BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Kerja praktik magang di PT Sewiwi Indonesia dilakukan sebagai *Back-end Developer* dalam bimbingan Julius Nata Saputra selaku kepala Divisi IT. Untuk memenuhi kebutuhan sistem *Point of Sale* (POS) yang dikoordinasikan oleh Bapak Julius Nata Saputra yang memberikan instruksi. Setelah diberikan instruksi dan melakukan *brainstroming* maka pembuatan *Point of Sale* dengan laravel untuk *back-end* menjadi hasil yang disepakati bersama oleh tim dan kepala Divisi IT sebagai instruktur dalam proyek tersebut.

Penulis mendapatkan proyek tersebut sebagai *back-end developer* yang diawasi dan dibimbing oleh pembimbing lapangan dan instruktur untuk melihat perkembangan proyek tersebut. Pertama penulis melakukan *brainstroming* untuk alur kerja sistem *Point of Sale* tersebut dan pembuatan *Entity Relation Diagram* (ERD) untuk isi dari *database* setelah itu melakukan pengerjaan dan yang terakhir melakukan tes apakah terdapat *bug*.

3.2 Tugas dan Uraian Kerja Magang

3.2.1 Tugas yang Dilakukan

Dalam kerja magang penulis bekerja sebagai *back-end developer*. Penulis belajar untuk memahami cara kerja di PT Sewiwi Indonesia. Penulis diberikan kasus dari klien PT Sewiwi Indonesia yaitu Sekolah Al Wafi untuk membuat aplikasi *point of sale* (POS) dengan pembagian tugas terdiri dari tiga orang dalam tim, penulis sebagai *backend* dan dua orang lainnya sebagai *frontend*. Dalam kasus ini penulis membuat POS yang sesuai dengan konsep yang sudah diberikan oleh instruktur. Setelah itu penulis mengimplementasikan pembuatan POS dari yang sudah dipelajari selama kerja magang.

Tabel 3.1. Pekerjaan yang dilakukan setiap minggu selama kerja magang

Minggu ke-	Pekerjaan yang Dilakukan
1-6	– Perkenalan dan pembekalan mengenai perusahaan. – Training mengenai *framework laravel* serta arsitektur dari *framework* tersebut dan juga *tools* yang akan digunakan selama pelaksanaan kerja magang.
7-8	Mempelajari dan mengimplemtasikan *payment gateway* dengan *third-party* secara simulasi atau *sandbox* untuk melakukan pembayaran.
9	Melakukan *deploy* pada aplikasi yang telah dibuat pada saat *training* dan juga untuk mendukung fitur *payment gateway*.
10-12	– Mempelajari dan mengimplementasi *ReST API* dengan menggunakan *framework laravel* – Mempelajari dan mengimplementasi *authentication* dengan *API* menggunakan *library Laravel Passport*.
14	Merancang dan analisis kebutuhan dalam pembangunan sistem *point of sale* pada koperasi sekolah Al Wafi.
15-18	Pembangunan sistem *point of sale* pada koperasi sekolah Al Wafi

A. Training

Pada pelaksanaan kerja magang di PT Sewiwi Indonesia diawali dengan tahapan pengenalan terhadap perusahaan dari visi, misi, dan peraturan yang berlaku pada perusahaan. Setelah pengenalan perusahaan dibagikan modul untuk mempelajari *framework laravel* secara dasar dan melakukan *training*. Dalam *training* terdapat *assignment* membuat aplikasi yang dikerjakan secara *fullstack*.

Tabel 3.2. Pembelajaran setiap minggu pada *training*

Minggu ke-	**Materi *training* yang diterima**
1	Pengenalan dan instalasi *tools* untuk *framework laravel* pada penggunaan selama kerja magang.
2	– Mempelajari arsitektur dari *framework laravel* – Mempelajari kasus ”N+1 Problem” yang sering terjadi agar beban yang server tidak terlalu berat
3	– Menghubungkan aplikasi dengan *cloud storage* untuk menyimpan aset yang dibutuhkan aplikasi. – Mempelajari *framework tailwind css* dan mengimplementasikannya pada aplikasi *assignment training*.
4	Mempelajari *AlpineJS* dan mengimplementasikannya untuk mempermudah pada tampilan dan *experience* dari pengguna.
5	– Mempelajari *library laravel livewire*. – Mengimplemtasi fitur notifikasi *e-mail* yang terdapat pada laravel.
6	– Mempelajari *background process* dan *scheduler* yang terdapat pada laravel. – Mempelajari Git untuk membantu mengatur versi pada aplikasi.

B. Analisis Kebutuhan

Tahap kerja magang selanjutnya setelah melakukan *training* yaitu analisis kebutuhan dengan melakukan *mind mapping* bersama tim dan Bapak Julius Nata Saputra selaku kepala divisi. Berikut merupakan daftar kebutuhan tersebut.

1. Sistem tersebut memiliki banyak relasi antar tabel dan mudah dalam melakukan *maintainance*. Maka dari itu, *database* MySQL yang digunakan dalam sistem *point of sale* karena cocok untuk data yang memiliki banyak relasi dan mudah untuk di-*maintainance*.
2. *Scan barcode* untuk mendapatkan data barang seperti nama produk dan harga jual satuan pada produk tersebut, serta menampilkan data tersebut ke dafta belanja. Jika *barcode* pada produk di-*scan* dan sudah terdapat dalam daftar belanja maka menambahkan jumlah produk tersebut pada daftar belanja.
3. Perubahan kuantitas secara manual dalam daftar belanja agar kasir tidak harus melakukan *scan barcode* secara berulang-ulang pada saat pembelian suatu produk dalam jumlah besar.
4. *Login* untuk mengetahui identitas yang sedang menggunakan sistem *point of sale* dan juga mengaktifkan sistem. Identitas yang sedang digunakan tidak dapat digunakan secara bersamaan karena untuk menghindari terjadi kecurangan dan kepentingan laporan.
5. Autentikasi yang dapat membedakan identitas antara kasir atau *supervisor*. Autentikasi ini juga dapat berfungsi sebagai otoritas orang yang dapat melakukan fungsi tertentu.

C. Perancangan Sistem

Setelah analisis kebutuhan telah dilakukan pada tahap selanjutnya meruapakan tahap perancangan sistem. Pada tahap ini dialakukan pembuatan *flowchart* dan *entity relationship diagram* untuk sistem yang akan dibuat. Hasil dari perancangan sistem akan menjadi acuan dalam pengembangan sistem selanjutnya.

