LAPORAN PROYEK MATA KULIAH PEMROGRAMAN DASAR

SEMESTER GASAL 2025/2026

SKENA: Sistem Kelola & Efisiensi Niaga Area-Kopi

Anggota Tim

1. Andhika Satria Wibisono 22/497821/TK/54578
2. Muhammad Fachrizal Giffari 22/504570/TK/55192

Departemen Teknik Elektro dan Teknologi Informasi
Fakultas Teknik
Universitas Gadjah Mada

2025

1. DESKRIPSI APLIKASI

SKENA atau Sistem Kelola & Efisiensi Niaga Area-kopi adalah aplikasi *Point of Sale* (POS) yang dibangun atas keresahan kami sebagai pecinta kopi. Kami mengobservasi bahwa sebagian toko kopi tidak menerapkan sistem *membership* untuk *customer*nya. Hal ini mempengaruhi kecenderungan *customer* untuk mengunjungi toko kopi tersebut. Selain itu, *customer* setia juga merasa tidak diapresiasi akan kesetiaannya. Space Coffee Roastery merupakan salah satu toko kopi yang berada di Yogyakarta yang menerapkan sistem *membership*. Toko kopi tersebut menawarkan harga yang lebih murah untuk para *member*nya. Selain itu, *member* juga mendapatkan notifikasi mengenai transaksi yang sudah dilakukan. Akibatnya, Space Coffee Roastery ramai dikunjungi dan merupakan salah satu toko kopi yang sering direkomendasikan oleh warga Yogyakarta. Oleh karena itu, kami mengembangkan SKENA dengan harapan banyak toko kopi dapat meraih kembali *customer*nya.

Aplikasi ini dikembangkan dengan analisis dan pemodelan menggunakan *Unified Modelling Language* (UML) untuk menganalisis dan memodelkan proses bisnis. Aplikasi ini juga dikembangkan menggunakan paradigma pemrograman berorientasi objek dengan bahasa pemrograman C++. Limitasi dari aplikasi ini yaitu belum bisa terhubung dengan basis data relasional maupun non-relasional seperti MySQL, PostgreSQL, MongoDB, dan lainnya. Aplikasi juga belum mendukung pembayaran non-tunai seperti QRIS dan transfer. Selain itu, aplikasi ini juga tidak disertai dengan *port forwarding*. Akibatnya, proses bisnis yang ada tidak bisa diakses selain di *user interface*.

2. ANALISIS KEBUTUHAN

Dalam mengembangkan aplikasi, tim pengembang tentunya membutuhkan analisis yang mendalam untuk kebutuhan bisnis, kebutuhan teknis, hingga kebutuhan manajemen. Aplikasi yang baik harus bisa diandalkan untuk melakukan proses bisnis dengan cepat tanpa adanya galat. Selain itu, aplikasi juga harus memiliki *downtime* yang sangat kecil agar pengguna mendapatkan *user experience* yang baik. Hal ini tentu dapat dicapai jika kebutuhan teknis dan kebutuhan manajemen sudah terpenuhi.

2.1. Kebutuhan Fungsional

2.1.1. Kebutuhan Fungsional *Customer*

SKENA mempunyai tiga sisi proses bisnis yaitu dari sisi *customer*, sisi produk, dan sisi *point of sale.* Untuk sisi *customer*, *customer* dapat melakukan registrasi jika ingin menjadi *member* dari toko dan mengumpulkan *loyalty point.* Jika *customer* sudah memiliki *loyalty point*, maka *customer* dapat menukarkan *loyalty point* menjadi diskon saat transaksi. Semua proses bisnis tersebut tentunya harus dapat disimpan dalam sebuah *database*. Untuk sekarang, *database* yang digunakan adalah *plain text*. Detail dari proses bisnis *customer* dapat dilihat di Tabel 1.

Tabel 1 Kebutuhan Fungsional Customer

Req-ID	Kebutuhan Fungsional	Deskripsi
C-01	Register Customer	Fitur untuk menambahkan data *customer* baru.
C-02	Edit Customer Data	Fitur untuk merubah data *customer* yang sudah ada.
C-03	Delete Customer Data	Fitur untuk menghapus *customer* dari data.
C-04	View All Customer	Fitur untuk melihat semua customer.
C-05	Insert Loyalty Point	Fitur untuk menambahkan *loyalty point* di *customer*.
C-06	Search Customer	Fitur untuk mencari *customer* dengan nama.

2.1.2. Kebutuhan Fungsional Produk

Untuk sisi produk, barista atau pelayan atau kasir dapat menambahkan produk dan mengubah detailnya. Selain itu, kasir juga dapat menghapus produk. Kasir dapat melihat semua produk yang ada dan juga dapat melihat produk berdasarkan tipe tertentu. Detail dari proses bisnis produk dapat dilihat di Tabel 2.

Tabel 2 Kebutuhan Fungsional Produk

Req-ID	Kebutuhan Fungsional	Deskripsi
P-01	Register Product	Fitur untuk menambahkan produk
P-02	Edit Product	Fitur untuk merubah data produk
P-03	Delete Product	Fitur untuk menghapus produk dari data
P-04	View All Product	Fitur untuk melihat semua produk
P-05	Filter Product	Fitur untuk menyaring produk

2.1.2. Kebutuhan Fungsional *Point of Sale*

Untuk sisi *point of sale*, barista atau pelayan atau kasir dapat menambahkan produk ke keranjang, mengubah kuantitas, hingga menghapus produk dari keranjang. Selain itu kasir juga dapat menyelesaikan transaksi maupun membatalkan transaksi. Barista juga dapat melihat riwayat transaksi yang sudah dilakukan. Detail dari proses bisnis *point of sale* dapat dilihat di Tabel 3.

Tabel 3 Kebutuhan Fungsional Point of Sale

Req-ID	Kebutuhan Fungsional	Deskripsi
POS-01	Add to Cart	Fitur untuk menambahkan produk ke keranjang
POS-02	Remove from Cart	Fitur untuk menghapus produk dari keranjang
POS-03	Update Quantity	Fitur untuk mengubah kuantitas produk dari keranjang

POS-04	Complete Transaction	Fitur untuk menyelesaikan transaksi dari keranjang
POS-05	Cancel Transaction	Fitur untuk membatalkan transaksi dari keranjang
POS-06	View History	Fitur untuk melihat riwayat transaksi

2.1.3. Kebutuhan Fungsional Data

Semua kebutuhan fungsional diatas tentunya harus ditunjang oleh penyimpanan data. Jika tidak, maka data tersebut hanya akan disimpan di *memory* yang bersifat sementara. Akibatnya, data hanya ada ketika aplikasi tersebut berjalan. Ketika aplikasi tersebut ditutup, maka data yang sudah ada akan hilang semua. Oleh karena itu, perlu adanya fitur untuk menyimpan data, mengubah data, hingga memuat data. Detail dari kebutuhan fungsional data dapat dilihat di Tabel 4.

Tabel 4 Kebutuhan Fungsional Data

Req-ID	Kebutuhan Fungsional	Deskripsi
D-01	Save Data	Fitur untuk menyimpan data baru maupun perubahan data
D-02	Load Data	Fitur untuk memuat data yang sudah ada

2.2. Kebutuhan Non Fungsional

Dalam mengembangkan SKENA pemilihan *tool*, pemilihan bahasa pemrograman, pembagian kerja, alokasi waktu, hingga *software development life cycle* (SDLC) juga harus dipertimbangkan. Pemilihan harus dipertimbangkan dengan matang agar pengembang nyaman dalam proses pengembangannya. Hal ini juga tentu meminimalisir perubahan mendadak di tengah pengembangan yang mengganggu pengembang. Detail dari kebutuhan non fungsional dapat dilihat di Tabel 5, sedangkan detail dari alokasi sumber daya manusia dapat dilihat di Tabel 6 dan detail dari alokasi waktu dapat dilihat di Tabel 7.

Tabel 5 Kebutuhan Non Fungsional SKENA

Jenis	Pemilihan	Deskripsi
IDE	CLion	CLion menawarkan integrasi CMake yang mulus dan fitur analisis kode cerdas yang sangat memudahkan konfigurasi serta debugging dalam proyek Qt 6. Selain itu, alat produktivitasnya seperti refactoring otomatis dan navigasi kode yang canggih mempercepat proses pengembangan aplikasi secara signifikan.
UML	PlantUML	PlantUML menggunakan pendekatan berbasis teks yang memungkinkan pengembang membuat diagram dengan cepat, ringan, serta mudah dilacak perubahannya melalui *version control system* seperti Git.
Desain Antarmuka	Figma	Figma sangat ideal karena dukungan Qt Bridge memungkinkan desain diekspor langsung menjadi kode QML, yang secara signifikan mempercepat proses transisi dari prototipe visual ke implementasi di Qt Design Studio. Selain itu, fitur kolaborasi real-time-nya memastikan desainer dan pengembang memiliki referensi visual yang konsisten sebelum penulisan logika aplikasi dimulai.
Version Control System	Git	Git memungkinkan pelacakan riwayat perubahan kode secara lengkap dan aman, sehingga memudahkan kolaborasi tim melalui sistem yang terdistribusi. Selain itu, fitur branching dan merging yang canggih memungkinkan pengembang mengerjakan fitur baru secara paralel tanpa mengganggu stabilitas kode utama.
Repository	GitHub	GitHub menyediakan platform kolaborasi terpusat yang memudahkan tim untuk meninjau kode dan mengelola proyek melalui fitur canggih seperti Pull Request.
SDLC	Waterfall	Aplikasi ini memiliki tingkat kerumitan yang sangat rendah, sehingga Waterfall merupakan SDLC yang sangat cocok. Selain sebagai penyimpanan awan yang aman, fitur otomatisasi seperti GitHub Actions

		memungkinkan pengujian dan penyebaran aplikasi dilakukan secara otomatis langsung dari repositori.
Framework	Qt Framework	Qt Framework memungkinkan pengembangan lintas platform (*cross-platform*), yang berarti kode yang sama dapat dikompilasi dan dijalankan di Windows, macOS, Linux, dan bahkan perangkat mobile tanpa perubahan besar. Selain itu, Qt menyediakan perpustakaan GUI yang sangat kaya, performa tinggi dengan C++, dan alat desain visual yang kuat untuk menciptakan antarmuka pengguna yang modern dan responsif.
Bahasa Pemrograman	C++	C++ memberikan performa eksekusi yang sangat tinggi dan manajemen memori langsung, yang krusial untuk memastikan antarmuka grafis berjalan mulus dan responsif tanpa lag. Selain itu, paradigma OOP memungkinkan pengorganisasian komponen GUI secara hierarkis dan modular—seperti membuat kelas 'Tombol' yang mewarisi sifat kelas 'Widget'—sehingga kode lebih mudah digunakan kembali dan dipelihara.
Compiler	Clang	Clang menghasilkan pesan kesalahan yang jauh lebih jelas dan ekspresif dibandingkan *compiler* lain seperti GCC, yang sangat membantu saat menelusuri masalah dalam kode yang penuh dengan macro dan template khas Qt. Selain itu, arsitekturnya yang modular mendukung integrasi alat analisis statis canggih seperti Clazy, yang secara khusus dirancang untuk mendeteksi inefisiensi dan kesalahan praktik terbaik dalam pengembangan aplikasi Qt.

