

LAMPIRAN

LISTING PROGRAM FUNGSI INTI SISTEM KASIR DIGITAL LINA FLOWERS

Mengingat keterbatasan jumlah halaman dan untuk menjaga kerapian dokumen skripsi, keseluruhan baris kode program (*source code*) dari Sistem Kasir Digital Lina Flowers tidak dilampirkan secara keseluruhan. Pada lampiran ini hanya ditampilkan potongan kode dari fungsi-fungsi inti yang mewakili proses utama sistem.

Sistem dibangun menggunakan *framework* Next.js (React) dengan bahasa TypeScript, serta basis data PostgreSQL yang diakses melalui *ORM* Prisma. Adapun fungsi-fungsi inti yang ditampilkan meliputi autentikasi pengguna, perhitungan harga satuan, pengelolaan keranjang belanja, penyimpanan transaksi, rekapitulasi laporan penjualan, manajemen produk, serta penomoran nota.

Berikut ditampilkan potongan kode dari masing-masing fungsi inti tersebut:

1. Autentikasi Pengguna (Login)
2. Perhitungan Harga per Satuan
3. Format Tampilan Jumlah & Satuan
4. Identitas Baris Keranjang
5. Menambah Produk ke Keranjang
6. Menyimpan Transaksi (Checkout Server)
7. Proses Checkout (Halaman POS)
8. Rekap Laporan Penjualan
9. Menambah Produk Baru
10. Penomoran Nota per Kode Pelanggan

Seluruh kode program tersebut dapat diakses dan diunduh, secara publik melalui tautan repositori Google Drive berikut ini:



https://drive.google.com/drive/folders/1InqnEtKOiLeCPN0KCvQEVhLkNc5NddSn?usp=drive_link

LIST PROGRAM FUNGSI INTI - SISTEM KASIR LINA FLOWERS

1. AUTENTIKASI PENGGUNA (LOGIN)

File:
app/(backend)/api/login/route.ts
Keterangan: Memverifikasi
username & password
menggunakan enkripsi
bcrypt, lalu membuat sesi
login berupa cookie httpOnly.

```
export async function
POST(request: Request) {
  const { username, password,
rememberMe } = await
request.json();
  const cleanUsername =
String(username || "").trim();

  // Ambil data user berdasarkan
username (tidak peka huruf
besar/kecil)
  const rows = await
prisma.$queryRaw<LoginUser
Row[]>`
  SELECT id, username,
"fullName", "profilePhoto",
password, role
  FROM "User" WHERE
LOWER(username) =
LOWER(${cleanUsername})
LIMIT 1`;
  const user = rows[0];
  if (!user)
    return NextResponse.json({
error: "Username tidak
ditemukan!" }, { status: 404 });

  // Verifikasi password:
bandingkan input dengan hash
```

```
bcrypt di database
  const isBcryptHash =
/^$2[aby]\$d{2}\$/.test(user.pa
ssword);
  const isPasswordValid =
isBcryptHash
  ? await
bcrypt.compare(password,
user.password)
  : password ===
user.password;
  if (!isPasswordValid)
    return NextResponse.json({
error: "Password salah!" }, {
status: 401 });
```

```
// Buat sesi (cookie httpOnly)
lalu kirim data user ke client
  const response =
NextResponse.json({
  id: user.id, username:
user.username,
  fullName: user.fullName ||
user.username, role: user.role,
});
  response.cookies.set(
serverSessionCookie.name,
```

```
createServerSessionToken(user.i
d, Boolean(rememberMe)),
  { httpOnly: true, sameSite:
"lax", secure: true, path: "/" }
);
  return response;
}
```