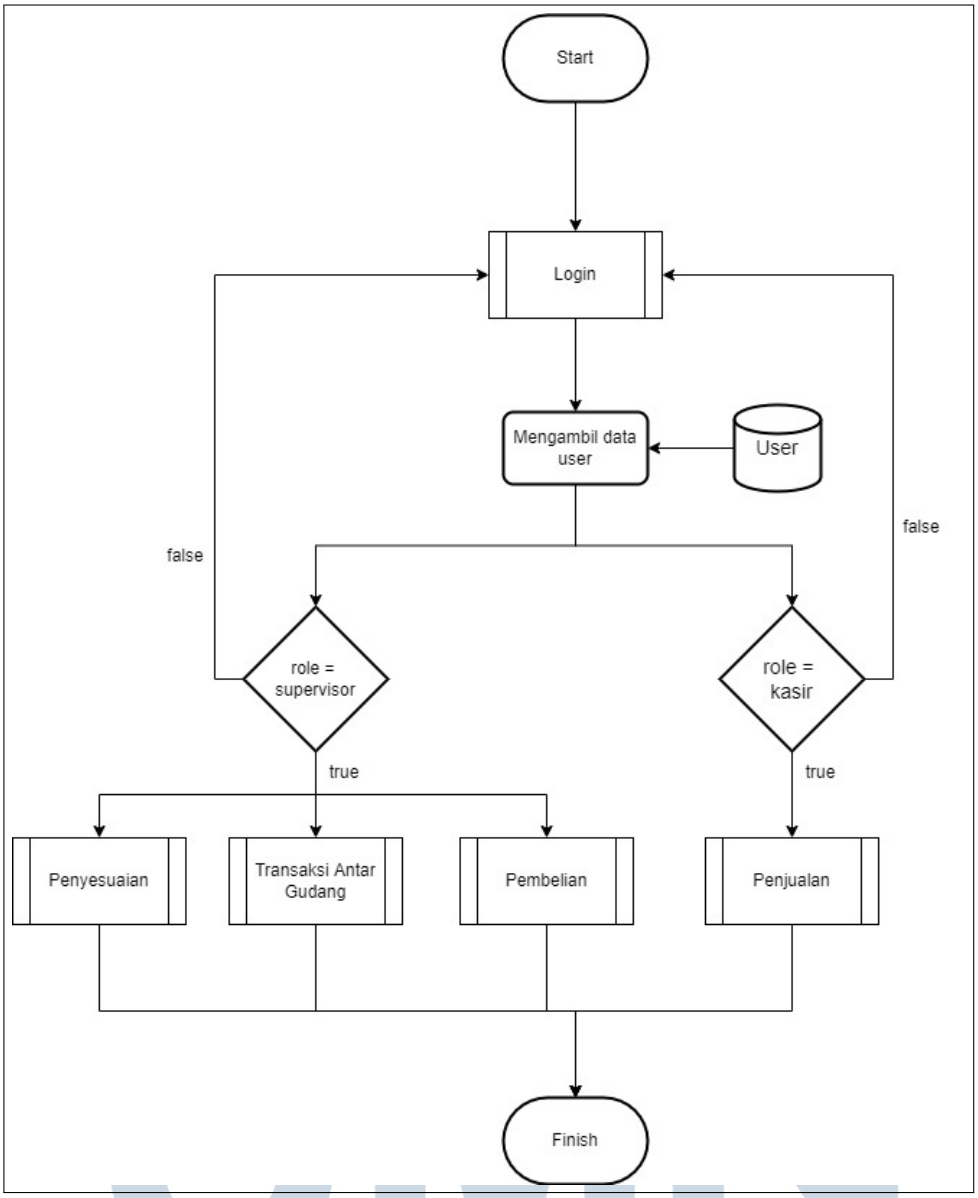
3.2.2 Uraian Kerja Magang

Pelaksanaan kerja magang di PT Sewiwi Indonesia penulis mendapatkan kasus sistem *point of sale* dengan melakukan perancangan sistem dari pembuatan

Flowchart, Entity Relationship Diagram, dan struktur tabel hingga implementasi program.

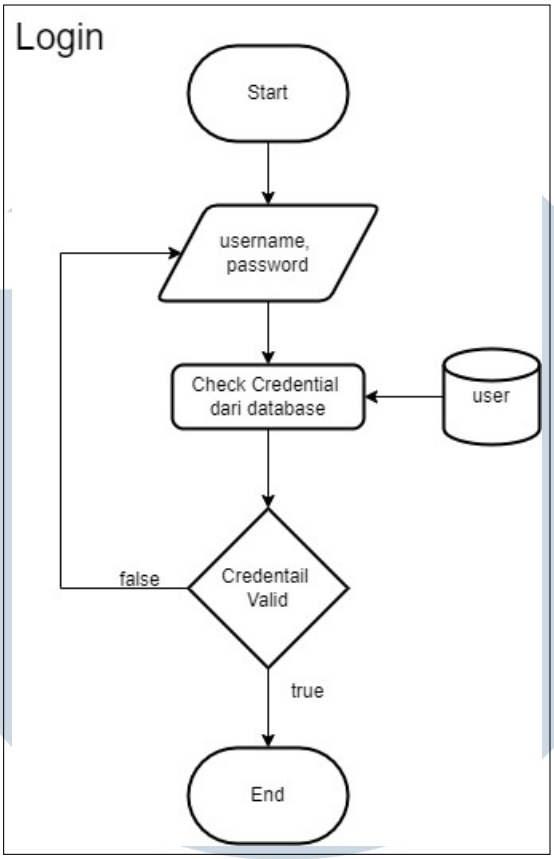
1. *Flowchart*

   Desain *flowchart* pada gambar 3.1 merupakan gambaran besar alur sistem *point of sale* koperasi pada sekolah Al Wafi, sistem tersebut dimulai dengan pengecekan apakah user sudah melakukan *login* atau tidak, jika sudah maka akan mengambil data dari *database* untuk melakukan pengecekan kembali untuk *role* atau hak akses dari user tersebut. Jika hak akses user merupakan kasir maka dapat melakukan penjualan dan pembelian. Sedangkan untuk hak akses user adalah supervisor maka dapat melakukan pembelian, membuat transaksi antar gudang, dan penyesuaian.



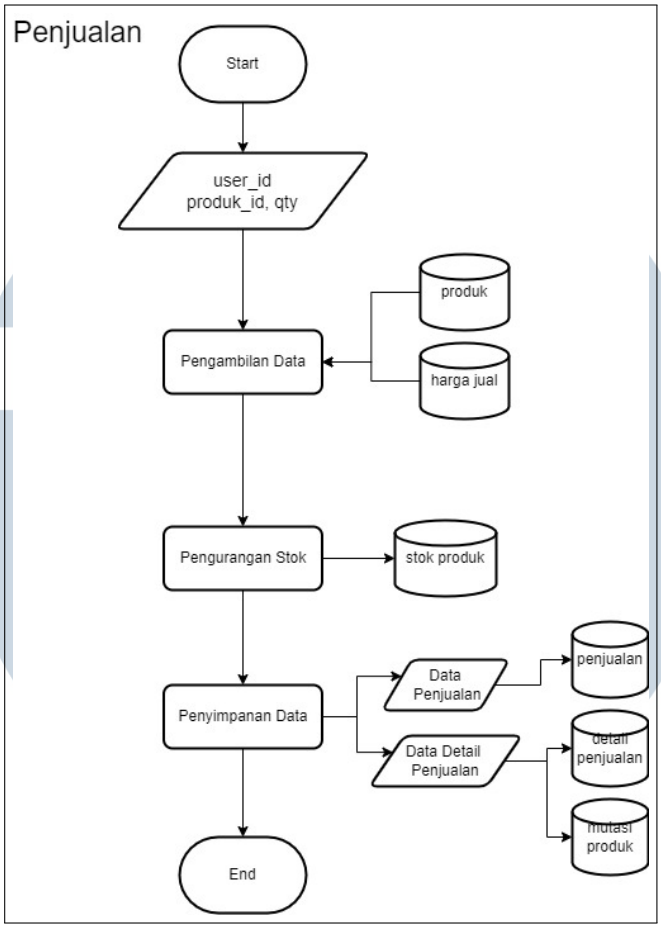
Gambar 3.1. *Flowchart Point of Sale* koperasi Al Wafi

Pada gambar 3.2 merupakan proses login agar dapat melakukan proses selanjutnya, seperti penjualan, pembelian, transaksi antar gudang, atau penyesuaian user harus melakukan *login* terlebih dahulu, agar dapat terlihat *role* atau hak akses user tersebut. Proses *login* harus memasukan *username* dan *password* jika terjadi kesalahan maka harus memasukan kembali yang sesuai.



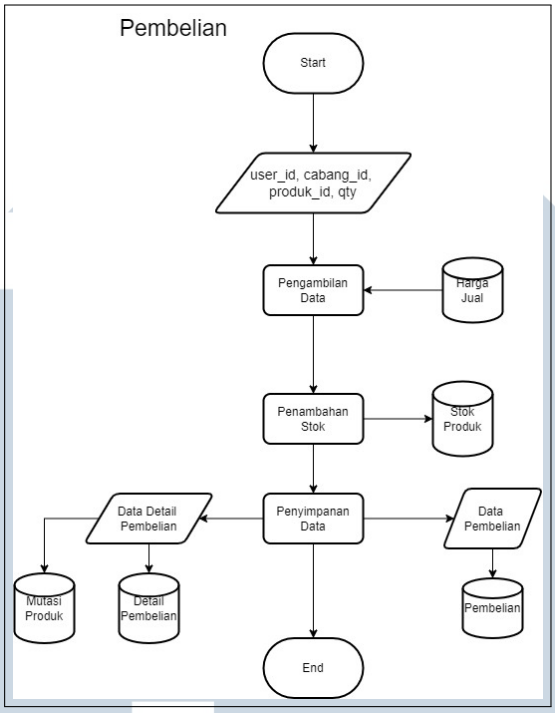
Gambar 3.2. Proses *Login*

Pada gambar 3.3 merupakan proses penjualan dengan melakukan *scan barcode* pada suatu produk lalu terdapat proses pengmabilan data untuk mendapatkan data produk dan harga jual. Setalah mendapatkan data dari produk tersebut seperti id dan nama produk, tahap selanjutnya user dapat melakukan input jumlah produk tersebut untuk dapat ke tahap selanjutnya, yaitu proses pengurangan stok dan penghitungan total dari produk yang terjual pada transaksi tersebut. Pada tahap akhir dari proses penjualan menyimpan transaksi tersebut ke dalam *database*.



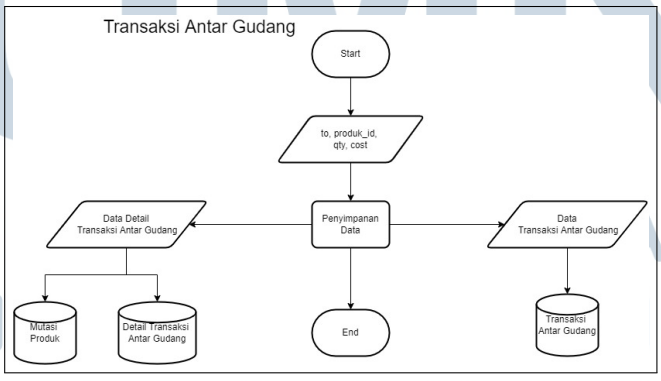
Gambar 3.3. Proses Penjualan

Pada gambar 3.4 merupakan proses Pembelian ini merupakan penambahan stok atau stok baru. Pada proses pembelian sama seperti proses penjualan user melakukan *scan* pada *barcode* yang terdapat pada produk tersebut lalu input jumlah produk dan harga beli atau *cost*.



