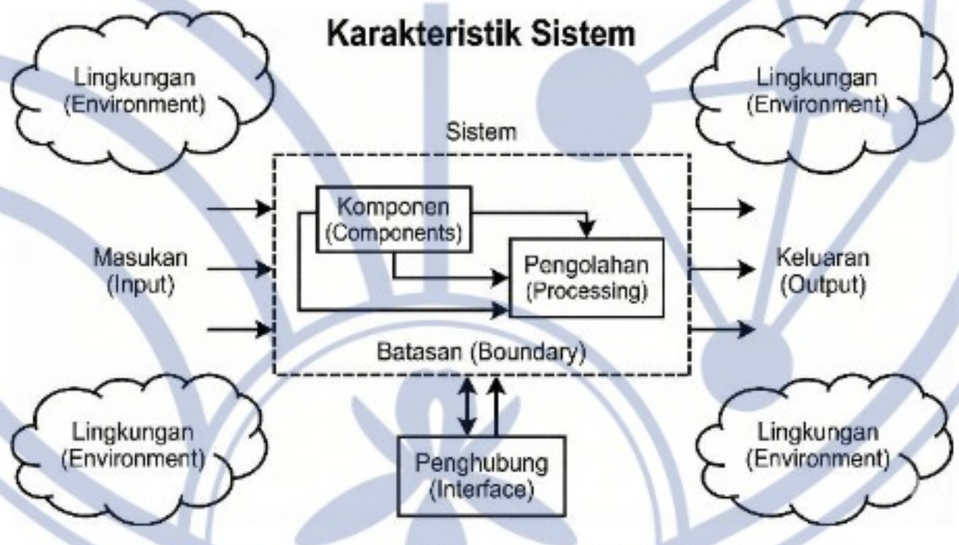
BAB II
KAJIAN LITERATUR

2.1 Konsep Dasar Sistem Informasi

Dalam pengembangan sebuah tekno*log*i tepat guna, pemahaman mendalam mengenai konsep dasar pembentuk sistem informasi sangat diperlukan. Sub-bab ini akan menguraikan definisi sistem, informasi, dan kesatuan sistem informasi secara terperinci.

2.1.1 Pengertian Sistem

Sistem didefinisikan sebagai sekumpulan elemen atau komponen yang saling berinteraksi, terintegrasi, dan bekerja sama dalam suatu batasan tertentu untuk mencapai tujuan yang sama [8]. Secara fundamental, sebuah sistem menerima *input* (masukan) berupa sumber daya data, memprosesnya melalui prosedur tertentu, dan menghasilkan *output* (keluaran) yang berguna bagi penggunanya [2]. Karakteristik utama yang harus dimiliki sistem meliputi komponen (*components*), batasan (*boundary*), lingkungan (*environment*), penghubung (*interface*), masukan (*input*), pengolahan (*processing*), dan keluaran (*output*) [8]. Sinergi antar komponen ini sangat penting agar operasional bisnis, seperti pada sebuah kafe, dapat berjalan secara otomatis dan terstruktur.



Gambar 2. 1 Karakteristik Sistem

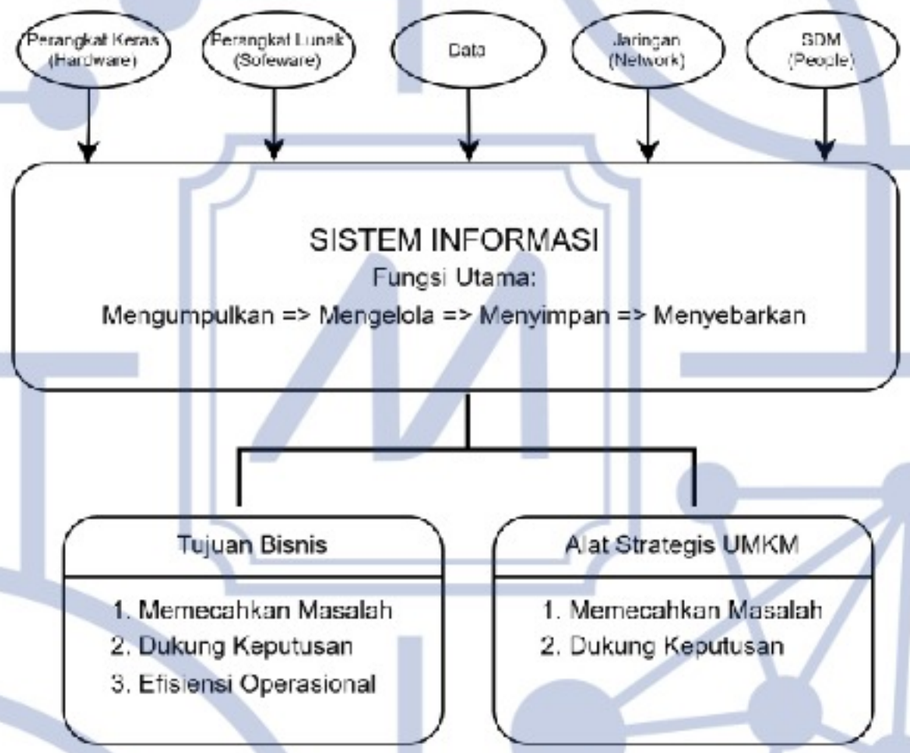
2.1.2 Pengertian Informasi

Informasi merupakan hasil pengolahan data yang telah diklasifikasikan, diinterpretasikan, dan diorganisasikan sedemikian rupa sehingga memiliki makna dan nilai bagi penerimanya [9]. Informasi berbeda dengan data mentah; informasi adalah data yang telah diberi konteks. Kualitas informasi sangat ditentukan oleh tiga dimensi utama: akurasi (bebas dari kesalahan), ketepatan waktu (tersedia saat dibutuhkan), dan relevansi

(bermanfaat untuk situasi yang dihadapi) [10]. Informasi yang berkualitas menjadi aset vital dalam mendukung operasional harian maupun strategi jangka panjang perusahaan.

2.1.3 Pengertian Sistem Informasi

Sistem informasi adalah kombinasi terorganisir dari perangkat keras (*Hardware*), perangkat lunak (*Software*), infrastruktur jaringan, data, dan sumber daya manusia yang bekerja sama untuk mengumpulkan, mengolah, menyimpan, dan menyebarkan informasi dalam suatu organisasi [1]. Sistem ini bertujuan untuk memecahkan masalah bisnis, mendukung pengambilan keputusan manajerial, serta meningkatkan efisiensi operasional melalui otomatisasi proses kerja [5]. Dalam konteks UMKM, sistem informasi berfungsi sebagai alat strategis untuk meningkatkan daya saing dan efisiensi pelayanan [4].



Gambar 2. 2 Proses Kerja Sistem Informasi

2.2 Sistem *Point of Sales* (POS)

Pemahaman mendalam mengenai sistem transaksi merupakan landasan utama dalam penelitian ini. Sebagai instrumen yang berada di garis depan pelayanan, sistem ini memegang peranan vital dalam mengintegrasikan data operasional menjadi informasi manajerial yang berharga.

2.2.1. Konsep *Point of Sales* (POS)

Point of Sales (POS) adalah sistem yang digunakan untuk memproses transaksi penjualan secara *real-time* di titik layanan, menggantikan fungsi mesin kasir konvensional [1]. Dalam perkembangannya, POS tidak hanya berfungsi sebagai alat pencatat transaksi, tetapi telah berevolusi menjadi sistem manajemen bisnis terintegrasi yang mencakup pengelolaan stok, laporan keuangan, manajemen pelanggan, dan analisis laba-rugi [6].

Penerapan aplikasi POS berbasis web memberikan fleksibilitas tinggi karena data tersimpan di *cloud* dan dapat diakses dari berbagai perangkat, memungkinkan pemilik usaha memantau performa penjualan dari jarak jauh tanpa harus berada di lokasi fisik toko [4]. Sistem ini terbukti mampu meningkatkan efisiensi dengan mengotomatisasi perhitungan harga, mengurangi kesalahan pencatatan manual *(human error)*, dan mempercepat proses transaksi [11].

Secara termino*log*i dasar, *Point of Sales* (POS) didefinisikan sebagai titik pertemuan antara penjual dan pembeli di mana transaksi pembayaran diselesaikan. Namun, dalam konteks sistem informasi manajemen, definisi POS telah meluas menjadi sebuah arsitektur sistem yang mengintegrasikan perangkat keras dan perangkat lunak untuk merekam arus transaksi bisnis. POS modern bukan lagi sekadar alat substitusi mesin kasir konvensional, melainkan sebuah ekosistem digital yang mampu mengolah data transaksi menjadi informasi yang bernilai bagi manajemen operasional perusahaan [12]

Evolusi tekno*log*i informasi telah mengubah paradigma POS dari alat pencatat pasif menjadi sistem manajemen aktif. Sistem POS kini memiliki peran ganda: sebagai alat operasional di garda depan (*front-end*) untuk melayani pelanggan, dan sebagai alat manajerial di sisi belakang (*back-end*) untuk pengelolaan basis data [13]. Fungsi ini mencakup integrasi otomatis antara penjualan dengan pengurangan stok inventaris, pengelolaan data pelanggan (*customer relationship management*), serta penyusunan laporan keuangan dasar secara *real-time*. Integrasi ini dirancang untuk meminimalisir *human error* yang sering terjadi pada pencatatan manual.

Lebih lanjut, perkembangan infrastruktur internet telah melahirkan konsep POS berbasis web atau *cloud computing*. Berbeda dengan sistem *legacy* (berbasis instalasi desktop lokal), konsep POS berbasis web menawarkan fleksibilitas arsitektur data [14]. Karakteristik utama dari POS berbasis web adalah sentralisasi data pada server internet (*cloud*) [15]. Secara teoritis, model ini memungkinkan data transaksi diakses secara *ubiquitous* (di mana saja dan kapan saja) melalui peramban (*browser*), sehingga memutus ketergantungan manajerial terhadap kehadiran fisik di lokasi usaha untuk melakukan pengawasan kinerja bisnis.