Tabel 6 Alokasi Sumber Daya Manusia

Tahap Kegiatan	Sumber Daya Manusia	
	Andhika	Fachrizal
Identifikasi Masalah		
Peninjauan Tempat		
Pemodelan Masalah		
Perancangan Umum Sistem		
Perancangan Detail Sistem		
Pengujian Sistem		
Pembuatan Laporan		

Tabel 7 Alokasi Waktu

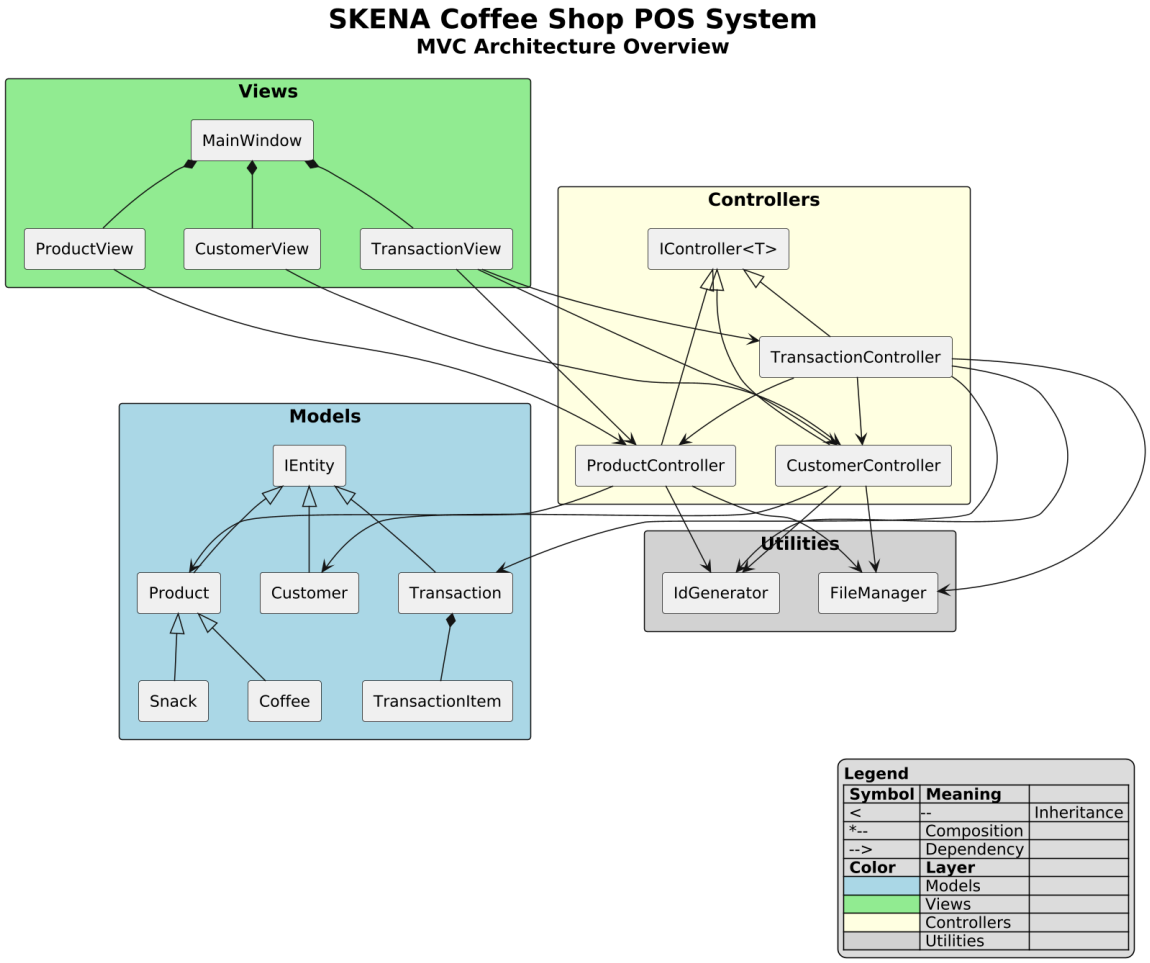
Tahap Kegiatan	Minggu ke -		
	1	2	3
Identifikasi Masalah			
Peninjauan Tempat			
Pemodelan Masalah			
Perancangan Umum Sistem			
Perancangan Detail Sistem			
Pengujian Sistem			
Pembuatan Laporan			

3. PEMODELAN DAN PERANCANGAN SISTEM

3.1. Arsitektur Sistem

SKENA menerapkan arsitektur *Model - View - Controller* (MVC) dalam pengembangannya. Arsitektur MVC memisahkan proses bisnis (*Controller*) dan data (*Model)* dari tampilan antarmuka (*View*). Hal ini memastikan kode tetap terstruktur rapi dan mudah dimodifikasi meskipun desain visualnya sering berubah. Selain itu, pola ini memudahkan pengelolaan data di *file* seperti daftar item belanjaan secara terpusat, sehingga pembaruan tampilan terjadi otomatis dan konsisten tanpa perlu kode yang rumit dan tumpang tindih. Berikut arsitektur dari SKENA yang dijelaskan di Gambar 1.

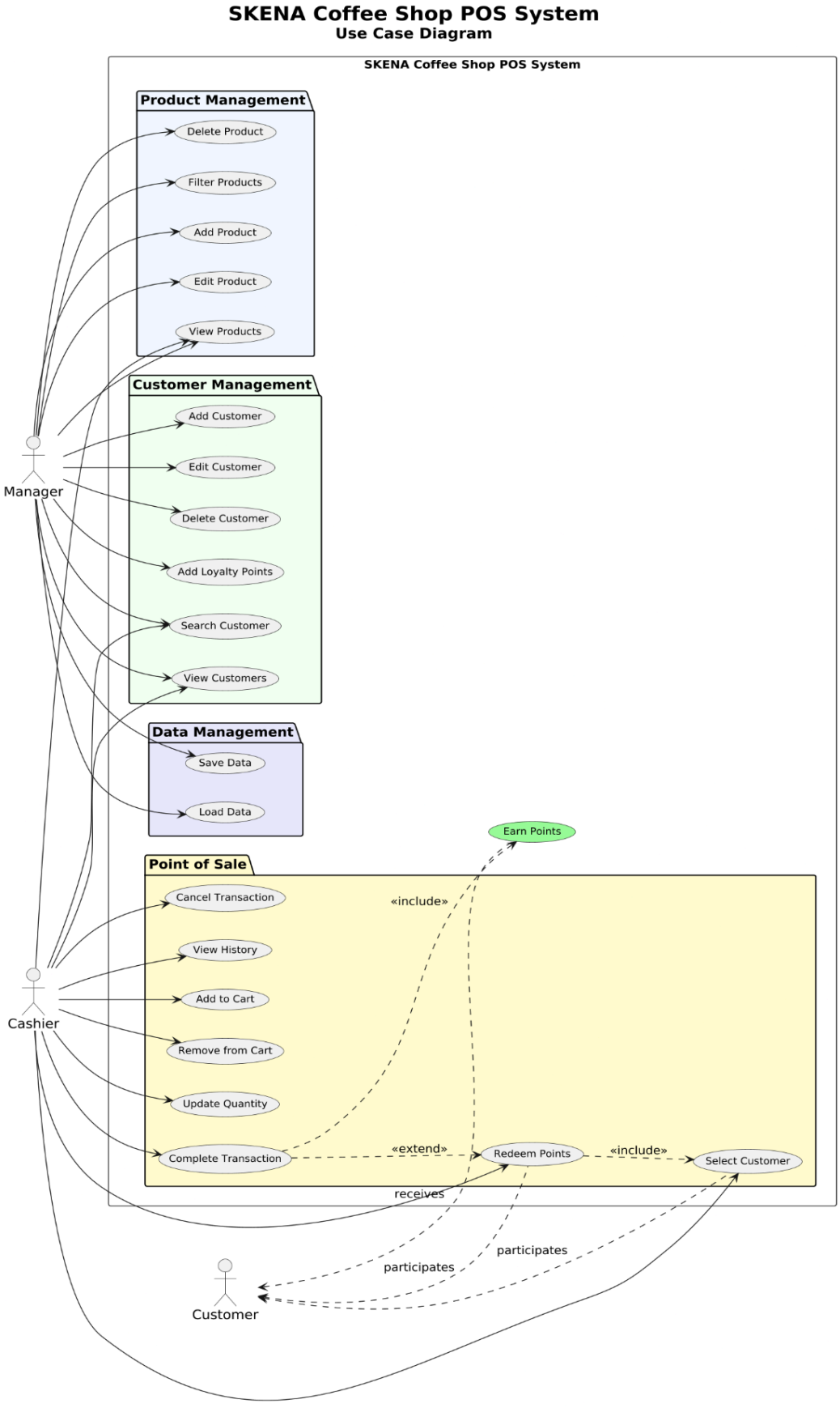
Gambar 1 Arsitektur SKENA



3.2. Use Case Diagram

SKENA memiliki 3 aktor dalam pengaplikasiannya yaitu *Manager*, *Customer*, dan *Cashier*. *Manager* bertanggung jawab atas pengelolaan *coffee shop* seperti manajemen produk, manajemen customer, dan manajemen data. *Cashier* bertanggung jawab atas pengelolaan transaksi dengan *customer* atau *point of sale*. *Customer* adalah aktor yang melakukan transaksi dengan *Cashier*. Berikut *use case diagram* dari SKENA yang dijelaskan di Gambar 2 dan Tabel 8.

Gambar 2 Use Case Diagram



Tabel 8 Skenario Use Case Diagram

Req-ID	Kebutuhan Fungsional	Skenario use case
C-01	Register Customer	1. *Manager* memilih menu *Customers* di Navbar 2. Aplikasi meminta data *customer* 3. *Manager* memasukkan data *customer* 4. Sistem menyimpan data
C-02	Edit Customer Data	1. *Manager* memilih menu *Customers* di Navbar 2. *Manager* memilih salah satu c*ustomer* 3. *Manager* mengganti data *customer* 4. Sistem menyimpan perubahan
C-03	Delete Customer Data	1. *Manager* memilih menu *Customers* di Navbar 2. *Manager* memilih salah satu *customer* 3. *Manager* menekan tombol *delete customer* 4. Sistem menyimpan perubahan
C-04	View All Customer	1. *Manager* memilih menu *Customers* di Navbar 2. Aplikasi memberikan semua data *customer*
C-05	Insert Loyalty Point	1. *Manager* memilih menu *Customers* di Navbar 2. *Manager* memilih salah satu *customer* 3. *Manager* memasukkan *loyalty point* 4. Sistem menyimpan perubahan
C-06	Search Customer	1. *Manager* memilih menu *Customers* di Navbar 2. *Manager* mencari nama *customer* 3. Aplikasi memberikan data *customer*
P-01	Register Product	1. *Manager* memilih menu *Products* di Navbar 2. Aplikasi meminta data produk 3. *Manager* memasukkan data produk 4. Sistem menyimpan data