Gambar 3.4. Proses Pembelian

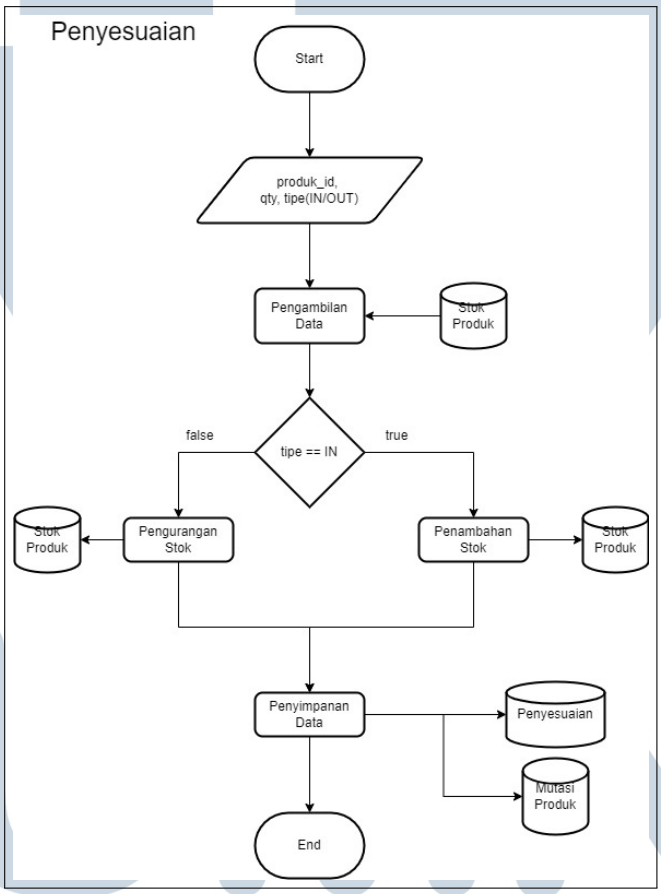
Pada gambar 3.5 merupakan proses transaksi antar gudang, proses ini merupakan pendataan pengiriman produk ke gudang lain. User harus input gudang tujuan, melakukan *scan barcode* dari produk yang dikirim untuk medapatkan data produk, jumlah setiap produk, dan harga beli setiap produk. Pada tahap akhir dari proses ini menyimpan transaksi tersebut ke *database*.



Gambar 3.5. Proses Transaksi Antar Gudang

Pada gambar 3.6 merupakan proses penyesuaian, pada proses ini hanya dapat dilakukan oleh supervisor dengan tujuan menyesuaikan stok suatu produk

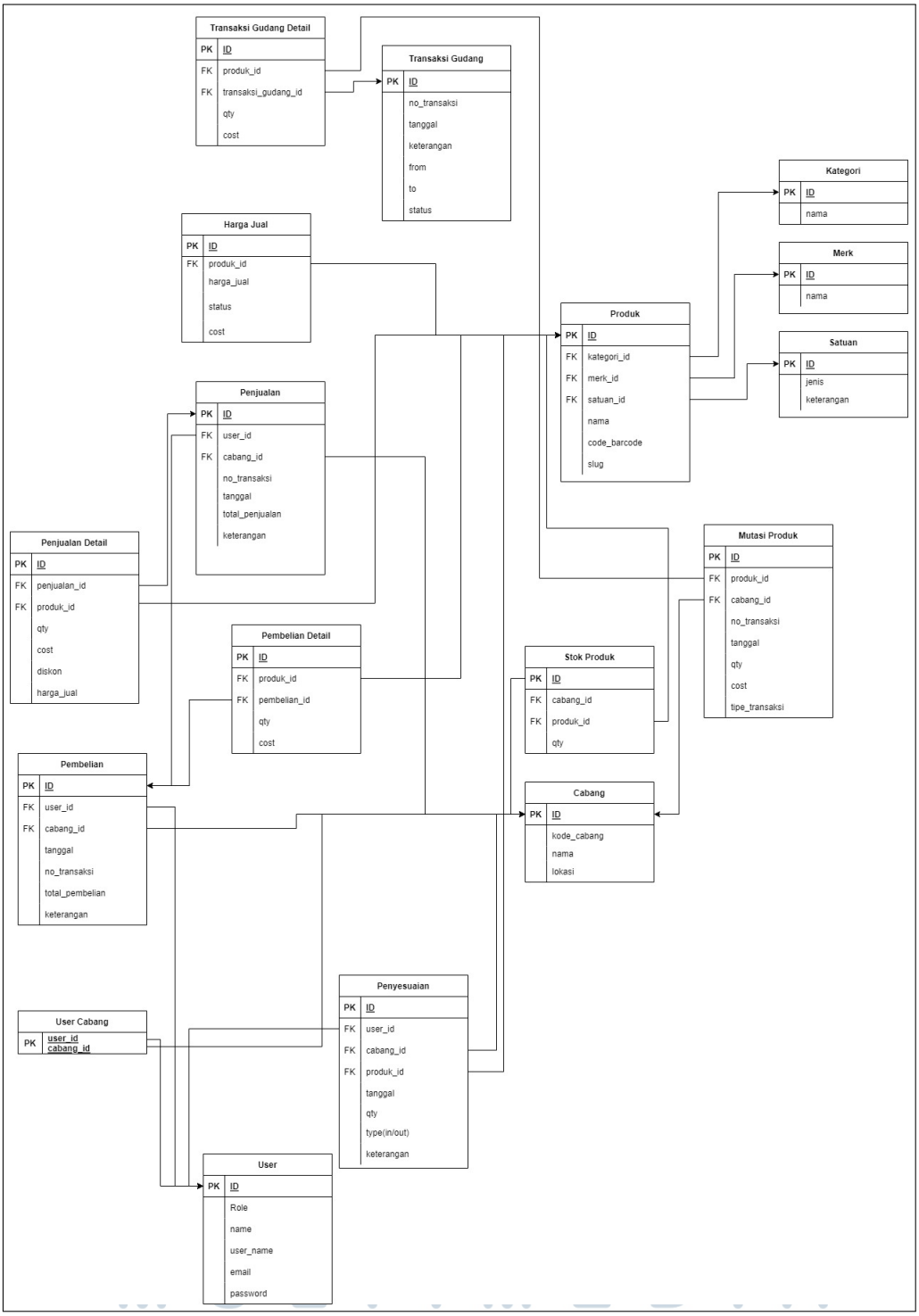
pada suatu gudang. Jika stok suatu produk tidak sesuai antara produk fisik dengan data yang terdapat pada *database*, maka dapat disesuaikan pada proses ini. User dapat mencari produk yang ingin disesuaikan dengan cara mencari nama produk atau dengan *scan barcode* pada produk tersebut untuk mendapatkan data, lalu tentukan tipe penyesuaian *IN/OUT*, dan tetukan jumlah produk. Lalu stok produk akan dikurangi atau ditambah sesuai pilihan dari tipe penyesuaian dan disimpan dalam *database*.



Gambar 3.6. Proses Penyesuaian

2. *Entity Relationship Diagram*

   Berikut merupakan *Entity Relationship Diagram* dari sistem *Point of Sale* yang dibuat untuk sekolah Al Wafi



Gambar 3.7. *Entity Relationship Diagram* dari sistem *Point of Sale* pada koperasi sekolah Al Wafi

Berdasarkan gambar 3.2, terdapat beberapa *table* yang berhubungan, yaitu

tabel Produk dengan *primary key id* berhubungan dengan *produk_id* pada tabel Transaksi Gudang Detail, Harga Jual, Penjualan Detail, Pembelian Detail, Stok Produk, Mutasi Produk, dan Penyesuaian. Tabel Cabang dengan *primary key id* berhubungan dengan *cabang_id* pada tabel Penjualan, Pembelian, Stok Produk, Mutasi Produk, dan Penyesuaian.

3. Struktur Tabel

Database yang diguankan pada sistem *Point of Sale* koperasi sekolah Al Wafi adalah MySQL. Berikut merupakan struktur tabel yang digunakan.

Tabel 3.3. Struktur Tabel cabang

Field Name	**Type**	**Length/Set**	**Information**
id	bigint	20	*Primary key*
kode_cabang	varchar	100	
nama 3	varchar	100	
lokasi	text		

Pada tabel 3.3 merupakan struktur tabel cabang dengan nama tabel pada *database*, ialah cabangs. Tabel ini memiliki fungsi untuk menyimpan cabang atau gudang yang tersedia.