2.2.2 Manfaat POS dalam Bisnis

Implementasi sistem *Point of Sales* (POS) dalam operasional bisnis memberikan dampak yang lebih luas daripada sekadar alat pembayaran. Penggunaan POS secara efektif dapat mentransformasi manajemen usaha dari pola tradisional menuju pola digital yang

terukur [15]. Manfaat fundamental dari penerapan sistem POS bagi keberlangsungan bisnis, yaitu [12]:

1. Peningkatan efisiensi operasional dan akurasi data manfaat utama dari sistem POS adalah otomatisasi proses yang sebelumnya dilakukan secara manual. Dalam sistem konvensional, risiko kesalahan manusia (*human error*) seperti kesalahan perhitungan total belanja atau kembalian sangat tinggi. POS memitigasi risiko ini melalui kalkulasi otomatis sistem, yang secara langsung meningkatkan kecepatan layanan (*service speed*) di kasir [12]. Hal ini berdampak pada pengurangan antrean pelanggan dan peningkatan volume transaksi yang dapat diproses dalam satu satuan waktu.
2. Pengendalian inventaris secara *real-time* salah satu tantangan terbesar dalam manajemen ritel adalah ketidaksesuaian data stok fisik dengan catatan pembukuan [13]. Sistem POS modern menawarkan solusi melalui integrasi data penjualan dan persediaan. Setiap kali transaksi terjadi, sistem secara otomatis memotong jumlah stok dalam basis data. Mekanisme ini memungkinkan pemilik bisnis untuk memantau ketersediaan barang secara *real-time*, mendeteksi penyusutan stok (*shrinkage*) akibat pencurian atau kerusakan lebih dini, serta merencanakan pengadaan barang (*restocking*) secara tepat waktu tanpa perlu menunggu perhitungan stok manual (*opname*) di akhir periode.
3. Pendukung keputusan berbasis data (*Data-driven decision making*) Selain fungsi operasional, POS juga memiliki fungsi strategis sebagai penyedia data analitik. POS mampu mengolah data transaksi mentah menjadi laporan manajerial yang komprehensif, seperti laporan penjualan harian, analisis produk terlaris (*best-selling items*), hingga pola jam sibuk toko [15]. Ketersediaan data historis yang akurat ini sangat krusial bagi manajemen untuk merumuskan strategi promosi yang efektif, menentukan harga jual, dan mengevaluasi kinerja bisnis secara objektif berdasarkan data, bukan sekadar asumsi.

2.2.3 Manajemen *Shift* dan Laporan *Z-Read*

Dalam operasional bisnis yang menggunakan pergantian sesi kerja (*shift*), efektivitas dan akuntabilitas penerimaan kas sangat bergantung pada sistem *Point of Sale* (POS) yang terkomputerisasi. Sistem POS memfasilitasi setiap pergantian *shift* melalui mekanisme *role-based access*, di mana kasir diwajibkan mencatat modal awal sebelum memulai transaksi [16]. Hal ini bertujuan sebagai batas kendali untuk mencocokkan akumulasi penerimaan sistem dengan uang fisik.

Lebih lanjut, dalam siklus sistem informasi akuntansi, mekanisme penutupan sesi atau yang lazim dikenal dengan istilah *Z-Report (End of Day)* menjadi prosedur wajib. Pada tahap ini, sistem POS akan mengunci total transaksi berjalan, mengakumulasikan seluruh pendapatan harian secara otomatis, dan mereset perhitungan untuk *shift* selanjutnya [17]. Prosedur rekonsiliasi akhir sesi ini meminimalisir manipulasi data secara manual serta memastikan bahwa ringkasan kas yang disetorkan oleh kasir tervalidasi secara *real-time* oleh sistem.

2.2.4 *Void* Transaksi dan *Audit trail*

Kesalahan input atau retur dari pelanggan merupakan risiko operasional yang lumrah, sehingga sistem *Point of Sale* (POS) harus dilengkapi dengan fitur pembatalan transaksi (*Void*). Namun, dalam prinsip pengendalian internal basis data, transaksi yang telah terekam dilarang keras untuk dihapus secara permanen (*hard delete*). Sebagai gantinya, sistem menggunakan metode *soft delete* yang menonaktifkan status transaksi namun tetap menyimpan riwayatnya secara utuh dalam peladen.

Pentingnya jejak digital atau *audit trail* (*log aktivitas*) dalam sistem POS keuangan. Setiap kali tindakan *Void* dieksekusi, sistem diwajibkan secara otomatis merekam identitas kasir yang membatalkan, keterangan waktu (*timestamp*), serta alasan spesifik (*void reason*) dilakukannya pembatalan tersebut [18]. Ketersediaan *log* aktivitas ini memberikan transparansi bagi pengelola (*owner*) untuk melacak aktivitas operasional yang anomali dan mencegah celah kecurangan (*fraud*) pada uang kas.

2.3 Konsep *E-business* pada Layanan Cafe

Penerapan konsep *electronic business* (*E-business*) pada sektor layanan kafe dimaknai sebagai integrasi menyeluruh tekno*log*i informasi ke dalam rantai proses bisnis, yang tidak hanya terbatas pada transaksi pembayaran elektronik tetapi juga mencakup digitalisasi manajemen operasional dari sisi pemesanan hingga pelaporan penjualan [4]. Transformasi ini bertujuan untuk menciptakan ekosistem kerja yang terotomatisasi guna meminimalisir inefisiensi dan risiko kesalahan pengelolaan data menu serta pesanan yang sering terjadi pada metode konvensional [19]. Dalam perkembangannya, implementasi sistem berbasis web menjadi standar baru yang memungkinkan sinkronisasi data secara *real-time* antara unit layanan, sehingga memberikan aksesibilitas penuh bagi manajemen untuk memantau kinerja bisnis tanpa batasan ruang dan waktu [3].

2.3.1 *E-business*

Electronic business (*E-business*) didefinisikan sebagai penggunaan tekno*log*i internet dan digital untuk bekerja dan memberdayakan proses bisnis, perdagangan elektronik (*E-commerce*), dan kolaborasi dengan mitra bisnis [20]. Penerapan sistem POS dan pemesanan digital di kafe merupakan bentuk transformasi menuju *e-business* yang bertujuan meningkatkan efisiensi operasional dan transparansi data.

Electronic business (*E-business*) memiliki cakupan yang jauh lebih luas dibandingkan sekadar perdagangan elektronik *(e-commerce*). Jika *e-commerce* hanya berfokus pada transaksi jual-beli eksternal (penjualan produk via internet), maka *e-business* adalah payung besar yang mencakup digitalisasi seluruh proses bisnis organisasi.

E-business didefinisikan sebagai transformasi proses bisnis yang memanfaatkan tekno*log*i digital untuk menghubungkan sistem internal perusahaan, pelanggan, dan mitra bisnis guna meningkatkan efisiensi operasional [21]. *E-business* melibatkan desain ulang proses bisnis (*business process re-engineering*) di mana tekno*log*i informasi digunakan tidak hanya untuk transaksi komersial, tetapi juga untuk kolaborasi internal, manajemen sumber daya manusia, dan pengelolaan rantai pasok [22]

Dalam konteks operasional bisnis modern, membedakan secara tegas bahwa sistem yang mengelola data inventaris, pesanan dapur, dan laporan keuangan secara digital diklasifikasikan sebagai *e-business system*, bukan sekadar *e-commerce* [23]. Hal ini dikarenakan fokus utamanya adalah perbaikan kinerja organisasi (*organizational performance*) dan integrasi data harian, bukan semata-mata memindahkan toko fisik ke toko *Online*.

2.3.2 Proses Bisnis Operasional Kafe (Studi Komparatif)

Proses bisnis inti pada sebuah kafe umumnya melibatkan tiga tahapan utama: pemesanan (*Ordering*), produksi (*kitchen/bar*), dan pembayaran (*payment*) [24]. Studi komparatif pada beberapa kedai kopi menunjukkan bahwa proses bisnis konvensional sering menghadapi hambatan inefisiensi [25]. Hambatan tersebut antara lain: kesalahan pencatatan pesanan, keterlambatan penyampaian pesanan ke dapur, antrian pembayaran yang panjang, dan selisih stok bahan baku akibat pencatatan manual. Oleh karena itu, digitalisasi alur kerja melalui sistem terintegrasi sangat diperlukan untuk menciptakan siklus bisnis yang lebih cepat dan akurat.

Masalah utama yang timbul adalah risiko *human error* dan asimetri informasi, di mana tulisan tangan pramusaji pada nota manual sering kali sulit terbaca atau terjadi

kesalahan input menu. Selain itu, ketiadaan informasi stok yang *real-time* di tangan pramusaji sering memicu situasi di mana pesanan telah dicatat namun bahan baku ternyata kosong, yang berujung pada kekecewaan pelanggan akibat miskomunikasi operasional.

Inefisiensi tersebut kemudian berdampak domino pada tahap produksi di area dapur atau bar, menyoroti adanya masalah latensi informasi pada sistem manual, yaitu jeda waktu yang terbuang antara saat pesanan dicatat dengan saat kertas pesanan fisik diserahkan ke dapur [13]. Selain itu, pengelolaan antrean pesanan yang tidak terorganisir seperti nota yang menumpuk atau hilang sering mengakibatkan kegagalan penerapan prinsip *First-In-First-Out (FIFO)*, sehingga terjadi keterlambatan penyajian yang signifikan. Siklus ini berakhir pada tahap pembayaran yang berfungsi sebagai validasi transaksi sekaligus rekonsiliasi stok.