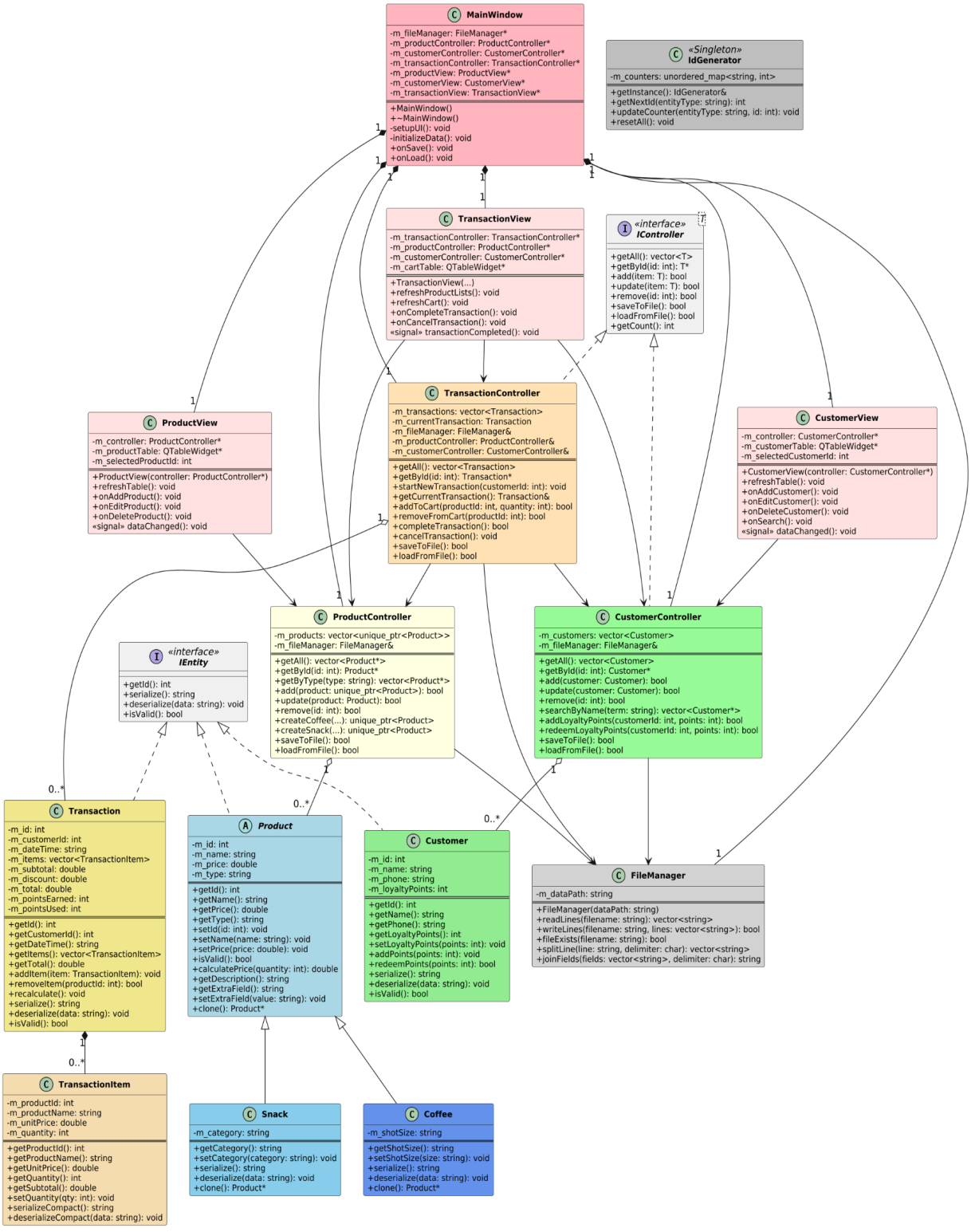
P-02	Edit Product	1. *Manager* memilih menu *Products* di Navbar 2. *Manager* memilih salah satu produk 3. *Manager* mengganti data produk 4. Sistem menyimpan perubahan
P-03	Delete Product	1. *Manager* memilih menu *Products* di Navbar 2. *Manager* memilih salah satu produk 3. *Manager* menghapus data produk 4. Sistem menyimpan perubahan
P-04	View All Product	1. *Manager* memilih menu *Products* di Navbar 2. Aplikasi memberikan data semua produk
P-05	Filter Product	1. *Manager* memilih menu *Products* di Navbar 2. *Manager* memilih tipe produk 3. Aplikasi memberikan semua data produk tipe tertentu
POS-01	Add to Cart	1. *Cashier* memilih menu *Point of Sale* di Navbar 2. *Cashier* memilih menu *Point of Sale* di Navbar sekunder 3. *Cashier* memilih data *customer* jika ada 4. *Cashier* memilih satu produk 5. Sistem menambahkan produk ke keranjang
POS-02	Remove from Cart	1. *Cashier* memilih menu *Point of Sale* di Navbar 2. *Cashier* memilih menu *Point of Sale* di Navbar sekunder 3. *Cashier* memilih satu produk di keranjang 4. *Cashier* menekan tombol *remove* 5. Sistem menghapus produk dari keranjang
POS-03	Update Quantity	1. *Cashier* memilih menu *Point of Sale* di Navbar 2. *Cashier* memilih menu *Point of Sale* di Navbar sekunder

		3. *Cashier* memilih satu produk di keranjang *4.* *Cashier* merubah kuantitas 5. Sistem merubah kuantitas produk di keranjang
POS-04	Complete Transaction	*1.* *Cashier* memilih menu *Point of Sale* di Navbar 2. *Cashier* memilih menu *Point of Sale* di Navbar sekunder *3.* *Cashier* memasukkan *loyalty point* jika ada 4. *Cashier* menekan tombol *complete*
POS-05	Cancel Transaction	*1.* *Cashier* memilih menu *Point of Sale* di Navbar 2. *Cashier* memilih menu *Point of Sale* di Navbar sekunder 3. *Cashier* menekan tombol *cancel*
POS-06	View History	*1.* *Cashier* memilih menu *Point of Sale* di Navbar 2. *Cashier* memilih menu *Transaction History* di Navbar sekunder 3. Aplikasi menampilkan riwayat transaksi
D-01	Save Data	*1.* *Cashier* atau *Manager* menjalankan aplikasi 2. *Cashier* atau *Manager* menjalankan proses bisnis 3. Sistem automatis menyimpan data
D-02	Load Data	*1.* *Cashier* atau *Manager* menjalankan aplikasi 2. Sistem automatis memuat data

3.3. Class Diagram

Diagram kelas menggambarkan bagaimana arsitektur dari sistem Point of Sale (POS) SKENA yang menerapkan pola desain MVC (*Model-View-Controller*). Dimana *MainWindow* berfungsi sebagai *main code* yang menghubungkan tiga *model* utama yaitu *Product*, *Customer*, dan *Transaction.* Setiap *model* tersebut kemudian dipasangkan dengan *controller* (*ProductController*, *CustomerController*, *TransactionController*) untuk logika bisnis dan *view* (*ProductView*, *CustomerView*, *TransactionView*) untuk tampilan antarmuka. SKENA menerapkan *interface IController* dan *IEntity* untuk menstandarisasi operasi logika serta struktur data, terlihat jelas pada hierarki *class Product* yang bersifat *abstract* dengan *inheritance* ke *Snack class* dan *Coffee class*, serta penggunaan *class* utilitas *FileManager* untuk persistensi data berbasis file dan *IdGenerator* (Singleton) untuk manajemen ID unik di seluruh aplikasi. Detail dari *class diagram* dapat dilihat di Gambar 3.

Gambar 3 Class Diagram



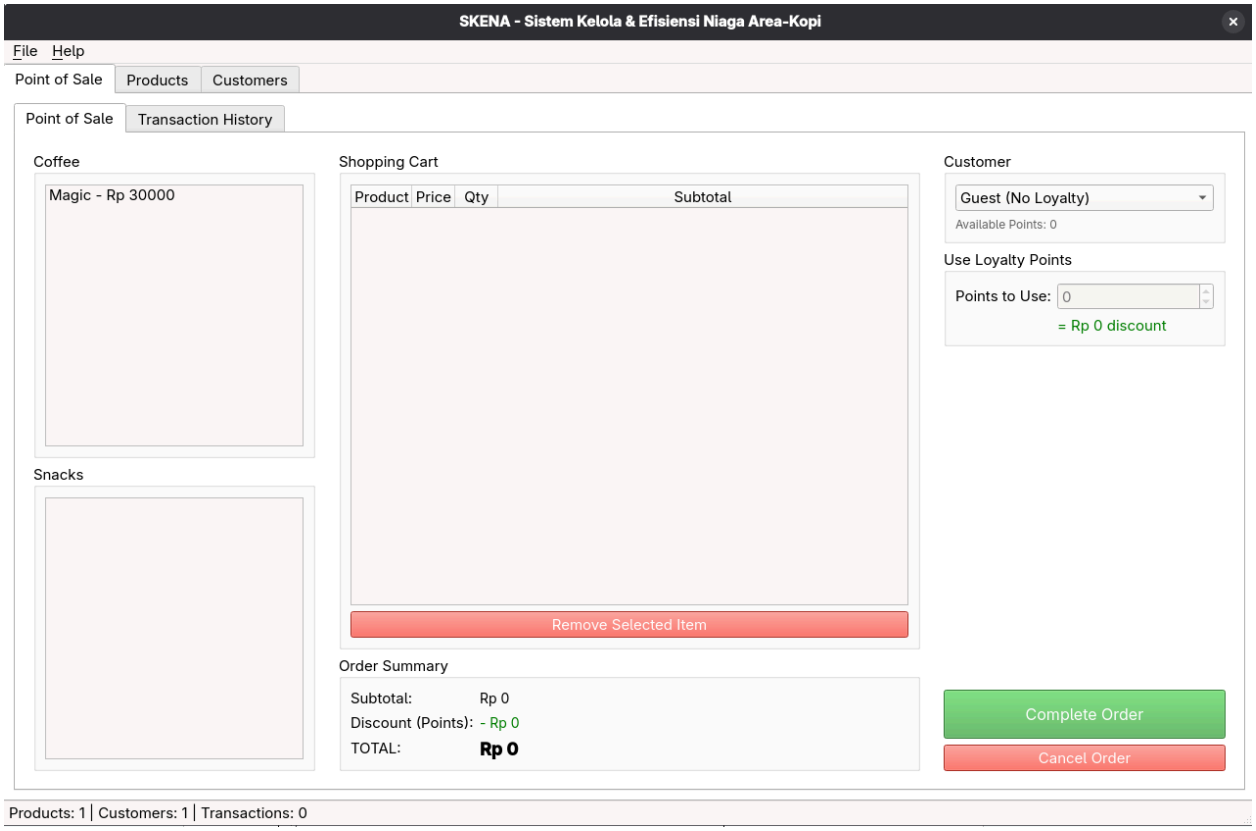
3.4. Perancangan User Interface

Desain tampilan antarmuka yang dipilih yaitu *graphical user interface* (GUI). GUI menawarkan pengalaman pengguna yang lebih baik ketimbang *console*. Selain itu, aplikasi ini memiliki target pengguna kasir, dimana mereka sangat sedikit memiliki pengalaman dengan *console.* Oleh karena itu, pengalaman pengguna merupakan aspek prioritas.

3.4.1 Perancangan User Interface Transaksi

Desain tampilan antarmuka untuk transaksi harus bisa melakukan proses bisnis POS-01, POS-02, POS-03, POS-04, dan POS-05 secara mudah. Detail dari tampilan antarmuka untuk transaksi dapat dilihat di Gambar 4.

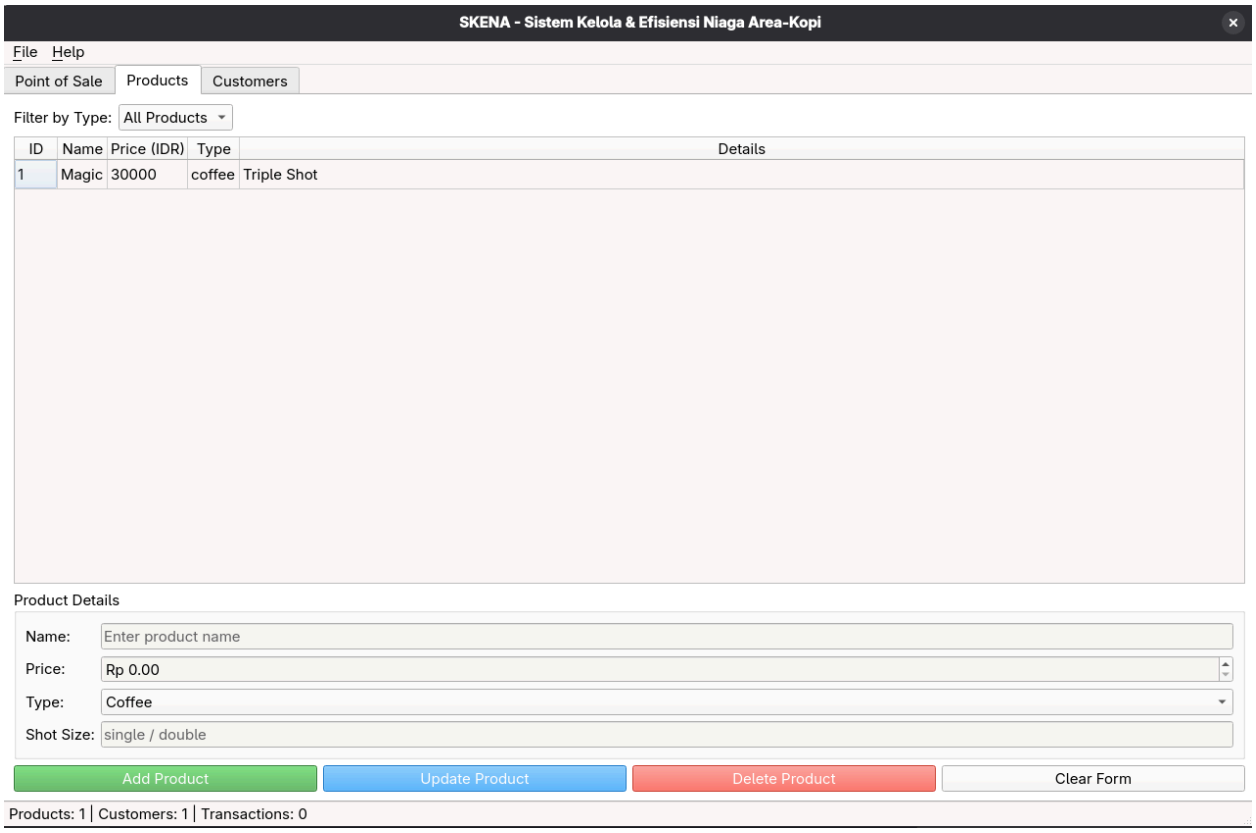
Gambar 4 Antarmuka Transaksi



3.4.2 Perancangan User Interface Produk

Desain tampilan antarmuka untuk produk harus bisa melakukan proses bisnis P-01, P-02, P-03, P-04, dan P-05 secara mudah. Detail dari tampilan antarmuka untuk produk dapat dilihat di Gambar 5.