Tabel 3.4. Struktur Tabel harga_jual

Field Name	**Type**	**Length/Set**	**Information**
id	bigint	20	*Primary key*
produk_id	bigint	20	*Reference key* terhadap *id* dalam tabel produk
harga_jual	double		
cost	double		
status	tinyint	1	*Default* = 1

Pada tabel 3.4 merupakan struktur tabel dari harga jual. Tabel ini memiliki fungsi untuk menyimpan harga jual dan harga pokok atau *cost* pada suatu produk.

Tabel 3.5. Struktur Tabel kategori

Field Name	Type	Length/Set	Information
id	bigint	20	*Primary key*
nama	varchar	255	

Pada tabel 3.5 merupakan struktur tabel dari kategori. Tabel ini menyimpan kategori dari produk yang tersedia di koperasi.

Tabel 3.6. Struktur Tabel merk

Field Name	Type	Length/Set	Information
id	bigint	20	*Primary key*
nama	varchar	255	

Pada tabel 3.6 merupakan struktur tabel merk. Tabel ini merupakan yang menyimpan merk dari suatu produk yang tersedia pada koperasi.

Tabel 3.7. Struktur Tabel mutasi_produk

Field Name	Type	Length/Set	Information
id	bigint	20	*Primary key*
produk_id	bigint	20	*Reference key* terhadap *id* dalam tabel produk
cabang_id	bigint	20	*Reference key* terhadap *id* dalam tabel cabang
nomor_transaksi	varchar	255	
tanggal	date		
qty	decimal	8	
tipe_transaksi	enum	Pembelian, Penjualan, Transaksi Antar Gudang, Penyesuaian	

Pada tabel 3.7 merupakan struktur tabel dari mutasi_produk dengan fungsi menyimpan semua transaksi yang terjadi pada suatu produk.

Tabel 3.8. Struktur Tabel pembelian

Field Name	Type	Length/Set	Information
id	bigint	20	*Primary key*
user_id	bigint	20	*Reference key* terhadap *id* dalam tabel user
cabang_id	bigint	20	*Reference key* terhadap *id* dalam tabel cabang
tanggal	date		
no_transaksi	varchar	255	
total_pembelian	double		
keterangan	text		

Pada tabel 3.8 merupakan struktur dari tabel pembelian untuk menyimpan pembelian atau stok baru pada produk pada suatu cabang.

Tabel 3.9. Struktur Tabel pembelian_detail

Field Name	Type	Length/Set	Information
id	bigint	20	*Primary key*
pembelian_id	bigint	20	*Reference key* terhadap *id* dalam tabel pembelian
produk_id	bigint	20	*Reference key* terhadap *id* dalam tabel produk
qty	int	11	
cost	double		

Pada tabel 3.9 merupakan tabel dari detail pembelian yang menyimpan detail atau produk-produk yang terdapat pada pembelian.

Tabel 3.10. Sturktur Tabel penjualan

Field Name	Type	Length/set	Information
id	bigint	20	*Primary key*
user_id	bigint	20	*Reference key* terhadap *id* dalam tabel user
cabang_id	bigint	20	*Reference key* terhadap *id* dalam tabel cabang
tanggal	date		
total_penjualan	double		
keterangan	text		

Pada tabel 3.10 merupakan struktur dari tabel penjualan yang menyimpan penjualan pada suatu cabang.

Tabel 3.11. Sturktur Tabel penjualan_detail

Field Name	Type	Length/set	Information
id	bigint	20	*Primary key*
penjualan_id	bigint	20	*Reference key* terhadap *id* dalam tabel penjualan
produk_id	bigint	20	*Reference key* terhadap *id* dalam tabel produk
qty	int	11	
cost	double		
diskon	double		
harga_jual	double		

Pada tabel 3.11 merupakan struktur dari tabel detail penjualan yang menyimpan detail dari suatu transaksi penjualan.

Tabel 3.12. Sturktur Tabel penyesuaian

Field Name	Type	Length/set	Information
id	bigint	20	*Primary key*
no_transaksi	varchar	255	
user_id	bigint	20	*Reference key* terhadap *id* dalam tabel user
cabang_id	bigint	20	*Reference key* terhadap *id* dalam tabel cabang
produk_id	bigint	20	*Reference key* terhadap *id* dalam tabel produk
tanggal	date		
qty	int	11	
type	enum	IN, OUT	
keterangan	text		

Pada tabel 3.12 merupakan struktur dari tabel penyesuaian yang menyimpan data jika terjadi pada suatu cabang yang tidak sesuainya antara stok pada *database* dengan barang fisiknya.

Tabel 3.13. Sturktur Tabel produk

Field Name	Type	Length/set	Information
id	bigint	20	*Primary key*
kategori_id	bigint	20	*Reference key* terhadap *id* dalam tabel kategori
merk_id	bigint	20	*Reference key* terhadap *id* dalam tabel merk
satuan_id	bigint	20	*Reference key* terhadap *id* dalam tabel satuan
nama	varchar	255	
kode_barcode	varchar	255	*unique*
slug	varchar	255	

Pada tabel 3.13 merupakan struktur dari tabel produk yang menyimpan data produk yang tersedia pada koperasi.

Tabel 3.14. Sturktur Tabel satuan

Field Name	Type	Length/set	Information
id	bigint	20	*Primary key*
jenis	varchar	255	
keterangan	text		

Pada tabel 3.14 merupakan tabel satuan dengan fungsi menyimpan satuan pada suatu produk.

Nama tabel: stok_produk

Fungsi: Tabel ini memiliki fungsi menyimpan jumlah stok suatu produk dan di suatu cabang.

Tabel 3.15. Sturktur Tabel stok_produk

Field Name	Type	Length/set	Information
id	bigint	20	*Primary key*
cabang_id	bigint	20	*Reference key* terhadap *id* dalam tabel cabang
produk_id	bigint	20	*Reference key* terhadap *id* dalam tabel produk
qty	int	11	

Pada tabel 3.15 merupakan struktur tabel stok produk yang menyimpan jumlah suatu produk pada suatu cabang.

Tabel 3.16. Sturktur Tabel transaksi_gudang

Field Name	Type	Length/set	Information
id	bigint	20	*Primary key*
no_transaksi	varchar	255	
tanggal	date		
keterangan	text		
from	bigint	20	Berisi *id* dari tabel cabang
to	bigint	20	Berisi *id* dari tabel cabang
status	enum	Delivery, Confirmed	

Pada tabel 3.16 merupakan struktur tabel dari transaksi antar gudang yang menyimpan terjadinya transaksi pengiriman barang antar gudang atau cabang.

Tabel 3.17. Sturktur Tabel transaksi_gudang_detail

Field Name	**Type**	**Length/set**	**Information**
id	bigint	20	*Primary key*
transaksi_gudang_id	bigint	20	*Reference key* terhadap *id* dalam tabel transaksi_gudang
produk_id	bigint	20	*Reference key* terhadap *id* dalam tabel produk
qty	int	11	

Pada tabel 3.17 meruapakan struktur tabel dari detail transaksi antar gudang yang menyimpan detail dari suatu transaksi antar gudang.

Tabel 3.18. Sturktur Tabel user

Field Name	**Type**	**Length/set**	**Information**
id	bigint	20	*Primary key*
user_name	varchar	255	*Unique*
name	varchar	255	
role	enum	admin, supervisor, kasir	
email	varchar	255	*Unique*
password	varchar	255	

Pada tabel 3.18 merupakan struktur tabel user yang menyimpan data user serta akun atau user yang dapat melakukan *login*.

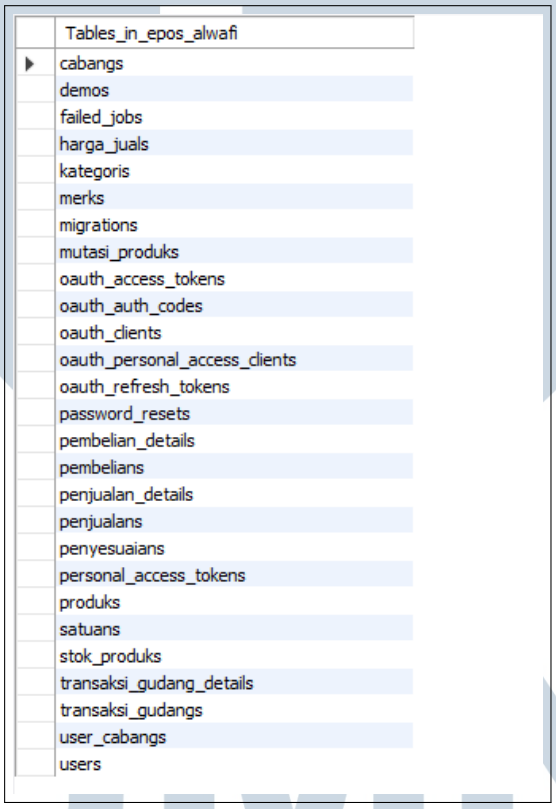
Tabel 3.19. Sturktur Tabel user_cabang

Field Name	**Type**	**Length/set**	**Information**
user_id	bigint	20	*Reference key* terhadap *id* dalam tabel user
cabang_id	bigint	20	*Reference key* terhadap *id* dalam tabel cabang

Pada tabel 3.19 merupakan struktur tabel user cabang. Tabel ini berfungsi sebagai filter user yang memiliki hak akses untuk suatu cabang.