Pada tahap akhir ini, sistem kasir manual atau semi-otomatis sering kali menciptakan *bottleneck* atau antrean panjang karena proses input harga yang lambat dan rentan salah [15]. Lebih jauh lagi, terpisahnya data penjualan dengan data inventaris menyebabkan pemilik usaha kesulitan memantau penyusutan stok (*shrinkage*), sehingga sering terjadi selisih kas dan ketidaksesuaian jumlah barang fisik di akhir periode. Oleh karena itu, digitalisasi seluruh alur kerja melalui sistem *e-business* terintegrasi menjadi solusi mutlak untuk menjamin kecepatan layanan serta akurasi data dari meja pesanan hingga pelaporan keuangan.

2.3.3 Laporan Pendapatan

Laporan pendapatan *(revenue report)* dalam sistem *Point of Sales* (POS) didefinisikan sebagai instrumen rekapitulasi keuangan yang merekam arus kas masuk dari aktivitas penjualan secara sistematis. Penerapan tekno*log*i ini pada UMKM menjadi solusi strategis untuk mengatasi kelemahan pencatatan manual yang rentan terhadap *human error* dan ketidakakuratan data [26]. Dengan adanya laporan pendapatan otomatis, pelaku usaha dapat memantau posisi omset harian secara *real-time*, yang menjadi indikator utama kesehatan finansial bisnis tanpa harus melalui proses audit manual yang memakan waktu [27].

Selain fungsi *monitoring*, laporan pendapatan berperan vital sebagai alat evaluasi kinerja bisnis yang lebih efisien dibandingkan sistem manajemen persediaan bahan baku yang kompleks. Data yang dihasilkan memungkinkan pemilik usaha untuk mengidentifikasi tren produk terlaris (*best-seller*) guna menentukan strategi pemasaran yang tepat sasaran [28]. Hal ini menegaskan bahwa prioritas sistem POS bagi usaha kuliner

adalah akurasi *capture* data penjualan dan kecepatan evaluasi pendapatan untuk mendukung pengambilan keputusan operasional.

2.4 Basis Data

Basis data (*database*) adalah kumpulan data yang terorganisir secara sistematis dan saling berhubungan yang disimpan dalam media penyimpanan elektronik agar dapat diakses, dikelola, dan diperbarui dengan mudah [29]. Struktur dasar basis data relasional melibatkan pengelompokan data ke dalam tabel-tabel dua dimensi yang terdiri dari baris (*record*) dan kolom (*field*). Untuk merancang struktur basis data yang baik, digunakan model ERD (*Entity Relationship Diagram*). ERD adalah model konseptual yang mendeskripsikan struktur data dan hubungan antar data tersebut menggunakan serangkaian simbol grafis [29]. Komponen utama penyusun ERD meliputi:

Dalam pengembangan sistem informasi transaksi seperti *Point of Sales*, pengelolaan data tidak hanya sekadar penyimpanan, melainkan melibatkan strukturisasi informasi yang kompleks. Basis data adalah koleksi data yang saling berelasi (*inter-related* data) yang diorganisasikan sedemikian rupa agar dapat diambil kembali dengan cepat dan mudah [30].

Secara spesifik, sistem ini menerapkan konsep basis data relasional (RDBMS). Dalam konsep ini, data disimpan dalam bentuk tabel-tabel dua dimensi yang terdiri dari baris (*record*) dan kolom (*field*). Penggunaan model relasional bertujuan untuk meminimalisir redudansi (pengulangan data) dan menjaga konsistensi data. Misalnya, data pelanggan hanya perlu disimpan satu kali dalam tabel master pelanggan, namun dapat dipanggil berkali-kali dalam tabel transaksi penjualan melalui kunci relasi (*Foreign key*), sehingga efisiensi penyimpanan dapat tercapai) [31].

2.5 ERD (Entity *Relationship Diagram*)

Entity relationship diagram (ERD) adalah instrumen pemodelan data yang digunakan untuk merepresentasikan struktur *log*is *database* [14]. ERD berfungsi sebagai jembatan komunikasi antara perancang sistem dan pengguna untuk memvisualisasikan bagaimana data saling berinteraksi. Dalam penyusunannya, ERD melibatkan tiga komponen fundamental dan aturan kardinalitas:

1. Entitas (*Entity*): Objek vital dalam sistem yang datanya perlu disimpan, digambarkan dengan persegi panjang (Contoh: Barang, Kasir, *Supplier*).

2. Atribut (*Attribute*): Properti detail yang melekat pada entitas, digambarkan dengan elips (Contoh: Kode Barang, Nama Kasir).
3. Relasi (*Relationship*): Hubungan alamiah yang terjadi antar entitas, digambarkan dengan belah ketupat.
4. Kardinalitas (*Cardinality*): Derajat hubungan antar entitas, yang sangat krusial dalam sistem POS. Misalnya, hubungan *One-to-Many* (1:N) di mana satu pelanggan dapat melakukan banyak transaksi, atau hubungan *Many-to-Many* (M:N) antara pesanan dan menu.

2.6 SQL (*Structured Query Language*)

Structured query language (SQL) adalah bahasa pemrograman khusus yang dirancang untuk mengelola data dalam RDBMS. Berbeda dengan bahasa pemrograman aplikasi (seperti PHP), SQL berfokus pada *log*ika manipulasi data.

Berdasarkan analisis teknis SQL bekerja dengan mengeksekusi pernyataan (*statements*) untuk melakukan operasi CRUD (Create, Read, Update, Delete) [15]. Dalam konteks sistem ini, SQL dibagi menjadi dua kelompok instruksi utama:

1. Data *Definition Language* (DDL): Merupakan kelompok perintah yang mendefinisikan kerangka fisik *database*. Perintah seperti *CREATE TABLE* digunakan untuk membangun struktur penyimpanan awal, sedangkan ALTER digunakan untuk memodifikasi struktur tersebut jika ada perubahan kebutuhan sistem.
2. Data *manipulation language* (DML): Merupakan kelompok perintah untuk mengolah data transaksi sehari-hari. Contoh penerapannya dalam sistem POS adalah penggunaan perintah SELECT dengan fungsi agregat (seperti SUM atau COUNT) untuk menghitung total belanja pelanggan secara otomatis, serta perintah *UPDATE* untuk mengurangi jumlah stok barang secara *real-time* setelah transaksi terjadi.

2.7 Model Basis Data Terpisah (*Separate Database*)

Untuk mendukung *multi-tenancy*, penelitian ini mengadopsi pendekatan *Separate Database* (Basis Data Terpisah). Dalam model ini, setiap *tenant* memiliki basis data fisiknya sendiri meskipun menggunakan basis kode yang sama [7]. Keunggulan model ini meliputi:

Untuk mendukung arsitektur *multi-tenancy* di mana sistem digunakan oleh berbagai mitra kafe secara bersamaan, penelitian ini mengadopsi pendekatan *Separate Database* (Basis Data Terpisah). Dalam model ini, meskipun seluruh pengguna mengakses satu kode

aplikasi yang sama, setiap penyewa (*tenant*) atau kafe memiliki skema basis data fisik yang berdiri sendiri [31]. Pendekatan ini dipilih untuk menjamin isolasi data tingkat tertinggi, memastikan bahwa data transaksi milik satu kafe benar-benar tersegregasi dan tidak dapat diakses oleh kafe lain, sehingga memitigasi risiko kebocoran data antar pengguna yang sering menjadi isu krusial dalam layanan berbasis *cloud*.

Selain aspek privasi, penerapan model basis data terpisah juga memberikan keunggulan signifikan dari sisi manajemen keamanan dan performa sistem. Secara teknis, isolasi ini memudahkan penerapan kebijakan keamanan spesifik, seperti pencadangan (*backup*) dan pemulihan data (*restore*) yang dapat dilakukan per kafe tanpa mengganggu operasional keseluruhan sistem. Lebih lanjut, model ini memungkinkan distribusi beban kerja (*workload*) yang lebih efisien, di mana lonjakan aktivitas transaksi yang tinggi pada satu kafe yang sedang ramai tidak akan membebani kinerja basis data kafe lain secara langsung, sehingga stabilitas sistem tetap terjaga [30].

2.8 Arsitektur Sistem *Multi-tenant dan Client-Server*

Keberhasilan implementasi sebuah sistem berbasis web sangat bergantung pada struktur arsitektur yang mendasarinya. Pemilihan arsitektur yang tepat tidak hanya menentukan bagaimana data didistribusikan antara *server* dan pengguna, tetapi juga bagaimana sistem mampu menangani efisiensi sumber daya saat melayani banyak pengguna secara bersamaan. Berikut adalah landasan teoretis mengenai model arsitektur yang diterapkan dalam pengembangan sistem ini.

2.8.1 Arsitektur *Client-Server* (Konsep *Client* Mandiri)

Arsitektur *client-server* adalah model pendistribusian aplikasi yang membagi beban kerja antara penyedia sumber daya atau layanan, yang disebut *server*, dan peminta layanan, yang disebut klien [32]. Dalam konteks sistem ini, *server* bertindak sebagai pusat penyimpanan data dan *log*ika aplikasi, sedangkan *client* bertindak secara mandiri mengakses layanan tersebut melalui *web browser*. Konsep *client* mandiri memungkinkan setiap kafe beroperasi dengan perangkatnya sendiri tanpa bergantung pada instalasi *server* fisik di lokasi masing-masing, cukup dengan koneksi internet ke *server* pusat.

2.8.2 Konsep *Software* as a Service (SaaS)

Software as a Service (SaaS) adalah model penyampaian perangkat lunak di mana *website* di-*hosting* oleh penyedia layanan dan tersedia bagi pelanggan melalui jaringan,

biasanya internet [33]. Dalam model ini, kafe tidak perlu membeli lisensi perangkat lunak atau mengelola infrastruktur *server*. Mereka cukup menyewa layanan sistem POS dan pemesanan, yang memungkinkan efisiensi biaya operasional dan kemudahan pemeliharaan sistem.