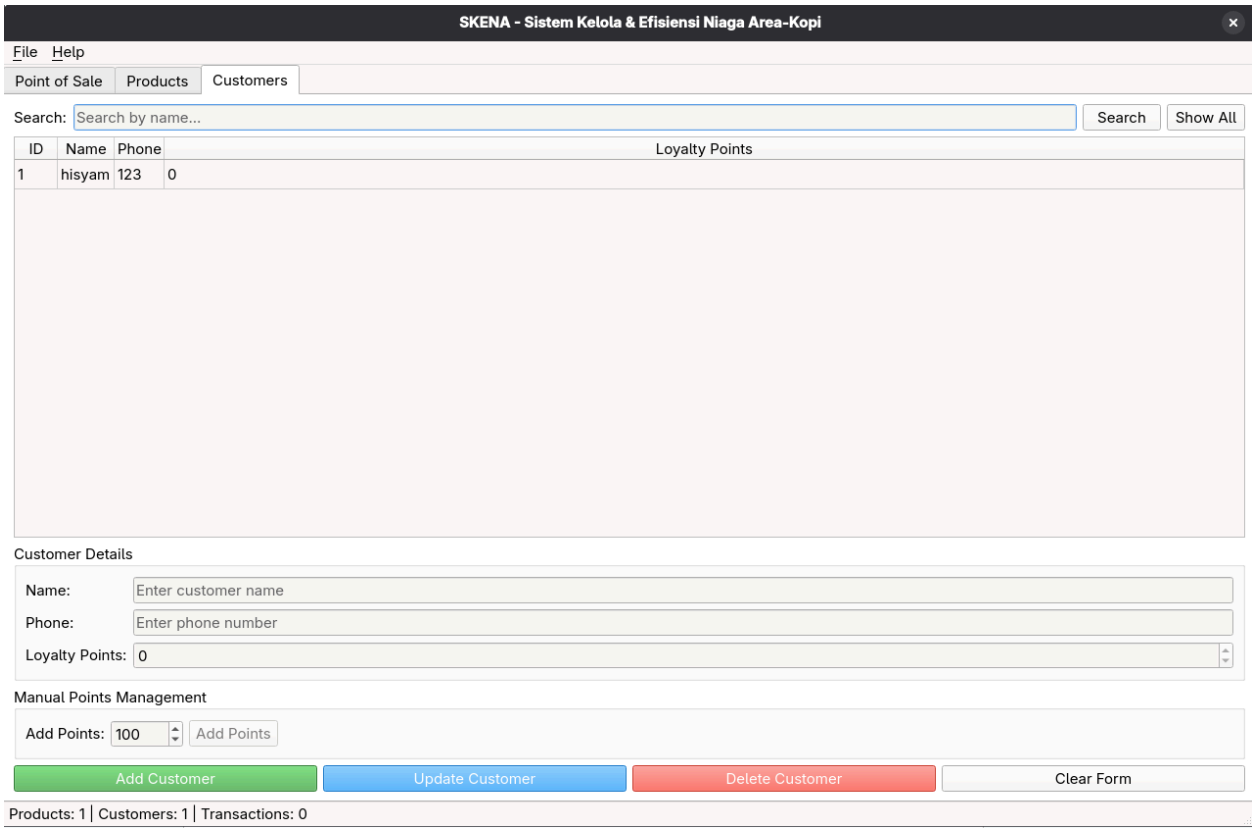
Gambar 5 Antarmuka Produk



3.4.3 Perancangan User Interface *Customer*

Desain tampilan antarmuka untuk produk harus bisa melakukan proses bisnis C-01, C-02, C-03, C-04, C-05, dan C-06 secara mudah. Detail dari tampilan antarmuka untuk *customer* dapat dilihat di Gambar 6.

Gambar 6 Antarmuka Customer



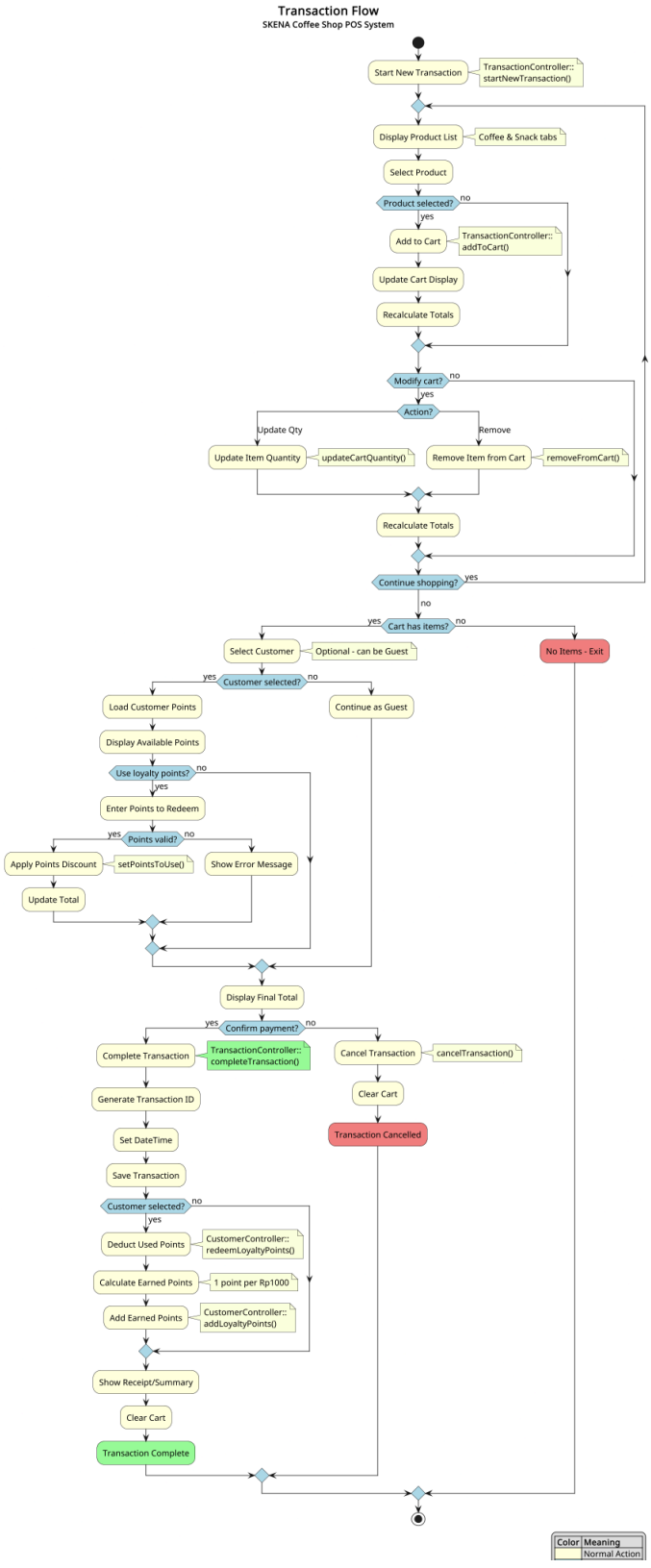
3.5. User Flow

Untuk memastikan pengalaman pengguna yang baik, maka dibutuhkan *user flow* yang baik juga. Dengan memastikan *user flow* yang baik, maka pengguna tidak memerlukan usaha yang berlebih untuk menjalankan proses bisnis atau berpindah dari proses bisnis lain. Dengan begitu, diharapkan pengguna dapat melakukan proses bisnis dengan cepat dan mudah sehingga *customer* tidak memerlukan waktu yang lama untuk mendapatkan yang diinginkan.

3.5.1 User Flow Transaksi

User Flow transaksi harus bisa mengintegrasikan proses bisnis POS-01, POS-02, POS-03, POS-04, dan POS-05 secara mudah. Detail dari *user flow* untuk *transaksi* dapat dilihat di Gambar 7.

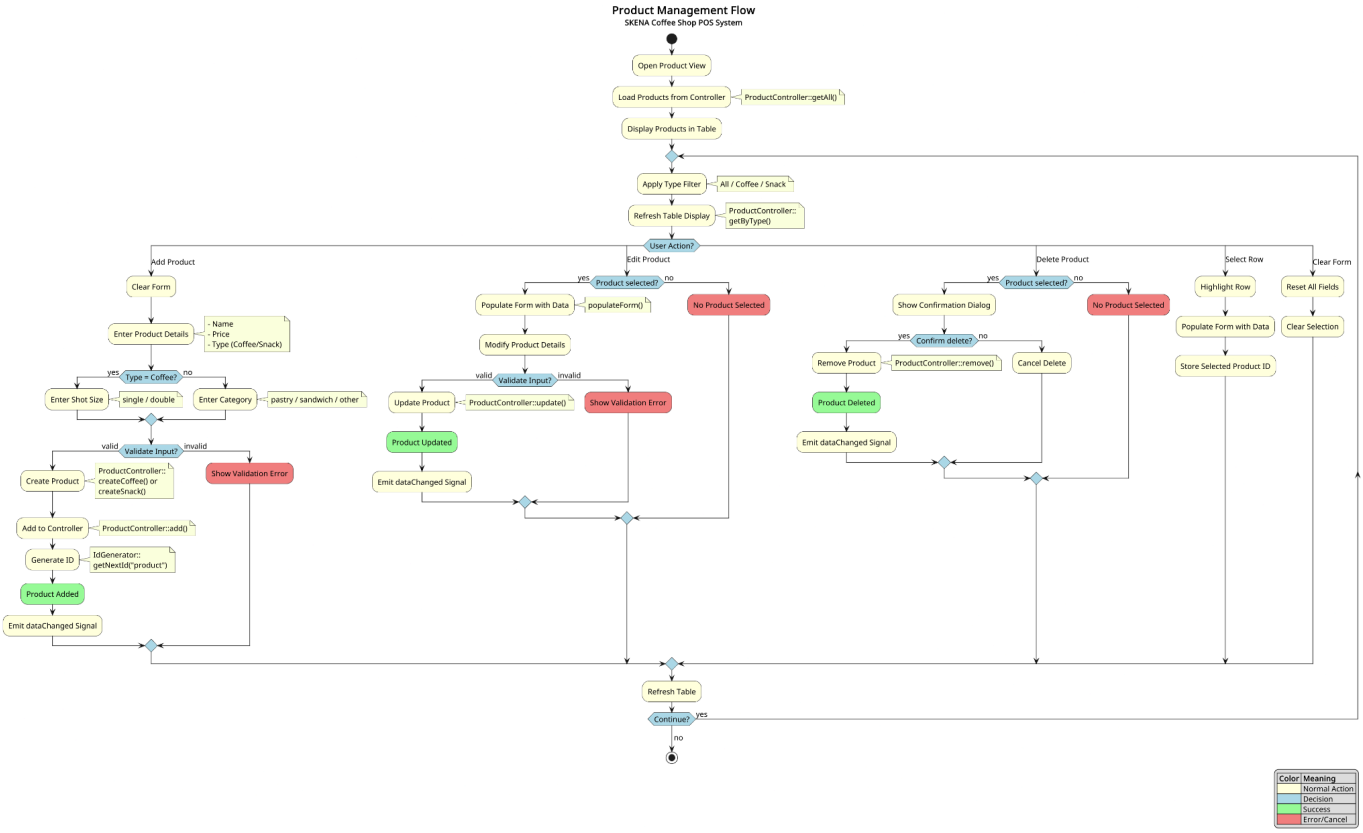
Gambar 7 User Flow Transaksi



3.5.2 User Flow Produk

User Flow produk harus bisa mengintegrasikan proses bisnis P-01, P-02, P-03, P-04, dan P-05 secara mudah. Detail dari *user flow* untuk *produk* dapat dilihat di Gambar 8.