2. PERHITUNGAN HARGA PER SATUAN

File: lib/satuan.ts
Keterangan: Menghitung
harga suatu satuan pesan
(pcs/lusin/1/2 gross/
gross) dari harga dasar

produk. Contoh: 280.000/gross -
 > satuan lusin
 = 280.000 x 12 / 144 =
 23.333.

```
// Jumlah pcs untuk tiap satuan
(1 gross = 144 pcs, 1/2 gross =
72, 1 lusin = 12)
export const PCS_PER_UNIT =
{ pcs: 1, lusin: 12,
setengah_gross: 72, gross: 144
};
```

```
export function
hitungHargaSatuan(
  hargaBase: number,
  satuanHarga: string,
  satuanPesan: string
): number {
  const perHarga =
PCS_PER_UNIT[satuanHarga]
?? 1;
  const perPesan =
PCS_PER_UNIT[satuanPesan]
?? 1;
  return Math.round((hargaBase
* perPesan) / perHarga);
}
```

3. FORMAT TAMPILAN JUMLAH & SATUAN

File: lib/satuan.ts
 Keterangan: Menampilkan
 jumlah + satuan secara rapi.
 Khusus 1/2 gross
 dihitung sebagai kelipatan
 gross penuh: jumlah 3 -> "1 1/2
 Gross",
 jumlah 5 -> "2 1/2 Gross".

```
export function
formatQtySatuan(jumlah:
number, satuanHarga: string):
```

```
string {
  const key = satuanHarga ||
"pcs";
  if (key === "setengah_gross")
  {
    const units = Math.max(0,
Math.round(Number(jumlah) ||
0)); // banyaknya 1/2 gross
    const gross =
Math.floor(units / 2);
    const half = units % 2;
    const num = `${gross > 0 ?
gross : ""}${half ? "1/2" : ""}` ||
"0";
    return `${num} Gross`;
  }
  const label =
SATUAN_LABELS[key] ??
"Pcs";
  return `${jumlah} ${label}`;
}
```

4. IDENTITAS BARIS KERANJANG

File: lib/satuan.ts
 Keterangan: Membuat ID unik
 tiap baris keranjang dari
 kombinasi produk
 + variasi + satuan + kode
 pelanggan, sehingga produk
 sama dengan
 satuan/kode berbeda menjadi
 baris terpisah.

```
export function
computeCartRowId(
  productId: number, variantId?:
number | null,
  satuanPesan?: string | null,
  label?: string | null
): string {
```

```

const v = variantId ?
Number(variantId) : 0;
const s = satuanPesan ?? "pcs";
const l = (label ??
"").trim().toUpperCase();
return
`${product.id}:${v}:${s}:${l}`;
}

```

5. MENAMBAH PRODUK KE KERANJANG

File: lib/store.ts (Zustand)
 Keterangan: Menambahkan produk ke keranjang. Jika baris sama sudah ada, hanya menambah jumlah (dengan pengecekan stok); jika belum, membuat baris baru beserta harga terhitung.

```

addToCart: (product) => {
  const cart = get().cart;
  const satuanPesan =
product.satuanPesan ??
product.satuanHarga ?? "pcs";
  const hargaBase =
product.hargaBase ??
product.harga;
  const hargaBaseAsli =
product.hargaBaseAsli ??
hargaBase; // harga katalog asli
  const harga =
hitungHargaSatuan(hargaBase,
product.satuanHarga ?? "pcs",
satuanPesan);
  const rowId =
computeCartRowId(product.id,
product.variantId, satuanPesan,
product.label);
  const existing =
cart.find((item) => item.id ===

```

```

rowId);

  if (existing) {
    // Baris sama -> cukup
    tambah jumlah (tidak melebihi
    stok)
    if (existing.quantity <
product.stok) {
      set({ cart: cart.map((i) =>
        i.id === rowId ? { ...i,
        quantity: i.quantity + 1 } : i ));
    } else {
      alert('Stok tidak cukup! Sisa
      stok: ${product.stok}');
    }
  } else if (product.stok > 0) {
    // Buat baris baru
    set({ cart: [...cart, {
      ...product, id: rowId,
      productId: product.id,
      harga, hargaBase,
      hargaBaseAsli, satuanPesan,
      quantity: 1,
    }] });
  },
},