2.8.3 Mekanisme Pertukaran Data *Client-Server*

Meskipun sistem ini beroperasi pada lingkungan server lokal (*localhost*), arsitektur komunikasi data yang dibangun tetap mengadopsi standar efisiensi yang dijelaskan dalam teori komputasi modern [33]. Memaparkan bahwa pemisahan antara pengelolaan data (*backend*) dan antarmuka pengguna (*frontend*) memerlukan format pertukaran data yang ringan dan terstruktur untuk menjaga performa sistem agar tetap optimal.

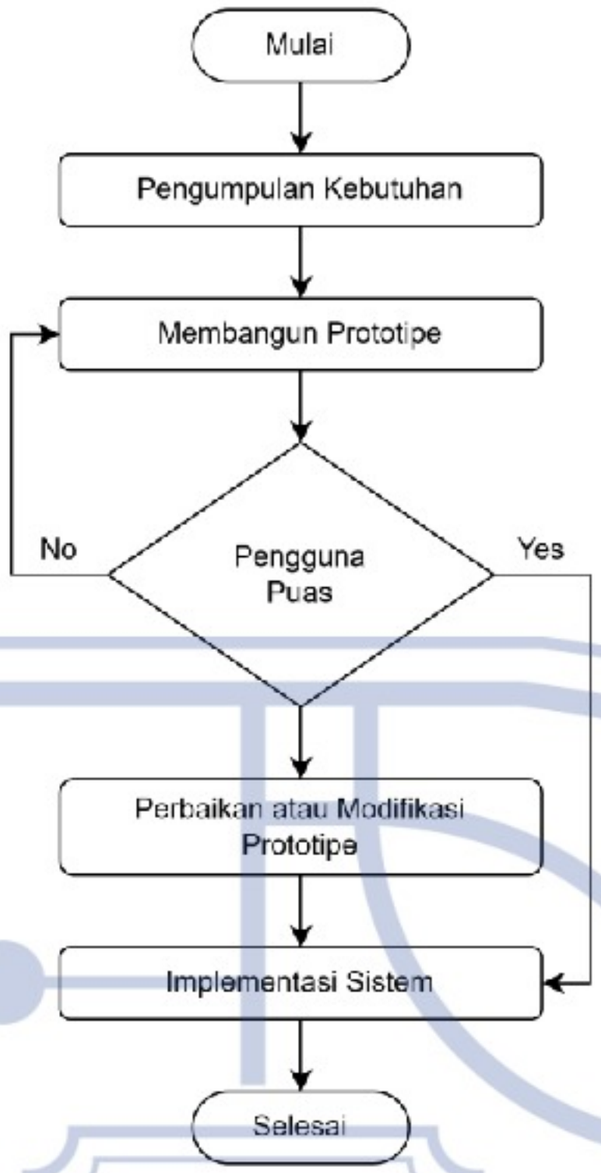
Sejalan dengan hal tersebut, dalam pengembangan sistem ini, mekanisme tersebut diimplementasikan menggunakan format JSON (*JavaScript Object Notation*) melalui jaringan lokal (LAN). Pendekatan ini memungkinkan data transaksi, validasi pengguna, dan pembaruan kata*log* menu dikirimkan dari *server* ke terminal kasir secara *asynchronous* tanpa memerlukan koneksi internet atau integrasi *cloud* eksternal. Dengan demikian, sistem tetap memiliki responsivitas tinggi layaknya aplikasi berbasis web modern, namun dengan kemandirian data penuh pada *server* lokal outlet.

2.8.4 Arsitektur *Multi-tenancy*

Multi-tenancy adalah arsitektur perangkat lunak di mana satu instansi aplikasi berjalan pada server dan melayani banyak penyewa *(tenant)* atau pelanggan secara bersamaan [34]. Dalam konteks sistem ini, setiap *tenant* merepresentasikan satu unit bisnis kafe yang berbeda (misalnya: Kafe Kuini dan Kafe Monstera) yang menggunakan satu platform yang sama namun merasa seolah-olah sistem tersebut didedikasikan khusus untuk mereka.

2.9 Metodo*log*i *Prototyping*

Metodo*log*i *prototyping* adalah model pengembangan perangkat lunak yang berfokus pada pembuatan model awal sistem (*prototype*) untuk mendapatkan umpan balik pengguna secara cepat sebelum sistem final dikembangkan [2]. Metode ini sangat cocok diterapkan pada kasus pengembangan sistem di mana kebutuhan pengguna mungkin belum terdefinisi secara detail di awal atau cenderung berubah-ubah [9]. Berikut adalah alur dari metodo*log*i *prototyping*:



Gambar 2. 3 Alur Kerja *Prototyping*

Penjelasan Tahapan *Prototyping* [9]:

1. Pengumpulan kebutuhan awal *(listen to customer)*: pada tahap ini, pengembang dan pengguna bertemu untuk mendefinisikan tujuan sistem secara umum dan mengidentifikasi kebutuhan dasar yang sudah diketahui. Masukan di tahap ini mungkin belum detail, namun cukup untuk memulai pembuatan model awal.
2. Membangun prototipe: pengembang dengan cepat membangun sebuah prototipe yang fokus pada antarmuka pengguna *(Shift interface)* dan alur kerja dasar. Ini bisa berupa *mockup* desain, *Wireframe*, atau aplikasi fungsional parsial yang menunjukkan alur kerja utama tanpa perlu terhubung ke *log*ika *backend* atau basis data penuh.
3. Evaluasi prototipe / uji coba pengguna *(customer test drive):* ini adalah inti dari metode *prototyping*. Prototipe diserahkan kepada pengguna untuk dievaluasi dan diuji coba. Pengguna mencoba mengoperasikan sistem dan melihat apakah alur dan fungsionalitasnya telah sesuai dengan proses bisnis yang mereka jalankan.
4. Umpan balik dan perbaikan *(msodification)*: setelah evaluasi, pengguna memberikan umpan balik. Mereka mungkin menemukan kekurangan, kebingungan dalam alur, atau mengusulkan fitur tambahan. Pengembang mencatat masukan ini untuk melakukan perbaikan atau modifikasi pada prototipe.

5. Iterasi (perulangan) proses dari tahap 2 membangun/revisi) ke tahap 4 (perbaikan) diulang terus-menerus: setiap siklus iterasi akan menghasilkan prototipe yang semakin detail dan semakin dekat dengan kebutuhan pengguna akhir. Siklus ini berhenti ketika pengguna telah menyetujui prototipe dan merasa sistem tersebut sudah "tepat guna".
6. Implementasi sistem final (penggunaan sistem): setelah prototipe disetujui sepenuhnya, prototipe tersebut dibekukan dan dijadikan cetak biru *(blueprint).* Pengembang kemudian membangun sistem final yang lengkap, kokoh, dan aman berdasarkan desain prototipe yang telah divalidasi tersebut.

2.10 *Unified Modeling Language* (UML)

Unified modeling language (UML) didefinisikan sebagai salah satu alat bantu yang sangat handal di dunia pengembangan sistem yang berorientasi objek [35]. UML menyediakan standar bahasa pemodelan visual yang digunakan untuk menspesifikasikan, memvisualisasikan, membangun, dan mendokumentasikan artifak dari sistem perangkat lunak [9]. Selain itu, UML memiliki sejumlah elemen grafis yang dapat dikombinasikan menjadi diagram untuk menggambarkan atau mendokumentasikan berbagai aspek dari sebuah sistem [35]. Dalam pengembangan sistem ini, UML berfungsi sebagai media komunikasi utama antara pengembang dan pemangku kepentingan untuk menyamakan persepsi mengenai struktur dan perilaku sistem yang akan dibangun.

2.10.1 *Use case* Diagram

Use case Diagram merupakan pemodelan untuk kelakuan *(behavior)* sistem informasi yang akan dibuat [35]. Diagram ini mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibangun [9]. Tujuan utamanya adalah untuk mengetahui fungsi apa saja yang ada di dalam sistem dan siapa saja yang berhak menggunakan fungsi-fungsi tersebut [35]. Secara umum, komponen utama yang menyusun *Use case Diagram* terdiri dari:

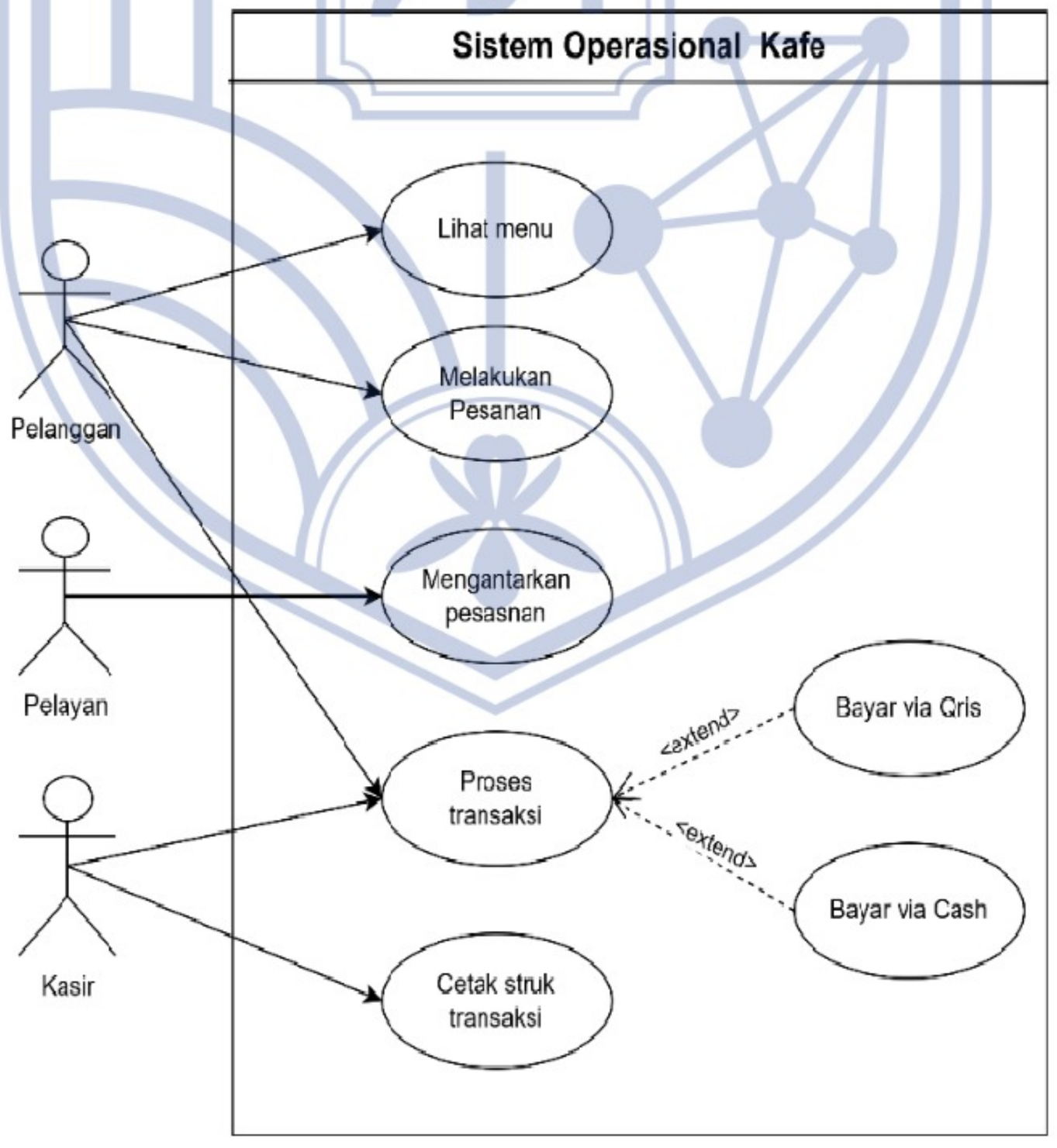
1. Aktor (*Actor*): merepresentasikan seseorang atau sistem lain yang berinteraksi dengan sistem untuk mencapai tujuan tertentu [9]. Aktor menggambarkan peran pengguna yang berinteraksi dengan sistem, bukan individu spesifik [35].
2. *Use case*: merepresentasikan fungsionalitas atau layanan spesifik yang disediakan oleh sistem bagi pengguna [8]. Setiap *use case* menjelaskan urutan aktivitas yang memberikan nilai yang dapat diamati bagi aktor [9].

3. Relasi (*Association*): garis penghubung yang menunjukkan hubungan komunikasi atau interaksi antara aktor dengan *use case* tertentu [35].

Namun, dalam pedoman perancangan standar terdapat beberapa kesalahan fatal (*common pitfalls*) yang wajib dihindari agar diagram valid secara sintaksis [36]:

1. Hubungan Antar-Aktor: Dilarang menghubungkan satu aktor dengan aktor lain secara langsung menggunakan garis asosiasi, karena interaksi aktor terjadi di luar batasan sistem.
2. Penggunaan Simbol Keputusan: Dilarang menggunakan simbol belah ketupat (*decision symbol*) dalam *Use case Diagram*. *Log*ika percabangan (seperti kondisi "Jika stok habis") harus direpresentasikan menggunakan relasi <<*extend*>>, bukan simbol grafis keputusan seperti pada Flowchart.
3. Ketiadaan *System boundary*: Diagram wajib memiliki kotak pembatas sistem untuk memisahkan secara tegas mana fungsionalitas yang berada di dalam sistem dan mana entitas eksternal.

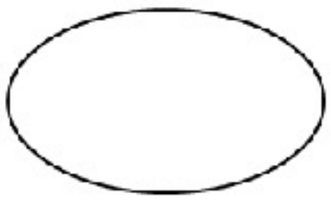
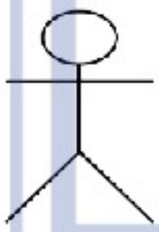
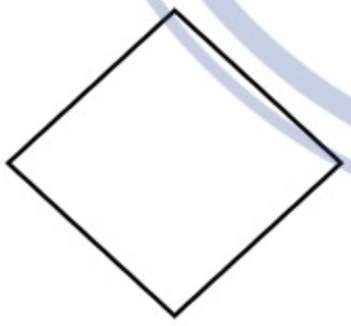
Berikut adalah gambar *Use case Diagram* untuk sistem operasional kafe secara umum:

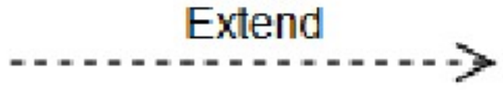
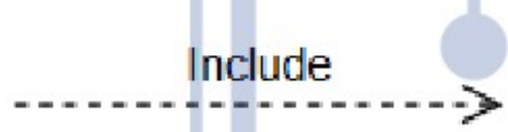


Gambar 2. 4 *Use case* Diagram Operasional Kafe secara Umum

Simbol-simbol *Use case Diagram*:

Tabel 2. 1 Penjelasan Simbol-simbol *Use case*

Gambar	Nama	Keterangan
	Oval	Digunakan untuk menggambarkan *use case*, yaitu fungsionalitas atau layanan yang disediakan oleh sistem. Setiap *use case* mewakili suatu tindakan atau proses yang dilakukan dalam sistem yang dapat diakses oleh aktor.
	Aktor	Digambarkan dengan simbol figur manusia atau entitas eksternal yang berinteraksi dengan sistem. Aktor bisa berupa pengguna sistem, seperti manusia, atau sistem lain yang terhubung dengan sistem yang dianalisis.
	Persegi panjang	Menunjukkan batas sistem, yaitu ruang lingkup yang membedakan apa yang termasuk dalam sistem dan apa yang berada di luar sistem. Semua *use case* berada di dalam batas sistem ini, sementara aktor berada di luar batas tersebut.
	*decision* (belah ketupat)	Berfungsi sebagai pengambilan keputusan atau percabangan kondisi dalam *Use case* direpresentasikan melalui Relasi *Extend* (<<*extend*>>). Artinya, jika ada kondisi tertentu atau opsional (misalnya: "Jika Saldo Kurang"), hal tersebut digambarkan sebagai *Use case* tambahan yang terhubung ke *Use case* utama dengan garis putus-putus, bukan dengan simbol belah ketupat.
	Garis lurus *(solid line)*	Untuk menunjukkan jalur komunikasi antara aktor dengan *use case* yang saling berpartisipasi, yang menandakan bahwa aktor tersebut terlibat

		langsung dalam menjalankan fungsionalitas sistem tersebut.
Extend 	garis relasi *extend*	Digunakan untuk menunjukkan hubungan yang bersifat *opsional* (pilihan). Garis ini menandakan bahwa *use case* dasar dapat diperluas oleh fungsionalitas lain, namun fungsionalitas tambahan tersebut tidak harus selalu dijalankan.
Include 	garis relasi *include*	Digunakan untuk menggambarkan hubungan ketergantungan di mana sebuah *use case* (basis) secara eksplisit membutuhkan fungsionalitas dari *use case* lain agar dapat berjalan sepenuhnya. Hubungan ini bersifat wajib, yang berarti setiap kali *use case* basis dieksekusi, maka *use case* yang di *include* juga pasti akan dijalankan.

Macam-Macam Hubungan Semantik

Untuk menggambarkan alur kerja yang lebih kompleks, aktor dan *use case* dihubungkan oleh relasi (garis) yang memiliki makna semantik [37]:

1. *Association* (Asosiasi): merupakan relasi paling dasar, digambarkan sebagai garis lurus, yang menunjukkan jalur komunikasi antara aktor dengan *use case* yang saling berpartisipasi.
2. *Include*: relasi ini menunjukkan bahwa satu *use case* (basis) akan meng-*include* fungsionalitas dari *use case* lainnya (tambahan). *Use case* tambahan akan diimplementasikan setiap kali *use case* basis diimplementasikan. *Use case* yang ditambahkan ini diperlukan untuk menjalankan fungsinya atau sebagai syarat dijalankannya *use case* basis.
3. *Extend*: relasi ini menunjukkan bahwa satu *use case* (basis) dapat diperluas oleh fungsionalitas *use case* lain (tambahan). *Use case* tambahan ini bersifat opsional dan bisa saja diimplementasikan atau tidak setiap kali *use case* basis dijalankan. Ini merupakan penambahan perilaku ke dalam *use case* dasar yang tidak tahu tentang hal tersebut.

4. *Generalization* (Generalisasi): merupakan hubungan generalisasi dan spesialisasi (umum-khusus). Relasi ini digunakan untuk membuat aktor atau *use case* yang lebih spesifik dari aktor atau *use case* yang lebih umum.

2.10.2 *Activity* Diagram

Activity Diagram merupakan representasi grafis dari alur kerja kegiatan *(workflow)* dalam sebuah sistem yang sedang dirancang. Diagram ini tidak hanya menggambarkan urutan proses bisnis, tetapi juga mampu menangani pemodelan sistem yang kompleks yang melibatkan proses paralel [9]. Secara teknis, *Activity* Diagram mendeskripsikan bagaimana aktivitas dikoordinasikan untuk menyediakan layanan, dengan komponen-komponen detail sebagai berikut [32]:

1. Partisi *(swimlanes):* garis vertikal atau horizontal yang membagi diagram menjadi beberapa area tanggung jawab, merepresentasikan aktor atau unit organisasi yang bertanggung jawab atas aktivitas tertentu (misalnya: pemisahan tugas antara pelanggan, sistem, dan kasir).
2. *Action state*: representasi dari eksekusi instruksi atomik atau langkah non-interupsi dalam sebuah proses.
3. *Transition:* menunjukkan aliran kontrol dari satu aktivitas ke aktivitas berikutnya.
4. *Decision node* (percabangan): disimbolkan dengan belah ketupat, digunakan untuk menggambarkan kondisi *log*is (*boolean*) yang menghasilkan jalur alternatif dalam alur kerja.
5. *Fork* dan *join node*: batang hitam tebal yang digunakan untuk memodelkan konkurensi (proses yang berjalan bersamaan). *Fork* memecah satu aliran menjadi beberapa aliran paralel, sedangkan Join menyinkronkan kembali aliran-aliran tersebut. Hal ini krusial dalam sistem POS untuk menggambarkan proses pesanan makanan ke dapur dan minuman ke bar yang terjadi secara simultan.