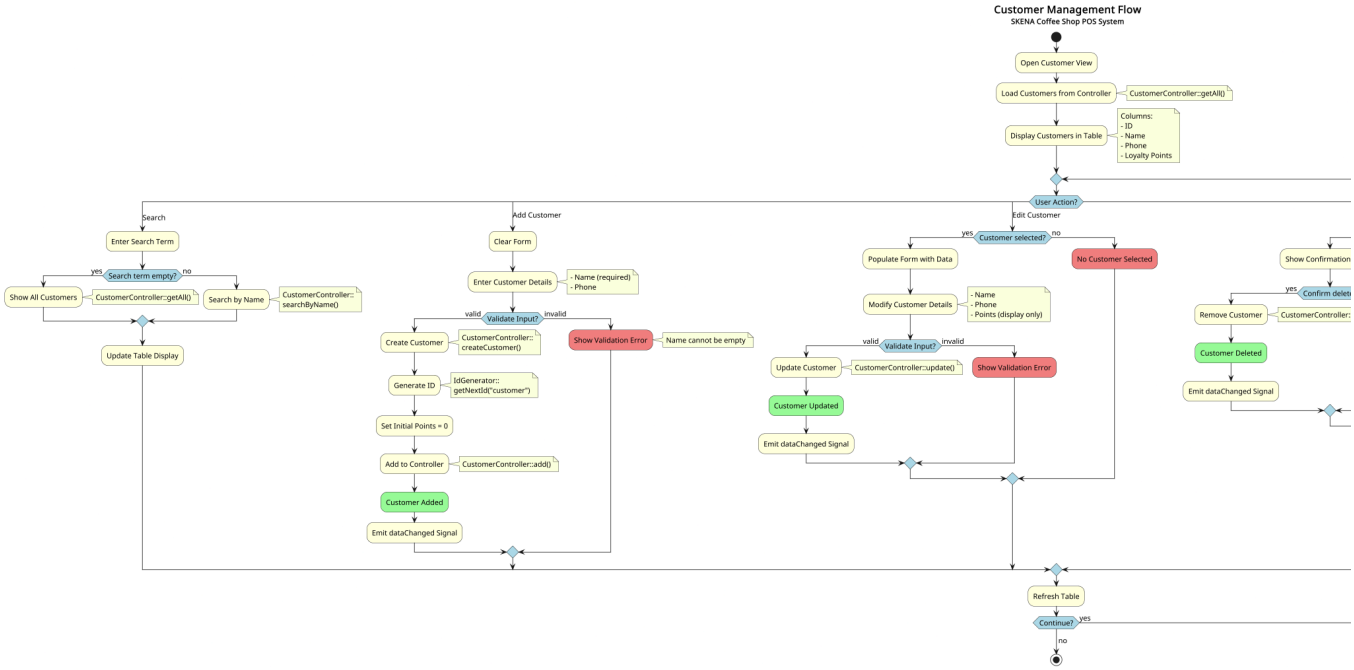
Gambar 8 User Flow Produk



3.5.3 User Flow *Customer*

User Flow customer harus bisa mengintegrasikan proses bisnis C-01, C-02, C-03, C-04, C-05, dan C-06 secara mudah. Detail dari *user flow* untuk customer dapat dilihat di Gambar 9.

Gambar 9 User Flow Customer



3.6. Transformasi Class Diagram Menjadi Kode Program

Setelah membuat *class diagram*, langkah selanjutnya yaitu mengubah setiap *class* yang ada di *class diagram* menjadi kode program. Kode program dibagi menjadi dua yaitu kode deklarasi atau kode kontrak yang mempunyai format .h dan kode implementasi yang mempunyai format .cpp. Kode deklarasi berfungsi untuk mendeklarasikan struktur seperti *property* dan *method* yang harus diimplementasikan di kode implementasi, sedangkan kode implementasi berfungsi untuk penerapan proses bisnis.

3.6.1 Transformasi Transaction Class Menjadi Kode Program

Berikut adalah transformasi *class* transaksi menjadi kode program. Kode program dibagi menjadi dua yaitu ada *header* dan *implementation*. Detail dari *header* dapat dilihat di Gambar 10 dan detail dari *implementation* dapat dilihat di Gambar 11.

Gambar 10 Header Code Transaction

```
/**
* @file Transaction.h
* @brief Transaction model class representing a complete order
*
* OOP Principles Applied:
```

```
* - Encapsulation: Private members with controlled access
* - Abstraction: Implements IEntity interface
*/

#ifndef TRANSACTION_H
#define TRANSACTION_H

#include "IEntity.h"
#include "TransactionItem.h"
#include <vector>
#include <string>

/**
* @class Transaction
* @brief Represents a complete sales transaction
*
* A transaction contains multiple items, customer information,
* and loyalty points data. It calculates totals and manages
* the points earned/used.
*/
class Transaction : public IEntity {
private:
   int m_id;                              ///< Unique transaction identifier
```

```
    int m_customerId;                       ///< Customer ID (0 for guest)
    std::string m_dateTime;                 ///< Transaction date/time
    std::vector<TransactionItem> m_items;   ///< Items in the transaction
    double m_subtotal;                      ///< Total before discount
    double m_discount;                      ///< Discount from points
    double m_total;                         ///< Final total after discount
    int m_pointsEarned;                     ///< Points earned from this
transaction
    int m_pointsUsed;                       ///< Points redeemed in this
transaction

public:
    /**
     * @brief Default constructor
     */
    Transaction();

    /**
     * @brief Parameterized constructor
     * @param id Transaction ID
     * @param customerId Customer ID (0 for guest)
     */
    Transaction(int id, int customerId = 0);
```

```
/**
 * @brief Virtual destructor
 */
virtual ~Transaction() = default;

// ============ IEntity Interface Implementation ============

int getId() const override;
std::string serialize() const override;
void deserialize(const std::string& data) override;
bool isValid() const override;

// ============ Getters ============

int getCustomerId() const;
std::string getDateTime() const;
const std::vector<TransactionItem>& getItems() const;
double getSubtotal() const;
double getDiscount() const;
double getTotal() const;
int getPointsEarned() const;
int getPointsUsed() const;
int getItemCount() const;
```

```cpp
// ============ Setters ============

void setId(int id);
void setCustomerId(int customerId);
void setDateTime(const std::string& dateTime);
void setPointsUsed(int points);

// ============ Item Management ============

/**
 * @brief Adds an item to the transaction
 * @param item TransactionItem to add
 *
 * If product already exists, quantity is incremented.
 */
void addItem(const TransactionItem& item);

/**
 * @brief Removes an item by product ID
 * @param productId Product ID to remove
 * @return true if removed, false if not found
 */
```

```
    bool removeItem(int productId);

    /**
     * @brief Updates quantity of an item
     * @param productId Product ID
     * @param quantity New quantity
     * @return true if updated, false if not found
     */
    bool updateItemQuantity(int productId, int quantity);

    /**
     * @brief Gets an item by product ID
     * @param productId Product ID to find
     * @return Pointer to item, nullptr if not found
     */
    TransactionItem* getItem(int productId);

    /**
     * @brief Clears all items from the transaction
     */
    void clearItems();

    /**
```

```
     * @brief Checks if transaction has any items
     * @return true if items exist
     */
    bool hasItems() const;

    // ============ Calculations ============

    /**
     * @brief Recalculates all totals
     *
     * Updates subtotal, discount, total, and points earned
     * based on current items and points used.
     */
    void recalculate();

    /**
     * @brief Calculates points that can be earned
     * @return Points earned based on final total
     */
    int calculatePointsEarned() const;

    /**
     * @brief Sets the current date/time
```

```
     */
    void setCurrentDateTime();

    // ============ Serialization Helpers ============

    /**
     * @brief Serializes items to compact format
     * @return "productId:qty,productId:qty,..." format
     */
    std::string serializeItems() const;

    /**
     * @brief Deserializes items from compact format
     * @param data Items string in compact format
     */
    void deserializeItems(const std::string& data);
};

#endif // TRANSACTION_H
```