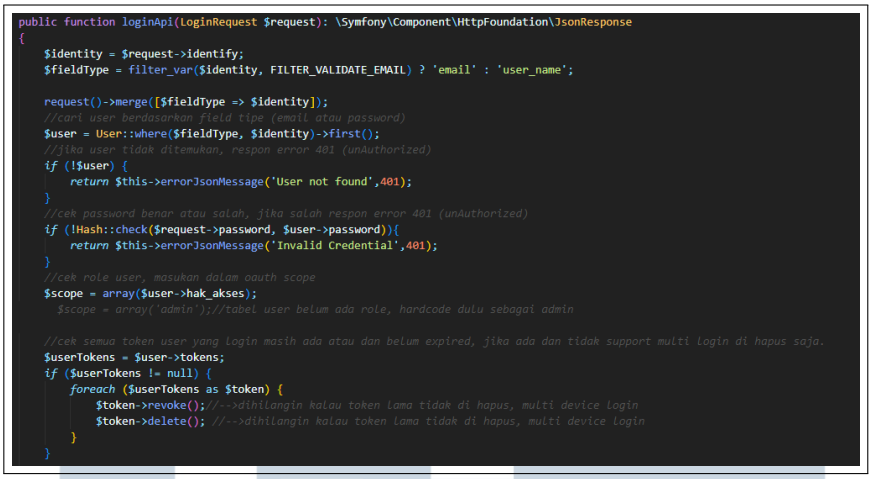
4. Implementasi Program

Pada implementasi diawali dengan pembuatan tabel untuk *database* yang sudah dirancang pada *Entity Relationship Diagram* seperti pada gambar 3.7, pada gambar 3.8 merupakan *list table* yang sudah diimplementasi.



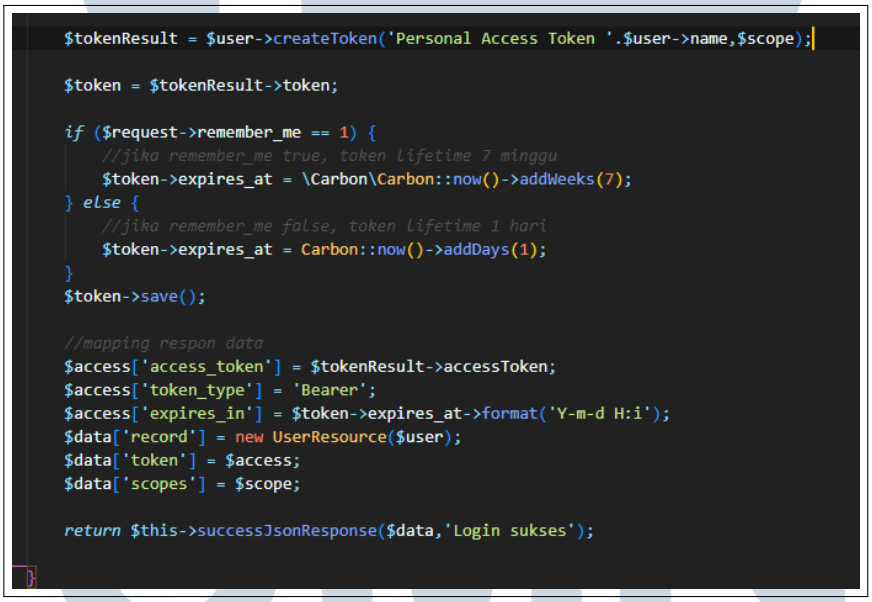
Gambar 3.8. *List table* pada *database*

Untuk melakukan proses seperti pembelian, penjualan, penyesuaian, maupun transaksi antar gudang harus melakukan *login* terlebih dahulu. *Login* dapat dilakukan hanya dengan melakukan *input username/email* dan *password* pada akun yang sudah terdaftar.



```php
public function loginApi(LoginRequest $request): \Symfony\Component\HttpFoundation\JsonResponse
{
    $identity = $request->identify;
    $fieldType = filter_var($identity, FILTER_VALIDATE_EMAIL) ? 'email' : 'user_name';

    request()->merge([$fieldType => $identity]);
    //cari user berdasarkan field tipe (email atau password)
    $user = User::where($fieldType, $identity)->first();
    //jika user tidak ditemukan, respon error 401 (unAuthorized)
    if (!$user) {
        return $this->errorJsonMessage('User not found',401);
    }
    //cek password benar atau salah, jika salah respon error 401 (unAuthorized)
    if (!Hash::check($request->password, $user->password)){
        return $this->errorJsonMessage('Invalid Credential',401);
    }
    //cek role user, masukan dalam oauth scope
    $scope = array($user->hak_akses);
      $scope = array('admin');//tabel user belum ada role, hardcode dulu sebagai admin

    //cek semua token user yang login masih ada atau dan belum expired, jika ada dan tidak support multi login di hapus saja.
    $userTokens = $user->tokens;
    if ($userTokens != null) {
        foreach ($userTokens as $token) {
            $token->revoke();//-->dihilangin kalau token lama tidak di hapus, multi device login
            $token->delete(); //-->dihilangin kalau token lama tidak di hapus, multi device login
        }
    }
```

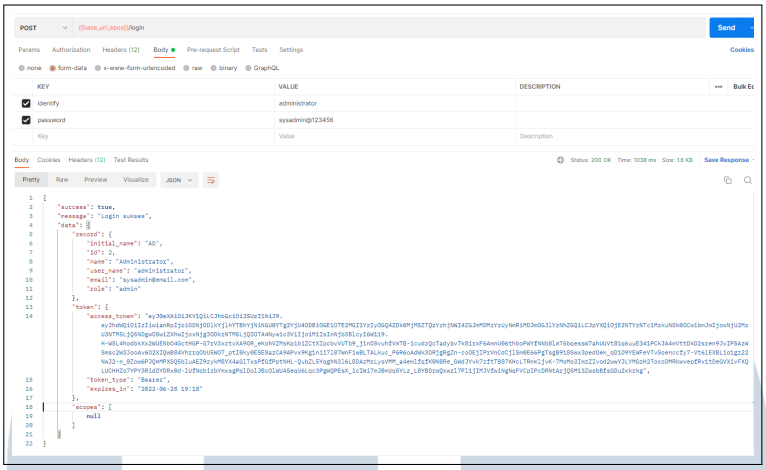
Gambar 3.9. *Controller Login* (1)



```php
    $tokenResult = $user->createToken('Personal Access Token '.$user->name,$scope);

    $token = $tokenResult->token;

    if ($request->remember_me == 1) {
        //jika remember_me true, token lifetime 7 minggu
        $token->expires_at = \Carbon\Carbon::now()->addWeeks(7);
    } else {
        //jika remember_me false, token lifetime 1 hari
        $token->expires_at = Carbon::now()->addDays(1);
    }
    $token->save();

    //mapping respon data
    $access['access_token'] = $tokenResult->accessToken;
    $access['token_type'] = 'Bearer';
    $access['expires_in'] = $token->expires_at->format('Y-m-d H:i');
    $data['record'] = new UserResource($user);
    $data['token'] = $access;
    $data['scopes'] = $scope;

    return $this->successJsonResponse($data,'Login sukses');

}
```

Gambar 3.10. *Controller Login* (2)

Setelah melakukan input selanjutnya akan melakukan *check credential* jika valid maka response dari API seperti pada gambar 3.9 dan akan mengirimkan *token* dengan tipe *Bearer* seperti pada gambar 3.10, dan pada gambar 3.11 merupakan respon dari API dengan mengembalikan *token*