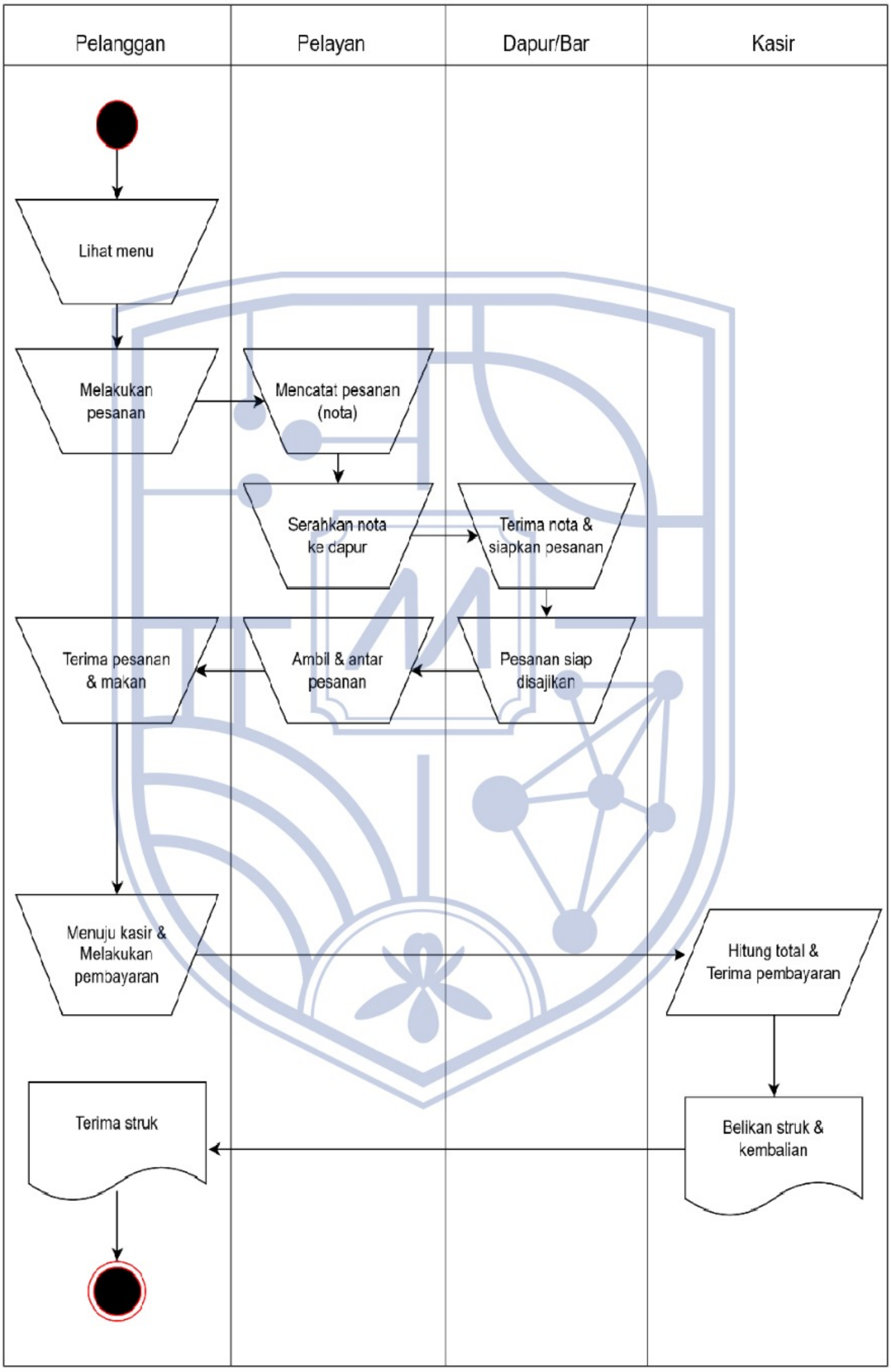
Mengacu pada pedoman standar validasi model sebuah *Activity Diagram* dikatakan valid jika memenuhi kaidah *log*ika alur (*flow logic*). Terdapat beberapa kesalahan fatal yang sering terjadi dalam perancangan dan wajib dihindari [38]:

1. Aktivitas Lubang Hitam (*Black hole*): Kesalahan *log*ika di mana sebuah aktivitas (*state*) memiliki aliran masuk (*incoming flow*) tetapi tidak memiliki aliran keluar (*outgoing flow*). Hal ini menyebabkan proses terhenti di satu titik dan tidak pernah mencapai status selesai.

2. Aktivitas Ajaib (*Miracle*): Kebalikan dari *black hole*, yaitu kesalahan di mana sebuah aktivitas tiba-tiba menghasilkan output atau aliran keluar tanpa adanya input atau pemicu sebelumnya. Secara *log*ika sistem, tidak ada proses yang bisa berjalan tanpa pemicu.
3. Ketiadaan *Final node*: Setiap diagram aktivitas wajib bermuara pada simbol *End state* (lingkaran solid bertepi). Kesalahan umum yang terjadi adalah alur yang menggantung tanpa penutup, yang menandakan proses bisnis tidak tuntas.Ambiguitas pada Simbol Keputusan (*Decision node*): Simbol belah ketupat (*decision*) wajib memiliki minimal dua aliran keluar (misalnya: "Ya" dan "Tidak"). Kesalahan terjadi jika simbol keputusan hanya memiliki satu aliran keluar, yang membuat fungsi pengecekan kondisi menjadi tidak berguna.

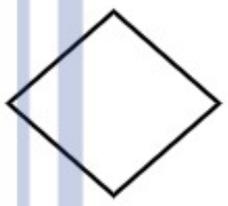
Berikut adalah gambar Alur *activity* Diagam untuk sistem operasional kafe secara umum:

Tabel 2. 2 Alur Activity Diagram Operasional Kafe



Simbol-Simbol *Activity Diagram:*

Tabel 2. 3 Penjelasan Simbol-simbol Activity Diagram

Gambar	Nama	Keterangan
	*Start point*	Menunjukkan aliran objek dari sebuah action atau *activity* ke *action*.
	*End point*	Menyatakan bahwa sebuah objek dibentuk atau diakhiri.
	*Decision*	Berfungsi sebagai titik percabangan *log*ika (branching point). Simbol ini menandakan bahwa sistem harus melakukan pengecekan kondisi (condition checking) sebelum melangkah ke proses berikutnya.
	Manual Operation *(Diamond)*	Berfungsi untuk menggambarkan aktivitas atau tindakan yang dilakukan secara fisik oleh manusia tanpa bantuan otomatisasi komputer.
	Jajar Genjang	Digunakan untuk menunjukkan proses input data secara manual, misalnya mengetikkan angka di kalkulator atau mesin kasir manual.
	Dokumen	Berfungsi untuk menunjukkan bahwa sebuah aktivitas menghasilkan atau menggunakan dokumen sebagai bagian dari alur proses.
	*Transition*	Menghubungkan aktivitas selanjutnya setelah aktivitas sebelumnya.

