_symbol="true" data-orig-value="' . $discount . '">' . $discount . '</span>';
                }
            )
            ->editColumn('transaction_date', '{{@format_date($transaction_date)}}')

            ->rawColumns(['line_discount_amount', 'unit_price_before_discount', 'item_tax',
'unit_price_inc_tax', 'item_tax', 'quantity', 'total'])
                    ->make(true);
```

```
        return $datatable;
    }

    public function getProfit($by = null)
    {
        $business_id = request()->session()->get('user.business_id');

        $query = TransactionSellLine
            ::join('transactions as sale', 'transaction_sell_lines.transaction_id', '=',
'sale.id')
            ->leftjoin('transaction_sell_lines_purchase_lines as TSPL',
'transaction_sell_lines.id', '=', 'TSPL.sell_line_id')
            ->leftjoin(
                'purchase_lines as PL',
                'TSPL.purchase_line_id',
                '=',
                'PL.id'
            )
            ->join('products as P', 'transaction_sell_lines.product_id', '=', 'P.id')
            ->where('sale.business_id', $business_id)
            ->where('transaction_sell_lines.children_type', '!=', 'combo');
        //If type combo: find childrens, sale price parent - get PP of childrens
        $query->select(DB::raw('SUM(IF (TSPL.id IS NULL AND P.type="combo", (
    SELECT Sum((tspl2.quantity - tspl2.qty_returned) * (tsl.unit_price_inc_tax -
pl2.purchase_price_inc_tax)) AS total
        FROM transaction_sell_lines AS tsl
            JOIN transaction_sell_lines_purchase_lines AS tspl2
        ON tsl.id=tspl2.sell_line_id
        JOIN purchase_lines AS pl2
        ON tspl2.purchase_line_id = pl2.id
        WHERE tsl.parent_sell_line_id = transaction_sell_lines.id),
        (TSPL.quantity - TSPL.qty_returned) * (transaction_sell_lines.unit_price_inc_tax -
PL.purchase_price_inc_tax))) AS gross_profit'));

        if (!empty(request()->start_date) && !empty(request()->end_date)) {
            $start = request()->start_date;
            $end =  request()->end_date;
            $query->whereDate('sale.transaction_date', '>=', $start)
                        ->whereDate('sale.transaction_date', '<=', $end);
        }

        if ($by == 'product') {
            $query->join('variations as V', 'transaction_sell_lines.variation_id', '=', 'V.id')
                ->leftJoin('product_variations as PV', 'PV.id', '=', 'V.product_variation_id')
                ->addSelect(DB::raw("IF(P.type='variable', CONCAT(P.name, ' - ', PV.name, ' - ',
V.name, ' (', V.sub_sku, ')'), CONCAT(P.name, ' (', P.sku, ')')) as product"))
                ->groupBy('V.id');
        }

        if ($by == 'category') {
            $query->join('variations as V', 'transaction_sell_lines.variation_id', '=', 'V.id')
                ->leftJoin('categories as C', 'C.id', '=', 'P.category_id')
                ->addSelect("C.name as category")
                ->groupBy('C.id');
        }

        if ($by == 'brand') {
            $query->join('variations as V', 'transaction_sell_lines.variation_id', '=', 'V.id')
                ->leftJoin('brands as B', 'B.id', '=', 'P.brand_id')
                ->addSelect("B.name as brand")
                ->groupBy('B.id');
        }

        if ($by == 'location') {
            $query->join('business_locations as L', 'sale.location_id', '=', 'L.id')
                ->addSelect("L.name as location")
                ->groupBy('L.id');
        }

        if ($by == 'invoice') {
            $query->addSelect('sale.invoice_no', 'sale.id as transaction_id')
                ->groupBy('sale.invoice_no');
        }

        if ($by == 'date') {
            $query->addSelect("sale.transaction_date")
                ->groupBy(DB::raw('DATE(sale.transaction_date)'));
        }

        if ($by == 'day') {
```

```
            $results = $query->addSelect(DB::raw("DAYNAME(sale.transaction_date) as day"))
                ->groupBy(DB::raw('DAYOFWEEK(sale.transaction_date)'))
                ->get();

            $profits = [];
            foreach ($results as $result) {
                $profits[strtolower($result->day)] = $result->gross_profit;
            }
            $days = ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday',
'sunday'];

            return view('report.partials.profit_by_day')->with(compact('profits', 'days'));
        }

        if ($by == 'customer') {
            $query->join('contacts as CU', 'sale.contact_id', '=', 'CU.id')
            ->addSelect("CU.name as customer")
                ->groupBy('sale.contact_id');
        }

        $datatable = Datatables::of($query)
            ->editColumn(
                'gross_profit',
                '<span class="display_currency gross-profit" data-currency_symbol="true" data-
orig-value="{{$gross_profit}}">{{$gross_profit}}</span>'
            );

        if ($by == 'category') {
            $datatable->editColumn(
                'category',
                '{{$category ?? __("lang_v1.uncategorized")}}'
            );
        }
        if ($by == 'brand') {
            $datatable->editColumn(
                'brand',
                '{{$brand ?? __("report.others")}}'
            );
        }

        if ($by == 'date') {
            $datatable->editColumn('transaction_date', '{{@format_date($transaction_date)}}');
        }