```

6. MENYIMPAN TRANSAKSI (CHECKOUT SERVER)

File:
 app/(backend)/api/transaksi/route.ts

Keterangan: Menyimpan transaksi baru: mengubah tiap item keranjang menjadi baris transaksi (snapshot harga & jumlah), menghitung total, lalu mengurangi stok produk.

```

// Ubah item keranjang -> baris
transaksi (menyimpan snapshot

```

```

harga, jumlah, kode)
const mapCartToItems = (cart)
=> cart.map((item) => ({
  productId:
Number(item.productId ??
item.id),
  variantId: item.variantId !==
null ? Number(item.variantId) :
null,
  variantName:
item.variantName ?? null,
  label: item.label ?? null,
// kode pelanggan (Aneka)
  basePrice:
Number(item.harga),
  jumlah:
Number(item.quantity),
  subtotal: Number(item.harga)
* Number(item.quantity),
  satuanHarga: item.satuanPesan
|| "pcs",
})));

```

```

export async function
POST(request: Request) {
  const body = await
request.json();
  const { cart,
metode_pembayaran,
nama_pembeli, nama_kasir,
status } = body;
  const total_harga =
cart.reduce(
  (t, i) => t + Number(i.harga)
* Number(i.quantity), 0);

```

```

// Simpan transaksi + seluruh
itemnya
const trx = await
prisma.transaction.create({
  data: {
    trxNumber: await
getNextTrxNumber(),

```

```

total_harga,
metode_pembayaran,
nama_pembeli, nama_kasir,
status: status || "Paid",
items: { create:
mapCartToItems(cart) },
},
include: { items: true },
});

```

```

// Kurangi stok tiap produk
yang terjual
for (const item of cart) {
  await
prisma.product.update({
    where: { id:
Number(item.productId ??
item.id) },
    data: { stok: { decrement:
Number(item.quantity) } },
  });
}
return NextResponse.json(trx,
{ status: 201 });
}

```

7. PROSES CHECKOUT (HALAMAN POS)

File:
app/(frontend)/pos/page.tsx
Keterangan: Memvalidasi
keranjang & nama pembeli,
mengirim data
transaksi ke server, lalu
mengosongkan keranjang bila
berhasil.

```

const processCheckout = async
() => {
  if (cart.length === 0) return
alert("Keranjang masih
kosong!");
  if (!namaPembeli.trim()) return

```

```

alert("Nama pembeli harus
diisi!");

const status =
metodePembayaran ===
"Belum Bayar" ? "Unpaid" :
"Paid";
const res = await
fetch("/api/transaksi", {
  method: "POST",
  headers: { "Content-Type":
"application/json" },
  body: JSON.stringify({
    cart,
    metode_pembayaran:
metodePembayaran,
    nama_pembeli:
namaPembeli.toUpperCase(),
    nama_kasir:
cashierDisplayName?.toUpperCase(),
    status, status_pengiriman:
"Sedang Disiapkan",
  }),
});

if (res.ok) {
  await clearSavedPosCart();
  alert(`Transaksi atas nama
${namaPembeli} Berhasil!`);
  clearCart();
  setNamaPembeli("");
  setIsSessionStarted(false);
} else {
  alert("Gagal memproses
transaksi.");
}
};

```

8. REKAP LAPORAN PENJUALAN

File:
app/(backend)/api/laporan/route

.ts

Keterangan: Mengambil
semua barang terjual dari
transaksi berstatus
Paid pada tanggal tertentu,
mengelompokkan per produk,
lalu mengurutkan
dari yang paling laris.

```

export async function
GET(request: Request) {
  const dateParam = new
URL(request.url).searchParams.
get("date");
  const tanggal = dateParam ??
new
Date().toISOString().split("T")[
0];
  const startDate = new
Date(tanggal +
"T00:00:00.000Z");
  const endDate = new
Date(tanggal +
"T23:59:59.999Z");