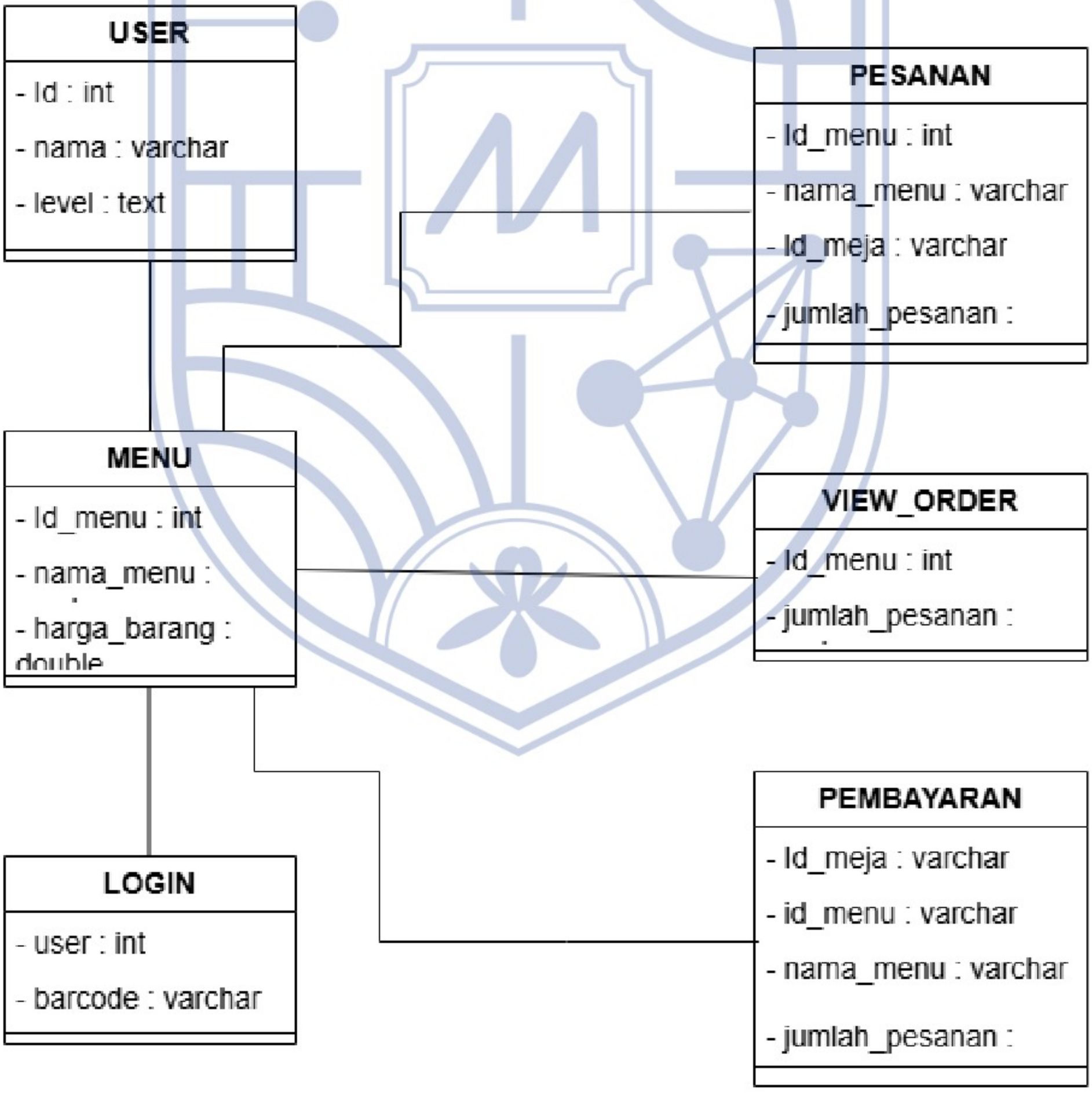
2.10.3 *Class* Diagram

Class diagram adalah diagram struktur statis yang menggambarkan struktur sistem dengan menunjukkan kelas-kelas sistem, atributnya, operasinya (metode), dan hubungan antar objek [9]. Diagram ini menjadi landasan fundamental bagi pemodelan data dan

pemrograman berorientasi objek (*Object Oriented Programming*). Struktur detail dari *class diagram* mencakup tiga kompartemen utama [32]:

1. *Class name*: nama unik yang mengidentifikasi entitas (contoh: transaksi).
2. *Attributes*: variabel yang mendeskripsikan properti atau karakteristik dari kelas tersebut. Penulisan atribut seringkali menyertakan tingkat visibilitas (*visibility*) seperti private (-) untuk keamanan data enkapsulasi, atau publik (+) untuk akses terbuka.
3. *Operations* (metode): fungsi-fungsi yang dapat dilakukan oleh objek dari kelas tersebut untuk memanipulasi data (contoh: hitungTotal(), simpanData()). Selain itu, diagram ini mendefinisikan kardinalitas (*multiplicity*) hubungan antar kelas, seperti *one-to-one* (1..1) atau *one-to-many* (1..*), yang menjadi cetak biru langsung bagi perancangan tabel dan *foreign key* dalam basis data.

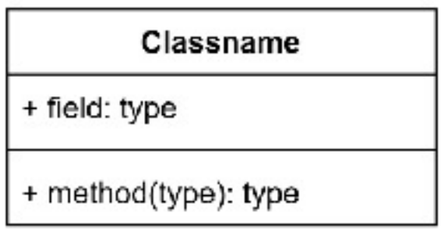
Berikut adalah gambar *class diagram* untuk sistem operasional kafe secara umum:



Gambar 2. 5 Class Diagram Operasional Kafe secara Umum

Simbol-simbol *class diagram*:

Tabel 2. 4 Penjelasan Simbol-simbol Class Diagram

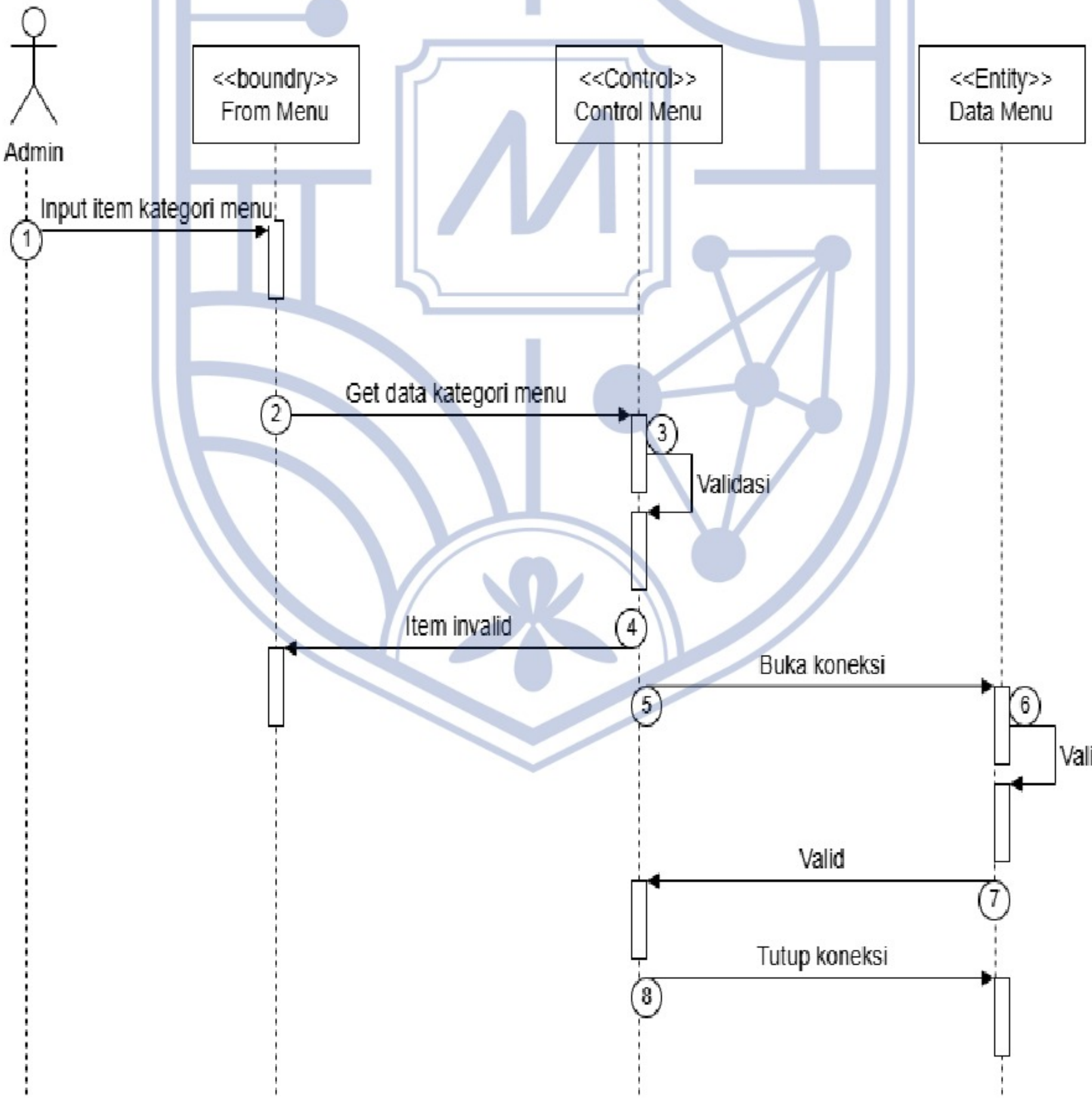
Simbol	Nama	Keterangan
	*Class*	Menggambarkan cetak biru (*blueprint*) dari sebuah objek yang terdiri dari nama kelas, atribut, dan operasi/metode.
	Association	Menunjukkan hubungan statis antar kelas yang menggambarkan bagaimana objek-objek tersebut saling berinteraksi.
	*Directed Association*	Menggambarkan hubungan antar kelas yang hanya memiliki sifat navigasi satu arah, di mana kelas asal dapat mengakses fitur atau atribut dari kelas target.
	Aggregation	Hubungan yang menunjukkan sebuah kelas adalah bagian dari kelas lain, namun kelas tersebut tetap dapat berdiri sendiri.
	*Composition*	Bentuk hubungan yang lebih kuat dari agregasi, di mana jika kelas utama dihapus, maka kelas bagiannya juga akan terhapus.
	*Multiplicity*	Digunakan untuk mendefinisikan batasan jumlah partisipasi antar objek dalam sebuah relasi. Penentuan *multiplicity* yang tepat sangat krusial karena akan menjadi acuan dalam pembentukan *foreign key* dan *log*ika validasi pada basis data, guna memastikan integritas data dalam sistem.
	*Generalization*	Menunjukkan hubungan pewarisan (*inheritance*) antara kelas induk (*superclass*) dengan kelas turunan (*subclass*).

2.10.4 *Sequence* Diagram

Sequence diagram adalah diagram interaksi yang menekankan pada pengurutan waktu pesan (*time-ordering of messages*). Diagram ini memvisualisasikan interaksi antara objek-objek dalam skenario tunggal dari sebuah *use case*, merinci bagaimana objek berkolaborasi dan dalam urutan apa pesan dikirimkan [9]. Elemen teknis dalam *sequence diagram* meliputi [7]:

1. *Lifeline*: garis putus-putus vertikal yang merepresentasikan keberadaan objek selama interaksi.
2. *Activation bar* (fokus kontrol): balok persegi panjang pada *lifeline* yang menunjukkan periode waktu di mana sebuah objek sedang aktif melakukan pemrosesan atau menunggu respons.
3. *Synchronous message*: pesan di mana pengirim menunggu respons sebelum melanjutkan proses (digambarkan dengan panah penuh).
4. *Asynchronous message*: pesan di mana pengirim tidak menunggu respons dan langsung melanjutkan proses (digambarkan dengan panah garis). Ini relevan dalam fitur notifikasi pesanan di sistem POS di mana sistem tidak perlu menunggu *waiter* membaca notifikasi untuk melanjutkan proses lain.