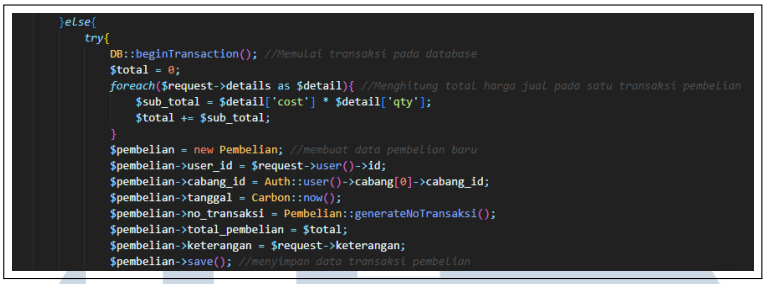
Gambar 3.11. Hasil response API Login

Proses implementasi selanjutnya ialah proses pembelian produk seperti pada gambar terdapat beberapa parameter agar proses *input* pembelian dapat melewati validasi seperti pada gambar 3.12 merupakan validasi untuk *input* detail pembelian.

```php
public function store(Request $request) : JsonResponse
{
    $validator = Validator::make($request->all(),[ //melakukan validasi untuk input dari user
        'details.*.produk_id' => 'required',
        'details.*.qty' => 'gt:0|required|numeric',
    ],[
        'details.*.produk_id.required' => 'Produk tidak tersedia',
        'details.*.qty.gt' => 'Kuantitas produk harus lebih dari 0',
        'details.*.qty.required' => 'Kuantitas barang harus diisi',
        'details.*.qty.numeric' => 'format kuantitas tidak dikenal'
    ]);

    if($validator->fails()){
        //jika terjadi kesalahan pada validasi maka mengembalik respon error
        return $this->errorJsonMessage('Data tidak sesuai validasi',422,$validator->getMessageBag());
    }
```

Gambar 3.12. Validasi *input* pada *controller* pembelian

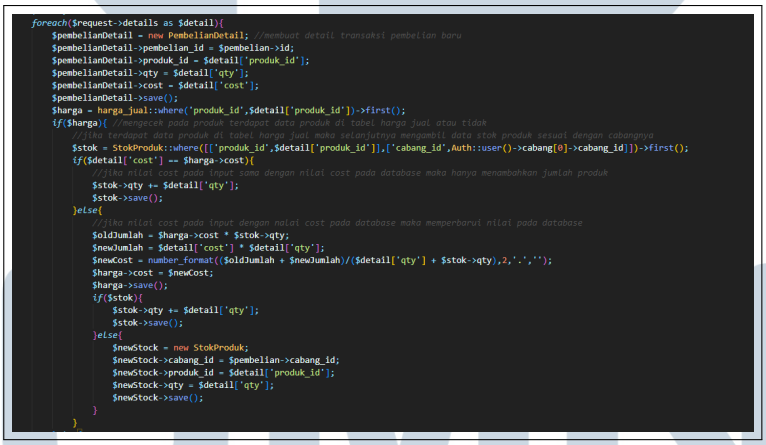
Pada proses pembelian akan menghitung total harga pokok produk pada transaksi pembelian tersebut dan menyimpan data tersebut sebagai *master* seperti pada gambar 3.13. Namun, pembelian untuk suatu cabang terdapat filter pada tabel user_cabang sesuai dengan akun yang sedang digunakan.



```
}else{
    try{
        DB::beginTransaction(); //Memulai transaksi pada database
        $total = 0;
        foreach($request->details as $detail){ //Menghitung total harga jual pada satu transaksi pembelian
            $sub_total = $detail['cost'] * $detail['qty'];
            $total += $sub_total;
        }
        $pembelian = new Pembelian; //membuat data pembelian baru
        $pembelian->user_id = $request->user()->id;
        $pembelian->cabang_id = Auth::user()->cabang[0]->cabang_id;
        $pembelian->tanggal = Carbon::now();
        $pembelian->no_transaksi = Pembelian::generateNoTransaksi();
        $pembelian->total_pembelian = $total;
        $pembelian->keterangan = $request->keterangan;
        $pembelian->save(); //menyimpan data transaksi pembelian
```

Gambar 3.13. *Controller* Pembelian (1)

Jika *master* pembelian sudah tersimpan maka selanjutnya menyimpan data detail pembelian yang terdapat produk yang diperbarui stok, jika suatu produk memiliki nilai harga pokok yang di-*input* oleh user berbeda dengan harga pokok pada *database* maka harga pokok akan diperbarui dengan mengambil rata-rata. Namun, jika harga pokok keduanya sama maka akan hanya ada proses penambahan stok seperti pada gambar 3.14 dan gambar 3.15



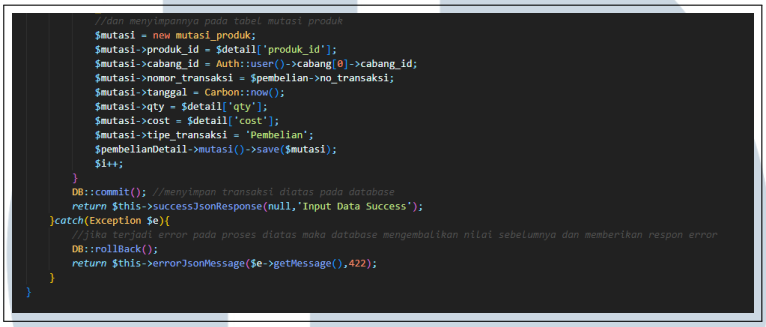
```
foreach($request->details as $detail){
    $pembelianDetail = new PembelianDetail; //membuat detail transaksi pembelian baru
    $pembelianDetail->pembelian_id = $pembelian->id;
    $pembelianDetail->produk_id = $detail['produk_id'];
    $pembelianDetail->qty = $detail['qty'];
    $pembelianDetail->cost = $detail['cost'];
    $pembelianDetail->save();
    $harga = harga_jual::where('produk_id',$detail['produk_id'])->first();
    if($harga){ //mengecek pada produk terdapat data produk di tabel harga jual atau tidak
        //jika terdapat data produk di tabel harga jual maka selanjutnya mengambil data stok produk sesuai dengan cabangnya
        $stok = StokProduk::where([['produk_id',$detail['produk_id']],['cabang_id',Auth::user()->cabang[0]->cabang_id]])->first();
        if($detail['cost'] == $harga->cost){
            //jika nilai cost pada input sama dengan nilai cost pada database maka hanya menambahkan jumlah produk
            $stok->qty += $detail['qty'];
            $stok->save();
        }else{
            //jika nilai cost pada input dengan nalai cost pada database maka memperbarui nilai pada database
            $oldJumlah = $harga->cost * $stok->qty;
            $newJumlah = $detail['cost'] * $detail['qty'];
            $newCost = number_format(($oldJumlah + $newJumlah)/($detail['qty'] + $stok->qty),2,'.','');
            $harga->cost = $newCost;
            $harga->save();
            if($stok){
                $stok->qty += $detail['qty'];
                $stok->save();
            }else{
                $newStock = new StokProduk;
                $newStock->cabang_id = $pembelian->cabang_id;
                $newStock->produk_id = $detail['produk_id'];
                $newStock->qty = $detail['qty'];
                $newStock->save();
            }
        }
```

Gambar 3.14. *Controller* Pembelian(2)

```
}else{
    $hargJual = new harga_jual;
    $hargJual->produk_id = $detail['produk_id'];
    $hargJual->harga_jual = $detail['harga_jual'];
    $hargJual->cost = $detail['cost'];
    $hargJual->save();
    $newStock = new StokProduk;
    $newStock->cabang_id = Auth::user()->cabang[0]->cabang_id;
    $newStock->produk_id = $detail['produk_id'];
    $newStock->qty = $detail['qty'];
    $newStock->save();
}
```

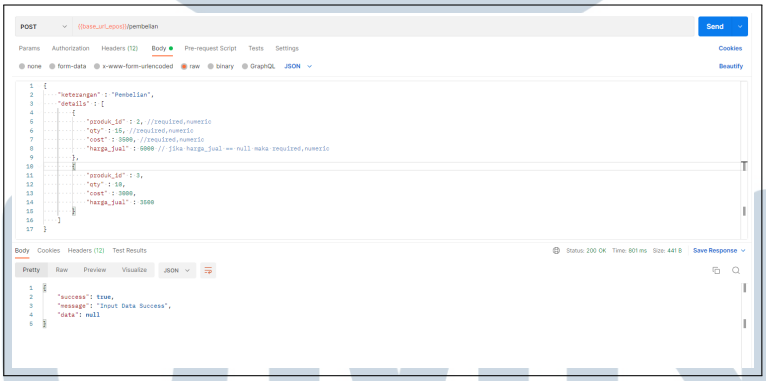
Gambar 3.15. *Controller* Pembelian(3)

Pada gambar 3.16 semua transaksi produk akan tersimpan pada tabel mutasi_produk dan memberikan respon bahwa data berhasil disimpan dan jika terjadi *error* maka akan memberikan respon bahwa terdapat kesalahan pada proses diatas.



Gambar 3.16. *Controller* Pembelian(4)

Gambar 3.17 merupakan hasil dari respon API untuk melakukan transaksi pembelian.