```

```

// Ambil semua item terjual
dari transaksi LUNAS pada
tanggal tsb
const terjual = await
prisma.transactionItem.findMan
y({
  where: {
    transaction: { tanggal: { gte:
startDate, lte: endDate }, status:
"Paid" },
  },
  include: { product: true },
});

```

```

// Kelompokkan per produk
(gabungkan jumlah &
pendapatan)

```

```

const rekap = {};
terjual.forEach((item) => {
  const id = item.productId;
  if (!rekap[id]) rekap[id] = {
    id, nama_produk:
item.product.nama_produk,
    jumlah_terjual: 0,
total_pendapatan: 0,
  };
  rekap[id].jumlah_terjual +=
item.jumlah;
  rekap[id].total_pendapatan
+= item.subtotal;
});

// Urutkan dari yang paling
laris
const hasil =
Object.values(rekap).sort((a, b)
=> b.jumlah_terjual -
a.jumlah_terjual);
return
NextResponse.json(hasil);
}

```

9. MENAMBAH PRODUK BARU

File:
app/(backend)/api/produk/routes

Keterangan: Menyimpan produk baru beserta variasinya (bila ada).

Barcode dibuat otomatis jika dikosongkan.

```

export async function
POST(request: Request) {
  const data = await
request.json();

```

// Barcode otomatis bila tidak

```

diisi
  let barcode = data.barcode;
  if (!barcode || barcode.trim()
=== "")
    barcode = `LINA-
${Date.now().toString().slice(-
6)}`;

  const produk = await
prisma.product.create({
    data: {
      nama_produk:
data.nama_produk,
      harga: Number(data.harga),
      satuanHarga:
data.satuanHarga || "pcs",
      stok: Number(data.stok),
      barcode,
      gambar: data.gambar || null,
      // Buat variasi bila dikirim
(mis. ukuran S/M/L)
      variants:
Array.isArray(data.variants)
? { create: data.variants
      .filter((v) => v.name
&& v.name.trim())
      .map((v, idx) => ({
        name: v.name.trim(),
        priceModifier:
Number(v.priceModifier ?? 0),
        order: idx,
      })) }
      : undefined,
    },
    include: { variants: true },
  });
  return
NextResponse.json(produk, {
status: 201 });
}

```

10. PENOMORAN NOTA PER KODE PELANGGAN

File:

app/(frontend)/penjualan/page.ts

x

Keterangan: Fitur khusus
(Toko Aneka):
mengelompokkan item per kode
pelanggan dan memberi
nomor nota (1)(2)(3), lalu
menyusun nama tampil
"(1) LN 13 AMN".

```
-----  
const CIRCLED = [""  
  "(1)","(2)","(3)","(4)","(5)","(6)"  
  ","(7)","(8)","(9)","(10)"];
```

```
// Peta kode pelanggan ->  
nomor nota,urut kemunculan  
pertama  
const buildNotaMap = (items,  
startNo = 1) => {  
  const map = new Map();  
  let n = Math.max(1,  
Math.floor(startNo)) - 1;  
  for (const it of items) {
```

```
    const code = (it.label ||  
it.variantName || "").trim();  
    if (code && !map.has(code))  
      map.set(code, ++n);  
  }  
  return map;  
};
```

```
// Susun nama tampil: "(1) LN  
13 AMN" (nomor + kode  
produk + kode pelanggan)  
const anekaItemName = (item,  
notaMap) => {  
  const code =  
extractKode(item.product?.nam  
a_produk); // "LN 13"  
  const code = (item.label ||  
item.variantName || "").trim();  
  const no = code ?  
notaMap.get(code) : undefined;  
  return `${no ? CIRCLED[no]  
+ " " : ""}${kode}${code ? " " +  
code : ""}`;  
};
```