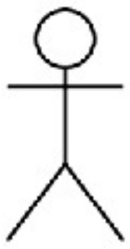
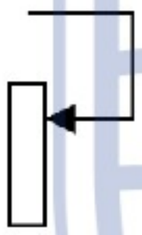
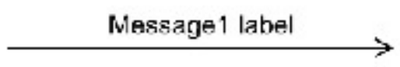
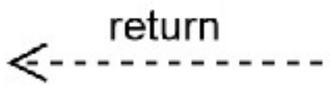
Berikut adalah gambar *sequence diagram* untuk sistem operasional kafe secara umum:



Gambar 2. 6 Sequence Diagram Operasional Kafe secara Umum

Simbol-simbol sequence diagram:

Tabel 2. 5 Penjelasan Simbol-simbol Sequence Diagram

Simbol	Nama	Keterangan
	*Actor*	Merepresentasikan entitas eksternal yang berinteraksi dengan sistem, seperti pengguna (user) yang menginisialisasi suatu proses.
	Lifeline	Garis putus-putus vertikal yang menggambarkan masa hidup dan keberadaan sebuah objek selama interaksi berlangsung.
	*Activation bar*	Balok vertikal pada *lifeline* yang menunjukkan periode waktu di mana sebuah objek sedang aktif melakukan pemrosesan atau eksekusi perintah.
	*Self Call*	Pesan yang dikirimkan oleh sebuah objek ke dirinya sendiri untuk mengeksekusi operasi internal dalam objek tersebut.
	Synchronous message	Pesan yang dikirimkan di mana pengirim menunggu respon dari penerima sebelum melanjutkan instruksi berikutnya (digambarkan dengan panah tertutup).
	*Asynchronous message*	Pesan yang dikirimkan di mana pengirim tidak perlu menunggu respon dan dapat langsung melanjutkan proses lain (digambarkan dengan panah terbuka).
	*Return message*	Pesan balasan yang dikirimkan kembali dari objek penerima ke objek pengirim setelah sebuah operasi berhasil diselesaikan (digambarkan dengan garis putus-putus).

2.11 Perangkat Lunak Pendukung

Dalam upaya membangun sistem *Point of Sale* (POS) yang stabil dan efisien, diperlukan pemilihan perangkat lunak pendukung yang tepat. Perangkat lunak ini

mencakup bahasa pemrograman, *framework*, hingga sistem manajemen basis data yang berfungsi sebagai fondasi utama dalam mengimplementasikan *log*ika sistem serta menjaga integritas data pelanggan. Berikut adalah rincian perangkat lunak yang digunakan dalam pengembangan sistem ini.

2.11.1 PHP (*Hypertext Preprocessor*)

PHP adalah bahasa pemrograman jenis *server-side scripting* yang dirancang khusus untuk pengembangan web dinamis. Artinya, seluruh *log*ika program dieksekusi di sisi server sebelum hasilnya dikirimkan ke *browser* pengguna dalam bentuk HTML statis [9]. PHP memiliki kemampuan untuk berinteraksi langsung dengan basis data, memproses data formulir, mengelola *session* pengguna, dan menangani *log*ika bisnis yang kompleks [4]. Dalam pengembangan sistem ini, PHP berfungsi sebagai otak yang menjembatani antarmuka pengguna dengan basis data. Keunggulan utama PHP adalah sifatnya yang *open-source*, kompatibilitas lintas platform (dapat berjalan di *Windows*, *Linux*, dll), serta dukungan komunitas yang luas [32]. Contoh sintaks dasar PHP untuk menampilkan teks:

contoh kode php.php

```php
<?php
    echo "Sistem POS Terintegrasi";
    // Koneksi ke database
    $db = mysqli_connect("localhost", "root", "", "db_pos");
?>
```

Gambar 2. 7 Contoh penulisan Sintaks Dasar Pemrograman

2.11.2 HTML (*Hypertext Markup Language*)

HTML adalah bahasa markah standar (*standard markup language*) yang digunakan untuk membangun kerangka atau struktur dasar dari sebuah halaman *web* [34]. HTML bukanlah bahasa pemrograman yang memiliki *log*ika (seperti perulangan atau kondisi), melainkan bahasa yang menggunakan *tag* untuk mendefinisikan elemen-elemen konten seperti teks, gambar, tautan, tabel, dan formulir input [33]. Fungsi utama HTML dalam sistem ini adalah sebagai kerangka antarmuka yang akan diisi oleh konten dinamis dari PHP dan dipercantik oleh CSS. Struktur dokumen HTML terdiri dari elemen *head* (informasi meta) dan *body* (konten yang terlihat). Contoh struktur dasar HTML:

contoh kode html.html

```html
<!DOCTYPE html>
<html>
  <head><title>Halaman Kasir</title></head>
  <body>
    <h1>Daftar Menu</h1>
    <button>Tambah Pesanan</button>
  </body>
</html>
```

Gambar 2. 8 Contoh Struktur Dasar Dokumen HTML

2.11.3 CSS (*Cascading Style Sheets*)

CSS *(Cascading Style Sheets)* adalah bahasa desain yang digunakan untuk menyederhanakan proses pembuatan halaman *web* dengan mengatur tampilan visual dari elemen HTML [34]. CSS bekerja dengan memisahkan konten (HTML) dari presentasi (desain), yang memungkinkan pengembang untuk mengontrol tata letak (*layout*), warna, jenis huruf (*font*), dan responsivitas halaman di berbagai ukuran layar (desktop, tablet, ponsel) secara terpusat [33]. Dalam sistem POS ini, CSS berperan penting untuk menciptakan antarmuka yang ramah pengguna (*user-friendly*), seperti membedakan warna tombol Bayar dan Batal, atau menata daftar menu agar rapi. Contoh penggunaan CSS:

```
# contoh kode css.css > ...
1   /* Mengubah warna tombol pesanan menjadi hijau */
2   .btn-order {
3     background-color: #28a745;
4     color: white;
5     padding: 10px 20px;
6     border-radius: 5px;
7   }
```

Gambar 2. 9 Contoh Aturan Penulisan CSS

2.11.4 JavaScript

JavaScript adalah bahasa pemrograman tingkat tinggi yang berjalan di sisi klien (*client-side*), yang berarti kode dieksekusi langsung oleh *browser* pengguna, bukan oleh *server* [29]. *JavaScript* berfungsi untuk membuat halaman *web* menjadi interaktif dan dinamis tanpa perlu memuat ulang (*reload*) halaman secara keseluruhan. Kelebihan utama *JavaScript* adalah kemampuannya untuk memanipulasi elemen HTML dan CSS secara *real-time*. Dalam konteks sistem pemesanan, *JavaScript* digunakan untuk fungsi-fungsi seperti menghitung total harga secara otomatis saat jumlah pesanan diubah, memvalidasi input formulir sebelum dikirim ke server, dan menampilkan notifikasi *pop-up* sukses/gagal [29]. Contoh fungsi *JavaScript* sederhana:

```
JS contoh kode js.js > ...
1   function hitungSubtotal(harga, qty) {
2     let total = harga * qty;
3     alert("Total Harga: Rp " + total);
4     return total;
5   }
```

Gambar 2. 10 Contoh *Log*ika Pemrograman JavaScript

2.11.5 Laragon

Laragon adalah sebuah *universal development environment* yang bersifat portabel, terisolasi, dan sangat ringan, yang dirancang khusus untuk meningkatkan efisiensi dalam

pengembangan aplikasi web berbasis PHP, Node.js, hingga Python [39]. Sebagai infrastruktur server lokal modern, Laragon menawarkan keunggulan melalui fitur *virtual host* otomatis yang memungkinkan pengembang mengakses proyek melalui URL kustom guna mensimulasikan lingkungan produksi yang lebih nyata. Selain itu, Laragon memiliki integrasi bawaan dengan fitur Ngrok yang berfungsi untuk melakukan sharing proyek lokal ke jaringan internet secara instan melalui *secure tunnel*. Dalam pengembangan sistem *multi-tenant* ini, Laragon berperan penting dalam mengelola layanan web server dan sistem manajemen basis data secara stabil, serta memfasilitasi pengujian akses layanan oleh klien dari luar jaringan lokal guna memastikan validitas arsitektur sistem yang dibangun [40].

2.11.6 Ngrok (*Localhost tunneling*)

Localhost tunneling merupakan teknologi jaringan yang memungkinkan server lokal (*localhost*) pada komputer pengembang untuk dapat diakses dari jaringan publik melalui internet tanpa memerlukan konfigurasi infrastruktur server berbayar atau alamat IP publik yang tetap (*static public IP*). Secara teknis, mekanisme ini bekerja dengan membangun sebuah terowongan aman (*secure tunnel*) antara server lokal dan sebuah layanan perantara (*relay service*) di internet, sehingga permintaan (*request*) dari perangkat eksternal dapat diteruskan ke *server* lokal melalui terowongan tersebut [41]. Salah satu implementasi *localhost tunneling* yang paling banyak digunakan dalam pengembangan aplikasi web adalah Ngrok. Ngrok sebagai sebuah platform yang menyediakan solusi *reverse proxy* untuk mengekspos aplikasi pada jaringan pribadi agar dapat diakses melalui internet, di mana layanan ini tidak memerlukan keberadaan IP publik pada sisi pengguna.

Dalam konteks pengembangan sistem *multi-tenant* berbasis web, penggunaan Ngrok memberikan nilai strategis yang signifikan pada tahap pengujian (*testing phase*). Ngrok memungkinkan pengembang untuk memvalidasi kemampuan akses sistem dari berbagai perangkat dan lokasi yang berbeda secara simultan sebagaimana yang dibutuhkan dalam skenario nyata operasional kafe *multi-tenant* tanpa harus terlebih dahulu melakukan *deployment* ke infrastruktur server produksi yang memerlukan biaya operasional tambahan [42]. *Ngrok* dimanfaatkan sebagai alat pengujian jarak jauh yang terintegrasi secara bawaan dalam lingkungan pengembangan laragon, sehingga validitas arsitektur *client-server* dan isolasi data antar-*tenant* dapat diuji dari perangkat eksternal di luar jaringan lokal (LAN) secara efisien dan aman.