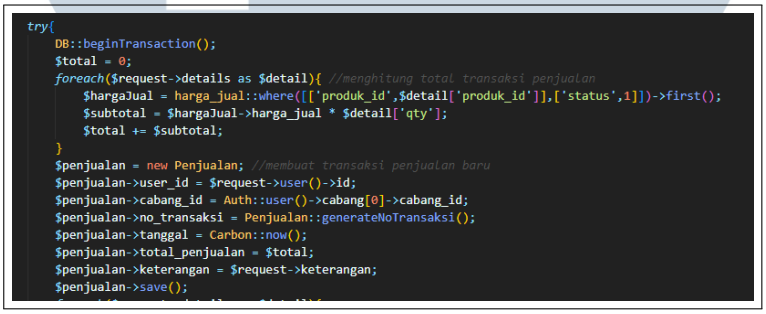
Gambar 3.17. Hasil respon API Pembelian

Pada penjualan terdapat parameter seperti *details* dengan tipe *array associative* merupakan detail dari penjualan pada gambar 3.18 merupakan validasi pada *controller* penjualan.

```php
public function store(Request $request) : JsonResponse
{
    $validator = Validator::make($request->all(),[ //melakukan validasi untuk input dari user
        'details.*.produk_id' => 'required',
        'details.*.qty' => 'gt:0|required|numeric',
    ],[
        'details.*.produk_id.required' => 'Produk tidak tersedia',
        'details.*.qty.gt' => 'Kuantitas produk harus lebih dari 0',
        'details.*.qty.required' => 'Kuantitas barang harus diisi',
        'details.*.qty.numeric' => 'format kuantitas tidak dikenal'
    ]);

    if($validator->fails()){
        //jika terjadi kesalahan pada validasi maka mengembalik respon error
        return $this->errorJsonMessage('Data tidak sesuai validasi',422,$validator->getMessageBag());
    }
```

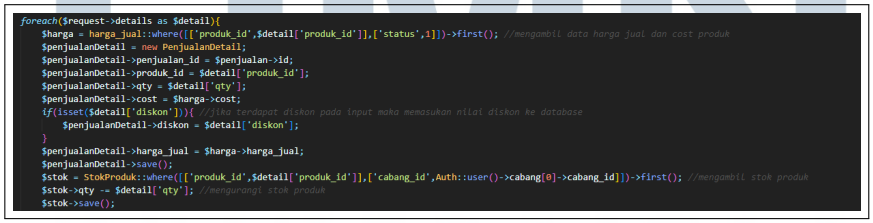
Gambar 3.18. Validasi *input* pada *controller* penjualan

Setelah dapat melewati validasi selanjutnya ke proses menghitung total harga produk yang terjual pada transaksi tersebut dan disimpan transanksi penjualan tersebut sebagai *master* seperti pada gambar 3.19. Selanjutnya proses pengurangan stok sesuai dengan produk terdapat pada *details* seperti gambar 3.20.



```php
try{
    DB::beginTransaction();
    $total = 0;
    foreach($request->details as $detail){ //menghitung total transaksi penjualan
        $hargaJual = harga_jual::where([['produk_id',$detail['produk_id']],['status',1]])->first();
        $subtotal = $hargaJual->harga_jual * $detail['qty'];
        $total += $subtotal;
    }
    $penjualan = new Penjualan; //membuat transaksi penjualan baru
    $penjualan->user_id = $request->user()->id;
    $penjualan->cabang_id = Auth::user()->cabang[0]->cabang_id;
    $penjualan->no_transaksi = Penjualan::generateNoTransaksi();
    $penjualan->tanggal = Carbon::now();
    $penjualan->total_penjualan = $total;
    $penjualan->keterangan = $request->keterangan;
    $penjualan->save();
```

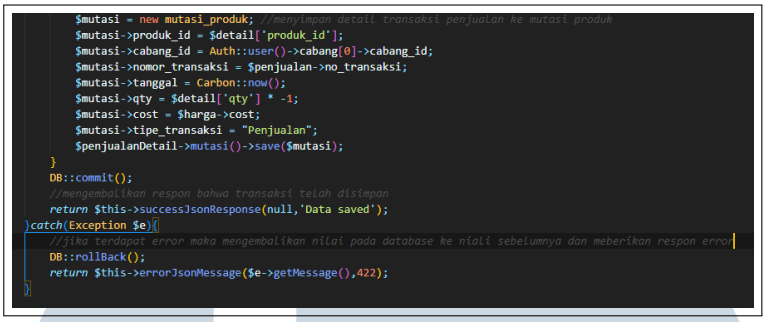
Gambar 3.19. *Controller* Penjualan(1)



```php
foreach($request->details as $detail){
    $harga = harga_jual::where([['produk_id',$detail['produk_id']],['status',1]])->first(); //mengambil data harga jual dan cost produk
    $penjualanDetail = new PenjualanDetail;
    $penjualanDetail->penjualan_id = $penjualan->id;
    $penjualanDetail->produk_id = $detail['produk_id'];
    $penjualanDetail->qty = $detail['qty'];
    $penjualanDetail->cost = $harga->cost;
    if(isset($detail['diskon'])){ //jika terdapat diskon pada input maka memasukan nilai diskon ke database
        $penjualanDetail->diskon = $detail['diskon'];
    }
    $penjualanDetail->harga_jual = $harga->harga_jual;
    $penjualanDetail->save();
    $stok = StokProduk::where([['produk_id',$detail['produk_id']],['cabang_id',Auth::user()->cabang[0]->cabang_id]])->first(); //mengambil stok produk
    $stok->qty -= $detail['qty']; //mengurangi stok produk
    $stok->save();
```

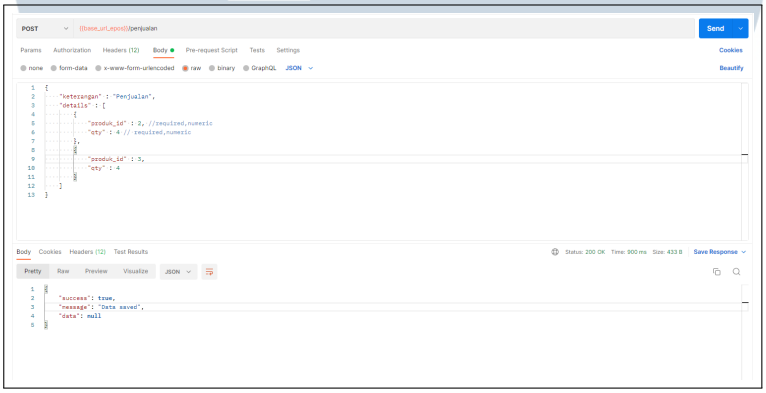
Gambar 3.20. *Controller* Penjualan(2)

Pada gambar 3.21 melakukan proses penyimpanan transaksi produk ke mutasi produk dan memberikan respon bahwa keseluruhan transaksi berhasil disimpan, namun jika terjadi *error* maka akan diberikan respon bahwa telah terjadi kesalahan.



Gambar 3.21. *Controller* Penjualan(3)

Pada gambar 3.22 merupakan *input* untuk penjualan dan hasil respon dari API.



Gambar 3.22. Hasil respon API Penjualan

Proses transaksi antar gudang untuk mengirim produk ke gudang lain, pada gambar 3.23 merupakan validasi dari *controller* transaksi antar gudang jika terjadi kesalahan pada validasi maka API akan memberikan respon bahwa terjadi kesalahan pada validasi *input*.

```php
public function store(Request $request) : JsonResponse
{
    $validator = Validator::make($request->all(),[
        'to' => 'required|numeric',
        'details.*.produk_id' => 'required',      You, 1 second ago • Uncommitted changes
        'details.*.cost' => 'numeric',
        'details.*.qty' => 'required|numeric|gt:0'
    ],[
        'to.required' => 'Tujuan cabang harus diisi',
        'to.numeric' => 'Tujuan cabang tidak sesuai',
        'details.*.produk_id.required' => 'Produk harus diisi',
        'details.*.produk_id.numeric' => 'Produk tidak sesuai',
        'details.*.cost.numeric' => 'Harga pokok tidak sesuai',
        'details.*.qty.required' => 'Kuantitas produk harus diisi',
        'details.*.qty.numeric' => 'Kuantitas produk tidak sesuai',
        'details.*.qty.gt' => 'Kuantitas produk harus lebih dari 0'
    ]);

    if($validator->fails()){
        return $this->errorJsonMessage('Error validasi',422,$validator->getMessageBag());
```