        $row_columns = ['gross_profit'];
        if ($by == 'invoice') {
            $datatable->editColumn('invoice_no', function ($row) {
                return '<a data-href="' . action('SellController@show', [$row->transaction_id])
                            . '" href="#" data-container=".view_modal" class="btn-modal">' .
$row->invoice_no . '</a>';
            });
            $row_columns[] = 'invoice_no';
        }
        return $datatable->rawColumns($row_columns)
                  ->make(true);
    }

    public function itemsReport(Request $request)
    {
        $business_id = Auth::user()->business_id;

        $query = TransactionSellLinesPurchaseLines::leftJoin('transaction_sell_lines
                    as SL', 'SL.id', '=', 'transaction_sell_lines_purchase_lines.sell_line_id')
                ->leftJoin('stock_adjustment_lines
                    as SAL', 'SAL.id', '=',
'transaction_sell_lines_purchase_lines.stock_adjustment_line_id')
                ->leftJoin('transactions as sale', 'SL.transaction_id', '=', 'sale.id')
                ->leftJoin('transactions as stock_adjustment', 'SAL.transaction_id', '=',
'stock_adjustment.id')
                ->join('purchase_lines as PL', 'PL.id', '=',
'transaction_sell_lines_purchase_lines.purchase_line_id')
                ->join('transactions as purchase', 'PL.transaction_id', '=', 'purchase.id')
                ->join('business_locations as bl', 'purchase.location_id', '=', 'bl.id')
                ->join(
                    'variations as v',
                    'PL.variation_id',
                    '=',
                    'v.id'
                    )
                ->join('product_variations as pv', 'v.product_variation_id', '=', 'pv.id')
                ->join('products as p', 'PL.product_id', '=', 'p.id')
```

```
                ->join('units as u', 'p.unit_id', '=', 'u.id')
                ->leftJoin('contacts as suppliers', 'purchase.contact_id', '=', 'suppliers.id')
                ->leftJoin('contacts as customers', 'sale.contact_id', '=', 'customers.id')
                ->where('purchase.business_id', $business_id)
                ->select(
                    'v.sub_sku as sku',
                    'p.type as product_type',
                    'p.name as product_name',
                    'v.name as variation_name',
                    'pv.name as product_variation',
                    'u.short_name as unit',
                    'purchase.transaction_date as purchase_date',
                    'purchase.ref_no as purchase_ref_no',
                    'purchase.type as purchase_type',
                    'suppliers.name as supplier',
                    'PL.purchase_price_inc_tax as purchase_price',
                    'sale.transaction_date as sell_date',
                    'stock_adjustment.transaction_date as stock_adjustment_date',
                    'sale.invoice_no as sale_invoice_no',
                    'stock_adjustment.ref_no as stock_adjustment_ref_no',
                    'customers.name as customer',
                    'transaction_sell_lines_purchase_lines.quantity as quantity',
                    'SL.unit_price_inc_tax as selling_price',
                    'SAL.unit_price as stock_adjustment_price',
                    'transaction_sell_lines_purchase_lines.stock_adjustment_line_id',
                    'transaction_sell_lines_purchase_lines.sell_line_id',
                    'transaction_sell_lines_purchase_lines.purchase_line_id',
                    'transaction_sell_lines_purchase_lines.qty_returned',
                    'bl.name as location'
                );

        if (!empty($request->input('purchase_start')) && !empty($request->input('purchase_end'))) {
            $start = $request->input('purchase_start');
            $end =  $request->input('purchase_end');
            $query->whereDate('purchase.transaction_date', '>=', $start)
                        ->whereDate('purchase.transaction_date', '<=', $end);
        }

        if (!empty($request->input('sale_start')) && !empty($request->input('sale_end'))) {
            $start = $request->input('sale_start');
            $end =  $request->input('sale_end');
            $query->where(function ($q) use ($start, $end) {
                $q->where(function ($qr) use ($start, $end) {
                    $qr->whereDate('sale.transaction_date', '>=', $start)
                       ->whereDate('sale.transaction_date', '<=', $end);
                })->orWhere(function ($qr) use ($start, $end) {
                    $qr->whereDate('stock_adjustment.transaction_date', '>=', $start)
                       ->whereDate('stock_adjustment.transaction_date', '<=', $end);
                });
            });
        }

        $supplier_id = $request->get('supplier_id', null);
        if (!empty($supplier_id)) {
            $query->where('suppliers.id', $supplier_id);
        }

        $customer_id = $request->get('customer_id', null);
        if (!empty($customer_id)) {
            $query->where('customers.id', $customer_id);
        }

        $location_id = $request->get('location_id', null);
        if (!empty($location_id)) {
            $query->where('purchase.location_id', $location_id);
        }

        $only_mfg_products = $request->get('only_mfg_products', 0);
        if (!empty($only_mfg_products)) {
            $query->where('purchase.type', 'production_purchase');
        }

        return $this->respondArray($query->get());
    }
}
```