Gambar 11 Implementation Code Transaction

```cpp
/**
* @file Transaction.cpp
* @brief Implementation of Transaction class
*/

#include "Transaction.h"
#include "Customer.h"
#include "../utils/FileManager.h"
#include <sstream>
#include <chrono>
#include <iomanip>
#include <algorithm>

Transaction::Transaction()
    : m_id(0)
    , m_customerId(0)
    , m_dateTime("")
    , m_subtotal(0.0)
    , m_discount(0.0)
    , m_total(0.0)
    , m_pointsEarned(0)
    , m_pointsUsed(0)
{
```

```
}


Transaction::Transaction(int id, int customerId)
    : m_id(id)
    , m_customerId(customerId)
    , m_dateTime("")
    , m_subtotal(0.0)
    , m_discount(0.0)
    , m_total(0.0)
    , m_pointsEarned(0)
    , m_pointsUsed(0)
{
    setCurrentDateTime();
}


int Transaction::getId() const {
    return m_id;
}


std::string Transaction::serialize() const {
    std::ostringstream oss;
    oss << m_id << "|"
        << m_customerId << "|"
```

```
        << m_dateTime << "|"
        << std::fixed << std::setprecision(0) << m_total << "|"
        << m_pointsEarned << "|"
        << m_pointsUsed << "|"
        << serializeItems();
    return oss.str();
}

void Transaction::deserialize(const std::string& data) {
    std::vector<std::string> fields = FileManager::splitLine(data, '|');

    if (fields.size() >= 7) {
        m_id = std::stoi(fields[0]);
        m_customerId = std::stoi(fields[1]);
        m_dateTime = fields[2];
        m_total = std::stod(fields[3]);
        m_pointsEarned = std::stoi(fields[4]);
        m_pointsUsed = std::stoi(fields[5]);
        deserializeItems(fields[6]);

        // Recalculate discount from points used
        m_discount = Customer::calculatePointsValue(m_pointsUsed);
        m_subtotal = m_total + m_discount;
```

```
    }
}

bool Transaction::isValid() const {
    return m_id > 0 && !m_items.empty();
}

int Transaction::getCustomerId() const {
    return m_customerId;
}

std::string Transaction::getDateTime() const {
    return m_dateTime;
}

const std::vector<TransactionItem>& Transaction::getItems() const {
    return m_items;
}

double Transaction::getSubtotal() const {
    return m_subtotal;
}
```

```cpp
double Transaction::getDiscount() const {
    return m_discount;
}

double Transaction::getTotal() const {
    return m_total;
}

int Transaction::getPointsEarned() const {
    return m_pointsEarned;
}

int Transaction::getPointsUsed() const {
    return m_pointsUsed;
}

int Transaction::getItemCount() const {
    int count = 0;
    for (const auto& item : m_items) {
        count += item.getQuantity();
    }
    return count;
}
```

```cpp
void Transaction::setId(int id) {
    m_id = id;
}

void Transaction::setCustomerId(int customerId) {
    m_customerId = customerId;
}

void Transaction::setDateTime(const std::string& dateTime) {
    m_dateTime = dateTime;
}

void Transaction::setPointsUsed(int points) {
    if (points >= 0) {
        m_pointsUsed = points;
        recalculate();
    }
}

void Transaction::addItem(const TransactionItem& item) {
    // Check if product already exists
    for (auto& existingItem : m_items) {
```

```
        if (existingItem.getProductId() == item.getProductId()) {
            existingItem.incrementQuantity(item.getQuantity());
            recalculate();
            return;
        }
    }

    // Add new item
    m_items.push_back(item);
    recalculate();
}

bool Transaction::removeItem(int productId) {
    auto it = std::find_if(m_items.begin(), m_items.end(),
        [productId](const TransactionItem& item) {
            return item.getProductId() == productId;
        });

    if (it != m_items.end()) {
        m_items.erase(it);
        recalculate();
        return true;
    }
```

```
    return false;
}

bool Transaction::updateItemQuantity(int productId, int quantity) {
    for (auto& item : m_items) {
        if (item.getProductId() == productId) {
            if (quantity <= 0) {
                return removeItem(productId);
            }
            item.setQuantity(quantity);
            recalculate();
            return true;
        }
    }
    return false;
}

TransactionItem* Transaction::getItem(int productId) {
    for (auto& item : m_items) {
        if (item.getProductId() == productId) {
            return &item;
        }
    }
```

```
    return nullptr;
}

void Transaction::clearItems() {
    m_items.clear();
    recalculate();
}

bool Transaction::hasItems() const {
    return !m_items.empty();
}

void Transaction::recalculate() {
    // Calculate subtotal
    m_subtotal = 0.0;
    for (const auto& item : m_items) {
        m_subtotal += item.getSubtotal();
    }

    // Calculate discount from points
    m_discount = Customer::calculatePointsValue(m_pointsUsed);

    // Ensure discount doesn't exceed subtotal (no negative totals)
```

```
    if (m_discount > m_subtotal) {
        m_discount = m_subtotal;
    }

    // Calculate final total
    m_total = m_subtotal - m_discount;

    // Calculate points earned (based on final total)
    m_pointsEarned = calculatePointsEarned();
}

int Transaction::calculatePointsEarned() const {
    return Customer::calculatePointsForAmount(m_total);
}

void Transaction::setCurrentDateTime() {
    auto now = std::chrono::system_clock::now();
    auto time = std::chrono::system_clock::to_time_t(now);
    std::ostringstream oss;
    oss << std::put_time(std::localtime(&time), "%Y-%m-%d %H:%M:%S");
    m_dateTime = oss.str();
}
```

```
std::string Transaction::serializeItems() const {
    if (m_items.empty()) {
        return "";
    }

    std::ostringstream oss;
    for (size_t i = 0; i < m_items.size(); ++i) {
        if (i > 0) {
            oss << ",";
        }
        oss << m_items[i].serializeCompact();
    }
    return oss.str();
}

void Transaction::deserializeItems(const std::string& data) {
    m_items.clear();

    if (data.empty()) {
        return;
    }

    // Split by comma
```

```
    std::vector<std::string> itemStrings = FileManager::splitLine(data, ',');


    for (const auto& itemStr : itemStrings) {
        TransactionItem item;
        item.deserializeCompact(itemStr);
        if (item.getProductId() > 0) {
            m_items.push_back(item);
        }
    }
}
```

3.6.2 Transformasi Product Class Menjadi Kode Program

Berikut adalah transformasi *class* produk menjadi kode program. Kode program dibagi menjadi dua yaitu ada *header* dan *implementation*. Detail dari *header* dapat dilihat di Gambar 12 dan detail dari *implementation* dapat dilihat di Gambar 13.

Gambar 12 Header Code Product

```
/**
* @file Product.h
* @brief Abstract base class for all product types
*
* OOP Principles Applied:
* - Abstraction: Abstract class with pure virtual methods
* - Encapsulation: Private/protected members with public accessors
* - Inheritance: Base class for Coffee and Snack
*
```

```
* SOLID Principles Applied:
* - Open/Closed: Open for extension (new product types), closed for
modification
* - Liskov Substitution: Derived classes can substitute for Product
*/

#ifndef PRODUCT_H
#define PRODUCT_H

#include "IEntity.h"
#include <string>

/**
* @class Product
* @brief Abstract base class for all products in the coffee shop
*
* This class defines the common interface and data for all products.
* Coffee and Snack classes inherit from this class and provide
* specific implementations.
*/
class Product : public IEntity {
protected:
   int m_id;              ///< Unique product identifier
   std::string m_name;    ///< Product name
```

```
    double m_price;         ///< Price in IDR
    std::string m_type;     ///< Product type ("coffee" or "snack")

public:
    /**
     * @brief Default constructor
     */
    Product();

    /**
     * @brief Parameterized constructor
     * @param id Unique identifier
     * @param name Product name
     * @param price Price in IDR
     * @param type Product type
     */
    Product(int id, const std::string& name, double price, const std::string&
type);

    /**
     * @brief Virtual destructor
     */
    virtual ~Product() = default;
```

```
// ============ IEntity Interface Implementation ============

int getId() const override;
bool isValid() const override;

// serialize() and deserialize() are implemented by derived classes

// ============ Getters ============

/**
 * @brief Gets the product name
 * @return Product name
 */
std::string getName() const;

/**
 * @brief Gets the product price
 * @return Price in IDR
 */
double getPrice() const;

/**
 * @brief Gets the product type
```

```
     * @return "coffee" or "snack"
     */
    std::string getType() const;

    // ============ Setters ============

    /**
     * @brief Sets the product ID
     * @param id New ID value
     */
    void setId(int id);

    /**
     * @brief Sets the product name
     * @param name New name
     */
    void setName(const std::string& name);

    /**
     * @brief Sets the product price
     * @param price New price in IDR
     */
    void setPrice(double price);
```

```
    // ============ Virtual Methods (Polymorphism) ============

    /**
     * @brief Gets a detailed description of the product
     * @return Product description string
     *
     * Pure virtual - must be implemented by derived classes
     */
    virtual std::string getDescription() const = 0;

    /**
     * @brief Calculates total price for a quantity
     * @param quantity Number of items
     * @return Total price
     *
     * Virtual with default implementation - can be overridden
     */
    virtual double calculatePrice(int quantity) const;

    /**
     * @brief Gets the extra field value specific to product type
     * @return Extra field value (e.g., shot size for coffee, category for
snack)
```

```
    */
    virtual std::string getExtraField() const = 0;

    /**
     * @brief Sets the extra field value specific to product type
     * @param value New extra field value
     */
    virtual void setExtraField(const std::string& value) = 0;

    /**
     * @brief Creates a clone of this product
     * @return Pointer to a new Product copy
     */
    virtual Product* clone() const = 0;
};

#endif // PRODUCT_H
```

Gambar 13 Implementation Code Product

```
/**
* @file Product.cpp
* @brief Implementation of Product base class
*/
```

```cpp
#include "Product.h"


Product::Product()
   : m_id(0)
   , m_name("")
   , m_price(0.0)
   , m_type("")
{
}


Product::Product(int id, const std::string& name, double price, const
std::string& type)
   : m_id(id)
   , m_name(name)
   , m_price(price)
   , m_type(type)
{
}


int Product::getId() const {
   return m_id;
}


bool Product::isValid() const {
```

```
    return m_id > 0 && !m_name.empty() && m_price >= 0;
}

std::string Product::getName() const {
    return m_name;
}

double Product::getPrice() const {
    return m_price;
}

std::string Product::getType() const {
    return m_type;
}

void Product::setId(int id) {
    m_id = id;
}

void Product::setName(const std::string& name) {
    m_name = name;
}
```

```
void Product::setPrice(double price) {
    if (price ≥ 0) {
        m_price = price;
    }
}

double Product::calculatePrice(int quantity) const {
    if (quantity ≤ 0) {
        return 0.0;
    }
    return m_price * quantity;
}
```

3.6.3 Transformasi Customer Class Menjadi Kode Program

Berikut adalah transformasi *class customer* menjadi kode program. Kode program dibagi menjadi dua yaitu ada *header* dan *implementation*. Detail dari *header* dapat dilihat di Gambar 14 dan detail dari *implementation* dapat dilihat di Gambar 15.

Gambar 14 Header Code Customer

```
/**
* @file Customer.h
* @brief Customer model class with loyalty points functionality
*
* OOP Principles Applied:
* - Encapsulation: Private members with controlled access via methods
* - Abstraction: Implements IEntity interface
```

```
*
* SOLID Principles Applied:
* - Single Responsibility: Only manages customer data and loyalty points
*/

#ifndef CUSTOMER_H
#define CUSTOMER_H

#include "IEntity.h"
#include <string>

/**
* @class Customer
* @brief Represents a customer in the loyalty program
*
* Customers can earn and redeem loyalty points based on their purchases.
* Points are earned at 1 point per 1,000 IDR and redeemed at 100 IDR per
point.
*/
class Customer : public IEntity {
private:
   int m_id;                ///< Unique customer identifier
   std::string m_name;      ///< Customer name
   std::string m_phone;     ///< Phone number
```

```
    int m_loyaltyPoints;      ///< Current loyalty points balance

public:
    // ============ Constants ============

    static constexpr int POINTS_PER_UNIT = 1000;      ///< IDR per point earned
    static constexpr int POINT_VALUE = 100;           ///< IDR value per point
    static constexpr int MIN_REDEEM_POINTS = 10;      ///< Minimum points to
redeem

    // ============ Constructors ============

    /**
     * @brief Default constructor
     */
    Customer();

    /**
     * @brief Parameterized constructor
     * @param id Unique identifier
     * @param name Customer name
     * @param phone Phone number
     * @param loyaltyPoints Initial loyalty points (default: 0)
     */
```

```
    Customer(int id, const std::string& name, const std::string& phone, int
loyaltyPoints = 0);

    /**
     * @brief Virtual destructor
     */
    virtual ~Customer() = default;

    // ============ IEntity Interface Implementation ============

    int getId() const override;
    std::string serialize() const override;
    void deserialize(const std::string& data) override;
    bool isValid() const override;

    // ============ Getters ============

    /**
     * @brief Gets the customer name
     * @return Customer name
     */
    std::string getName() const;

    /**
```

```
     * @brief Gets the phone number
     * @return Phone number string
     */
    std::string getPhone() const;

    /**
     * @brief Gets current loyalty points balance
     * @return Number of points
     */
    int getLoyaltyPoints() const;

    // ============ Setters ============

    /**
     * @brief Sets the customer ID
     * @param id New ID
     */
    void setId(int id);

    /**
     * @brief Sets the customer name
     * @param name New name
     */
```

```
    void setName(const std::string& name);

    /**
     * @brief Sets the phone number
     * @param phone New phone number
     */
    void setPhone(const std::string& phone);

    /**
     * @brief Sets the loyalty points balance
     * @param points New points balance
     */
    void setLoyaltyPoints(int points);

    // ============ Loyalty Points Operations ============

    /**
     * @brief Adds points to the customer's balance
     * @param points Number of points to add
     */
    void addPoints(int points);

    /**
```

```
     * @brief Attempts to redeem points
     * @param points Number of points to redeem
     * @return true if successful, false if insufficient points
     */
    bool redeemPoints(int points);

    /**
     * @brief Calculates points earned for a transaction amount
     * @param amount Transaction amount in IDR
     * @return Number of points earned
     */
    static int calculatePointsForAmount(double amount);

    /**
     * @brief Calculates the IDR value of points
     * @param points Number of points
     * @return Value in IDR
     */
    static double calculatePointsValue(int points);

    /**
     * @brief Checks if points can be redeemed
     * @param points Points to check
```

```
    * @return true if enough points available
    */
   bool canRedeemPoints(int points) const;
};


#endif // CUSTOMER_H
```