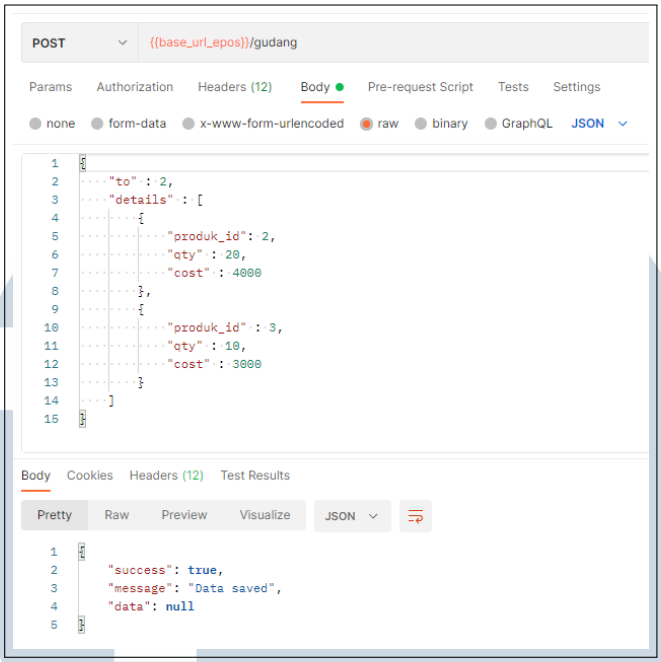
Gambar 3.23. Validasi *input* pada *controller* transaksi antar gudang

Untuk dapat melakukan proses ini hanya *user* dengan hak akses atau *role* sebagai *supervisor* dan melakukan proses ini hanya tentukan gudang tujuan dan *input* produk yang tersedia. Pada gambar 3.24 merupakan proses penyimpanan data transaksi antar gudang dan disimpan juga ke mutasi produk.

```php
}else{
    try{
        DB::beginTransaction();
        $transaksiGudang = new TransaksiGudang;
        $transaksiGudang->no_transaksi = TransaksiGudang::generateNoTransaksi();
        $transaksiGudang->tanggal = Carbon::now();
        $transaksiGudang->keterangan = $request->keterangan;
        $transaksiGudang->from = Auth::user()->cabang[0]->cabang_id;
        $transaksiGudang->to = $request->to;
        $transaksiGudang->save();
        $details = (object) $request->details;
        foreach($details as $detail){
            $transaksiGudangDetail = new TransaksiGudangDetail;
            $transaksiGudangDetail->transaksi_gudang_id = $transaksiGudang->id;
            $transaksiGudangDetail->produk_id = $detail['produk_id'];
            $transaksiGudangDetail->qty = $detail['qty'];
            $transaksiGudangDetail->save();
            $mutasi = new mutasi_produk;
            $mutasi->produk_id = $detail['produk_id'];
            $mutasi->cabang_id = $request->to;
            $mutasi->nomor_transaksi = $transaksiGudang->no_transaksi;
            $mutasi->tanggal = Carbon::now();
            $mutasi->qty = $detail['qty'];
            $mutasi->cost = $detail['cost'];
            $mutasi->tipe_transaksi = "Transaksi Antar Gudang";
            $transaksiGudangDetail->mutasi()->save($mutasi);
        }
        DB::commit();
        return $this->successJsonResponse(null,'Data saved');
    }catch(Exception $e){
        return $this->errorJsonMessage($e->getMessage(),403);
        DB::rollBack();
    }
}
```

Gambar 3.24. *Controller* transaksi antar gudang

Pada gambar 3.25 merupakan hasil dari respon dari API transaksi antar gudang, pada proses ini memberitahukan bahwa terdapat pengiriman produk dari suatu cabang ke suatu cabang.



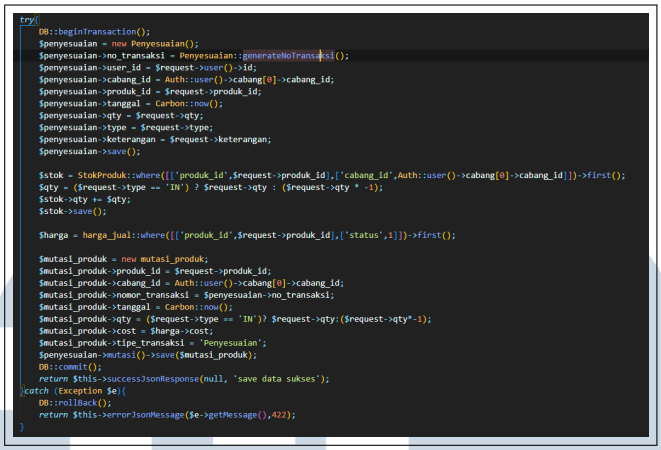
Gambar 3.25. Hasil respon API transaksi antar gudang

Penyesuaian merupakan proses yang menyesuaikan antara stok pada bentuk fisik dengan stok pada *database*. Proses ini hanya dapat dilakukan oleh *supervisor*. Dalam proses penyesuaian terdapat validasi seperti pada gambar 3.26.

```
public function store(Request $request)
{
    $validator = Validator::make($request->all(),[
        "produk_id" => 'required',
        "qty" => "required|numeric|gt:0",
        "type" => "required|in:IN,OUT",
    ],[
        'produk_id.required' => 'Produk harus diisi',
        'produk_id.numeric' => 'Produk tidak sesuai',
        'qty.required' => 'Kuantitas harus diisi',
        'qty.numeric' => 'Kuantitas tidak sesuai',
        'qty.gt' => 'Kuantitas tidak sesuai',
        'type.required' => 'Tipe harus diisi',
        'type.in' => 'Tipe tidak sesuai'
    ]);
    if($validator->fails()){
        return $this->errorJsonMessage('Error validasi',422,$validator->getMessageBag());
    }
```

Gambar 3.26. Validasi *input* pada *controller* penyesuaian

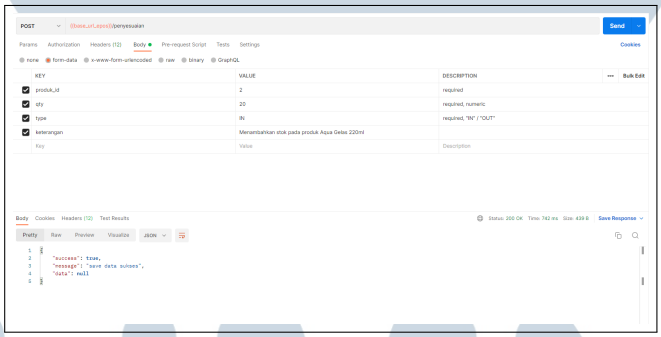
Untuk melakukan proses ini hanya perlu mengisi produk yang ingin disesuaikan dengan kuantitas atau jumlah suatu produk maupun tipe penyesuaian. Seperti pada gambar 3.27 penyesuaian akan tersimpan pada *database* untuk kepentingan *report* dan transaksi produk akan tersimpan pada mutasi produk.



```
try{
    DB::beginTransaction();
    $penyesuaian = new Penyesuaian();
    $penyesuaian->no_transaksi = Penyesuaian::generateNoTransaksi();
    $penyesuaian->user_id = $request->user()->id;
    $penyesuaian->cabang_id = Auth::user()->cabang[0]->cabang_id;
    $penyesuaian->produk_id = $request->produk_id;
    $penyesuaian->tanggal = Carbon::now();
    $penyesuaian->qty = $request->qty;
    $penyesuaian->type = $request->type;
    $penyesuaian->keterangan = $request->keterangan;
    $penyesuaian->save();

    $stok = StokProduk::where([['produk_id',$request->produk_id],['cabang_id',Auth::user()->cabang[0]->cabang_id]])->first();
    $qty = ($request->type == 'IN') ? $request->qty : ($request->qty * -1);
    $stok->qty += $qty;
    $stok->save();

    $harga = harga_jual::where([['produk_id',$request->produk_id],['status',1]])->first();

    $mutasi_produk = new mutasi_produk;
    $mutasi_produk->produk_id = $request->produk_id;
    $mutasi_produk->cabang_id = Auth::user()->cabang[0]->cabang_id;
    $mutasi_produk->nomor_transaksi = $penyesuaian->no_transaksi;
    $mutasi_produk->tanggal = Carbon::now();
    $mutasi_produk->qty = ($request->type == 'IN')? $request->qty:($request->qty*-1);
    $mutasi_produk->cost = $harga->cost;
    $mutasi_produk->tipe_transaksi = 'Penyesuaian';
    $penyesuaian->mutasi()->save($mutasi_produk);
    DB::commit();
    return $this->successJsonResponse(null, 'save data sukses');
}catch (Exception $e){
    DB::rollBack();
    return $this->errorJsonMessage($e->getMessage(),422);
}
```

Gambar 3.27. *Controller* penyesuaian

Seperti pada gambar 3.28 merupakan contoh *input* data dan hasil respon dari API.



Gambar 3.28. Hasil dari API Penyesuaian

3.2.3 Kendala yang Ditemukan

Pada saat mengerjakan kasus tersebut penulis menemukan kendala dalam pengerjaannya selama kerja magang. Kendala yang ditemukan antara lain:

1. Tidak familier dalam penggunaan *framework* Laravel

3.2.4 Solusi atas Kendala yang Ditemukan

Namun terdapat solusi dari kendala yang ditemukan penulis selama kerja magang adalah:

1. Mempelajari *framework* Laravel dengan bertanya kepada pembimbing lapangan atau dari website maupun video