Gambar 15 Implementation Code Customer

```
/**
* @file Customer.cpp
* @brief Implementation of Customer class
*/

#include "Customer.h"
#include "../utils/FileManager.h"
#include <sstream>

Customer::Customer()
   : m_id(0)
   , m_name("")
   , m_phone("")
   , m_loyaltyPoints(0)
{
}
```

```
Customer::Customer(int id, const std::string& name, const std::string& phone,
int loyaltyPoints)
    : m_id(id)
    , m_name(name)
    , m_phone(phone)
    , m_loyaltyPoints(loyaltyPoints)
{
}

int Customer::getId() const {
    return m_id;
}

std::string Customer::serialize() const {
    std::ostringstream oss;
    oss << m_id << "|" << m_name << "|" << m_phone << "|" << m_loyaltyPoints;
    return oss.str();
}

void Customer::deserialize(const std::string& data) {
    std::vector<std::string> fields = FileManager::splitLine(data, '|');

    if (fields.size() >= 4) {
```

```
        m_id = std::stoi(fields[0]);
        m_name = fields[1];
        m_phone = fields[2];
        m_loyaltyPoints = std::stoi(fields[3]);
    }
}

bool Customer::isValid() const {
    return m_id > 0 && !m_name.empty();
}

std::string Customer::getName() const {
    return m_name;
}

std::string Customer::getPhone() const {
    return m_phone;
}

int Customer::getLoyaltyPoints() const {
    return m_loyaltyPoints;
}
```

```cpp
void Customer::setId(int id) {
    m_id = id;
}

void Customer::setName(const std::string& name) {
    m_name = name;
}

void Customer::setPhone(const std::string& phone) {
    m_phone = phone;
}

void Customer::setLoyaltyPoints(int points) {
    if (points ≥ 0) {
        m_loyaltyPoints = points;
    }
}

void Customer::addPoints(int points) {
    if (points > 0) {
        m_loyaltyPoints += points;
    }
}
```

```cpp
bool Customer::redeemPoints(int points) {
    if (canRedeemPoints(points)) {
        m_loyaltyPoints -= points;
        return true;
    }
    return false;
}

int Customer::calculatePointsForAmount(double amount) {
    if (amount <= 0) {
        return 0;
    }
    // 1 point per 1,000 IDR (integer division)
    return static_cast<int>(amount / POINTS_PER_UNIT);
}

double Customer::calculatePointsValue(int points) {
    if (points <= 0) {
        return 0.0;
    }
    // Each point is worth 100 IDR
    return points * POINT_VALUE;
```

```
}


bool Customer::canRedeemPoints(int points) const {
    return points ≥ MIN_REDEEM_POINTS && points ≤ m_loyaltyPoints;
}
```

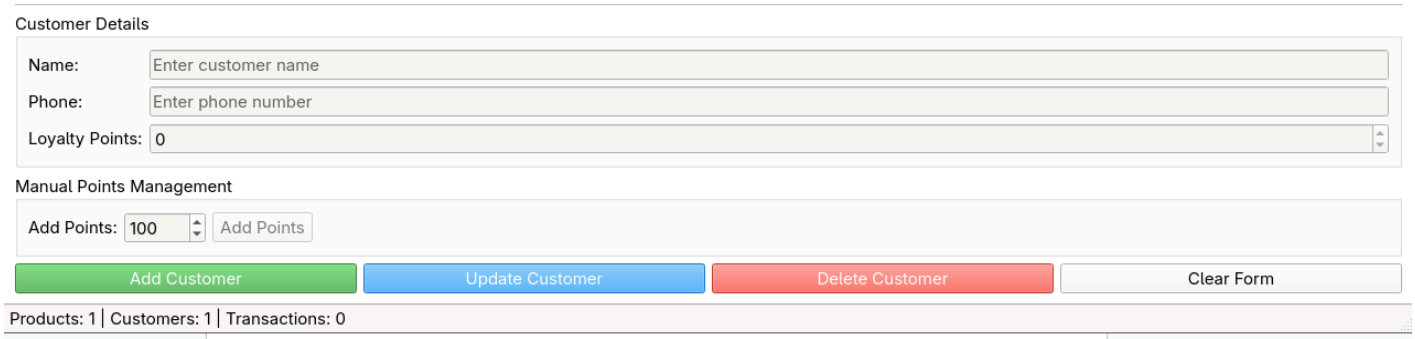
4. IMPLEMENTASI APLIKASI

Setelah mengubah *class diagram* menjadi kode, akhirnya dilakukan perancangan sistem yang detail. Perancangan ini yaitu mengubah fitur atau proses bisnis ke dalam bentuk kode. Kode yang mengandung proses bisnis ditandai dengan nama *controller*.

4.1. Fitur C-01 Register Customer

Fitur untuk menambahkan *customer* baru. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 16. Terlihat tombol *Add Customer* untuk menambahkan *customer* baru.

Gambar 16 Tampilan Antarmuka Fitur C-01, C-02, dan C-03



4.2. Fitur C-02 Edit Customer Data

Fitur untuk merubah data *customer*. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 16. Terlihat tombol *Update Customer* untuk merubah data *customer*.

4.3. Fitur C-03 Delete Customer Data

Fitur untuk menghapus *customer*. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 16. Terlihat tombol *Delete Customer* untuk menghapus *customer*.

4.4. Fitur C-04 View All Customer

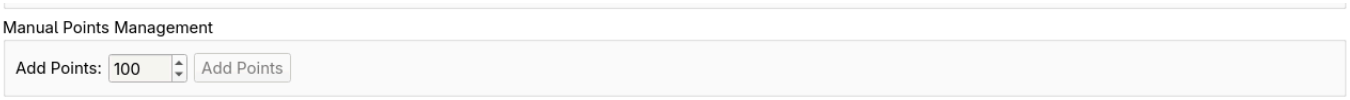
Fitur untuk melihat semua *customer*. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 17. Terlihat tombol *Show All* untuk melihat semua data customer.

Gambar 17 Tampilan Antarmuka Fitur C-04

4.5. Fitur C-05 Insert Loyalty Point

Fitur untuk menambahkan *loyalty point*. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 18. Terlihat tombol *Add Points* untuk menambahkan *loyalty point* secara manual.

Gambar 18 Tampilan Antarmuka Fitur C-05



4.6. Fitur C-06 Search Customer

Fitur untuk mencari *customer*. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 19. Terlihat tombol *Search* dan *text box* untuk masukkan berupa nama.

Gambar 19 Tampilan Antarmuka Fitur C-06

4.7. Fitur P-01 Register Product

Fitur untuk menambahkan produk baru. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 20. Terlihat tombol *Add Product* untuk menambahkan produk baru.

Gambar 20 Tampilan Antarmuka Fitur P-01, P-02, dan P-03

4.8. Fitur P-02 Edit Product Data

Fitur untuk mengubah data produk. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 20. Terlihat tombol *Update Product* untuk mengubah data produk.

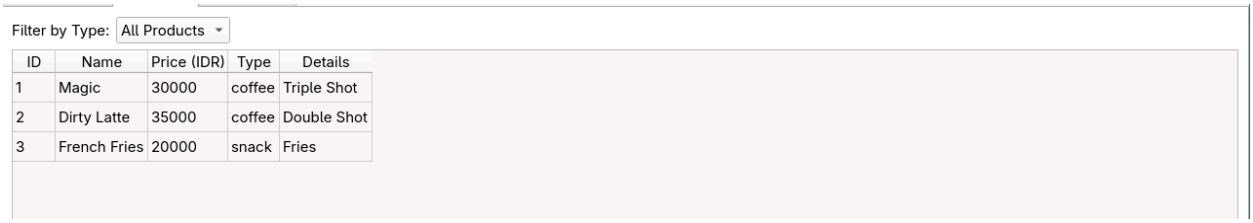
4.9. Fitur P-03 Delete Product Data

Fitur untuk menghapus produk. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 20. Terlihat tombol *Delete Product* untuk menghapus produk.

4.10. Fitur P-04 View All Product

Fitur untuk melihat semua produk. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 21. Terlihat tombol *All Products* untuk melihat semua produk.

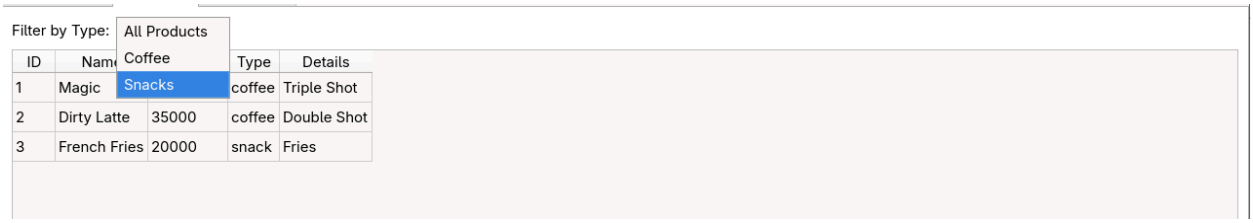
Gambar 21 Tampilan Antarmuka Fitur P-04



4.11. Fitur P-05 Filter Product

Fitur untuk melihat semua produk berdasarkan tipe produk. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 22. Terlihat *label Filter by Type* untuk melihat semua produk berdasarkan tipe produk.

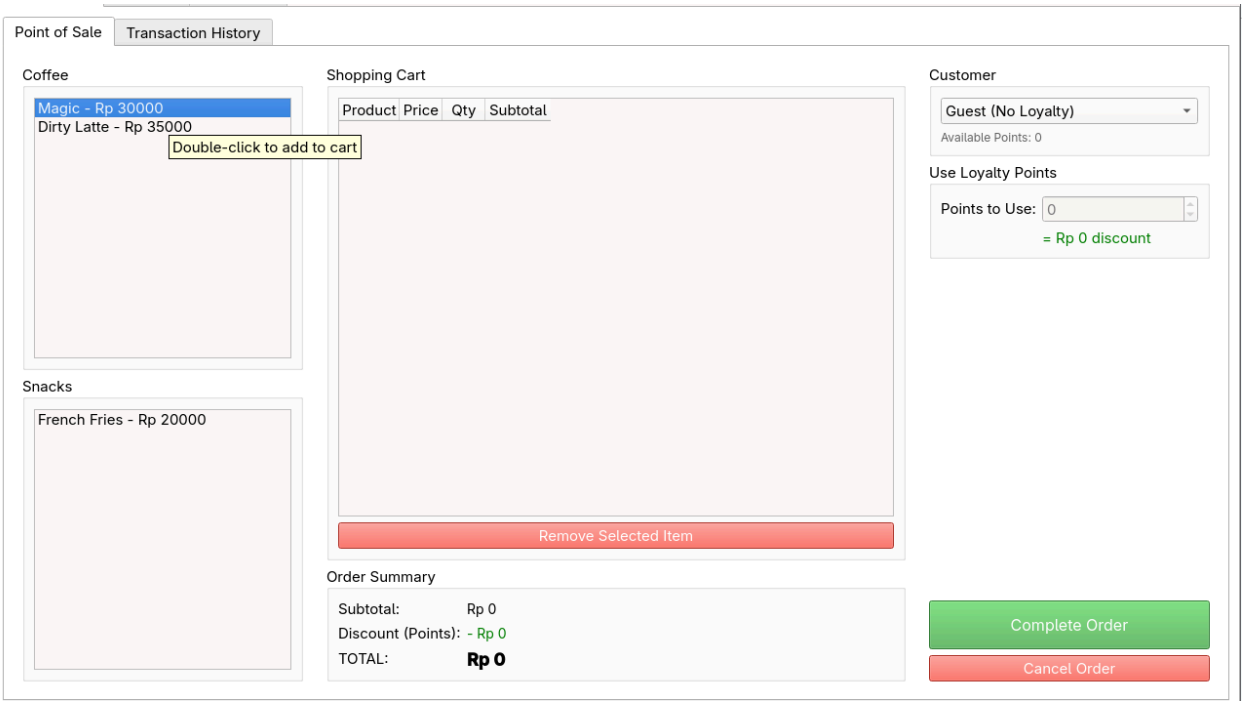
Gambar 21 Tampilan Antarmuka Fitur P-04



4.12. Fitur POS-01 Add to Cart

Fitur untuk menambahkan produk ke keranjang. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 22. Terlihat ada *dialog* untuk menyuruh pengguna menekan dua kali di produk agar menambahkan produk ke keranjang.

Gambar 22 Tampilan Antarmuka Fitur P-01, P-02, P-04



4.13. Fitur POS-02 Remove from Cart

Fitur untuk menghapus produk dari keranjang. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 22. Terlihat ada *button Remove Selected Item* untuk menghapus produk dari keranjang.

4.14. Fitur POS-03 Update Quantity

Fitur untuk mengubah kuantitas produk di keranjang. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 23. Terlihat ada *Input Stepper* mengubah kuantitas produk di keranjang.

Gambar 23 Tampilan Antarmuka Fitur P-03

4.15. Fitur POS-04 Complete Transaction

Fitur untuk menyelesaikan transaksi di keranjang. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 22. Terlihat ada *Button* bernama *Complete Order* untuk menyelesaikan transaksi di keranjang.

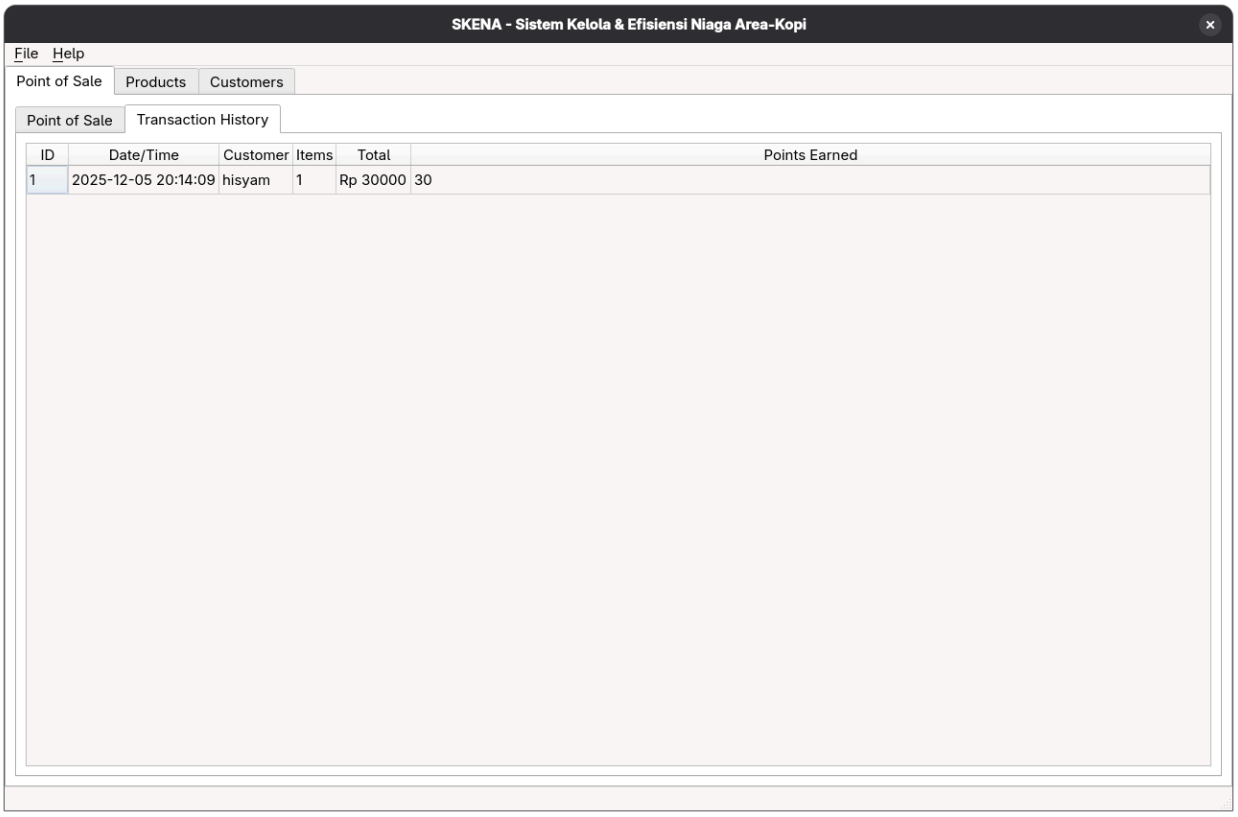
4.16. Fitur POS-05 Cancel Transaction

Fitur untuk membatalkan transaksi di keranjang. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 22. Terlihat ada *Button* bernama *Cancel Order* untuk membatalkan transaksi di keranjang.

4.17. Fitur POS-06 View History

Fitur untuk melihat riwayat transaksi. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 24. Terlihat ada *Button* bernama *Transaction History* untuk melihat riwayat transaksi.

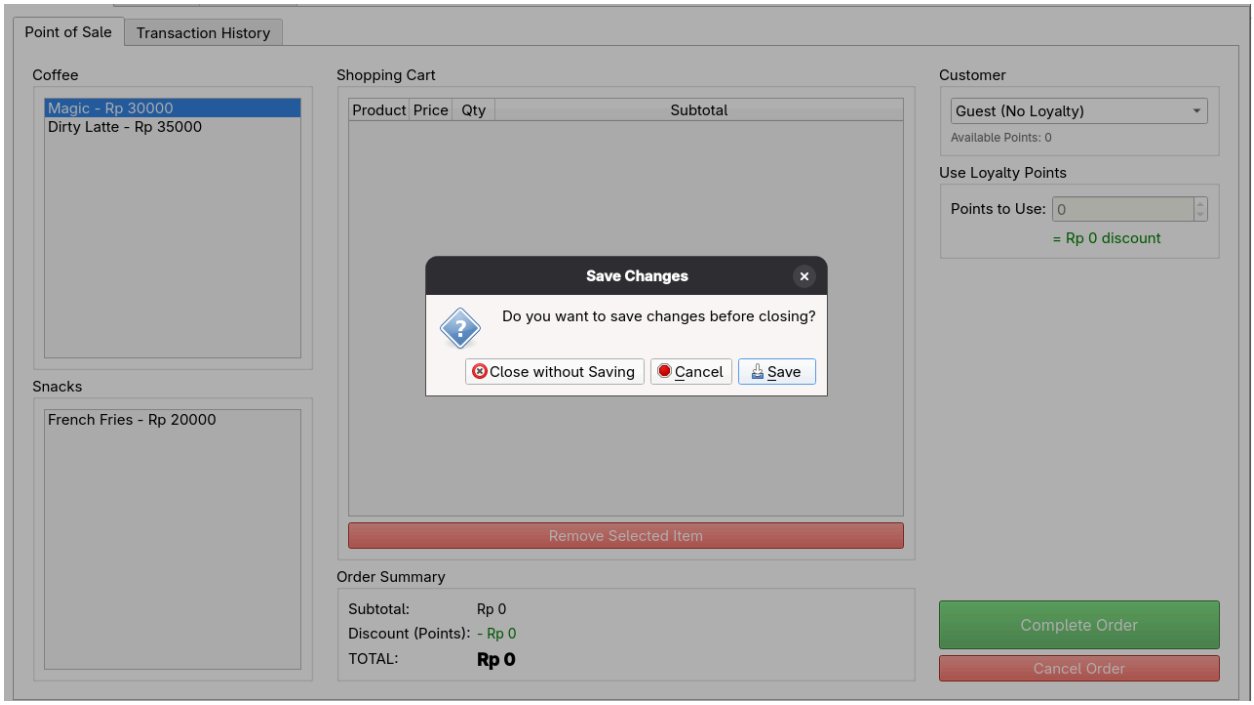
Gambar 24 Tampilan Antarmuka Fitur P-06



4.18. Fitur D-01 Save Data

Fitur untuk menyimpan semua data. Tampilan antarmuka dari fitur ini dapat dilihat pada Gambar 25. Terlihat ada *Button* bernama *Save* untuk menyimpan semua data.

Gambar 25 Tampilan Antarmuka Fitur D-01



4.19. Fitur D-02 Load Data

Fitur untuk memuat semua data. Fitur ini berjalan dengan sendirinya setiap pengguna menjalankan aplikasi.

LAMPIRAN

- Source code *IController.h*

```
/**
* @file IController.h
* @brief Abstract controller interface defining common operations
*
* SOLID Principles Applied:
* - Interface Segregation: Minimal interface with essential CRUD operations
* - Dependency Inversion: High-level modules depend on this abstraction
*/

#ifndef ICONTROLLER_H
#define ICONTROLLER_H

#include <vector>

/**
* @class IController
* @brief Template interface for all controllers
*
* This template class defines the contract for controller classes,
* providing standard CRUD operations and file I/O methods.
*
* @tparam T The type of entity this controller manages
*/
template<typename T>
class IController {
```

```
public:
    /**
     * @brief Virtual destructor
     */
    virtual ~IController() = default;

    // ============ CRUD Operations ============

    /**
     * @brief Gets all entities
     * @return Vector of all entities
     */
    virtual std::vector<T> getAll() const = 0;

    /**
     * @brief Gets an entity by ID
     * @param id Entity ID to find
     * @return Pointer to entity, nullptr if not found
     */
    virtual T* getById(int id) = 0;

    /**
     * @brief Adds a new entity
     * @param item Entity to add
     * @return true if successful
     */
    virtual bool add(const T& item) = 0;
```

```
    /**
     * @brief Updates an existing entity
     * @param item Entity with updated data
     * @return true if found and updated
     */
    virtual bool update(const T& item) = 0;

    /**
     * @brief Removes an entity by ID
     * @param id Entity ID to remove
     * @return true if found and removed
     */
    virtual bool remove(int id) = 0;

    // ============ File I/O ============

    /**
     * @brief Saves all entities to file
     * @return true if successful
     */
    virtual bool saveToFile() = 0;

    /**
     * @brief Loads all entities from file
     * @return true if successful
     */
    virtual bool loadFromFile() = 0;
```

```
    // ============ Query ============

    /**
     * @brief Gets the total count of entities
     * @return Number of entities
     */
    virtual int getCount() const = 0;
};

#endif // ICONTROLLER_H
```

- Source code CustomerController.h

```
/**
* @file CustomerController.h
* @brief Controller for managing customers and loyalty points
*
* SOLID Principles Applied:
* - Single Responsibility: Only manages customer operations
* - Dependency Inversion: Depends on FileManager abstraction
*/

#ifndef CUSTOMERCONTROLLER_H
#define CUSTOMERCONTROLLER_H

#include "IController.h"
#include "../models/Customer.h"
#include "../utils/FileManager.h"
#include <vector>
```

```cpp
/**
* @class CustomerController
* @brief Manages customer CRUD operations and loyalty points
*/
class CustomerController : public IController<Customer> {
private:
   std::vector<Customer> m_customers;  ///< Customer collection
   FileManager& m_fileManager;          ///< File I/O handler
   static constexpr const char* FILENAME = "customers.txt";

public:
   /**
    * @brief Constructs controller with FileManager dependency
    * @param fileManager Reference to FileManager for I/O
    */
   explicit CustomerController(FileManager& fileManager);

   /**
    * @brief Destructor
    */
   ~CustomerController() override = default;

   // ============ IController Interface Implementation ============

   std::vector<Customer> getAll() const override;
   Customer* getById(int id) override;
   bool add(const Customer& customer) override;
```

```
    bool update(const Customer& customer) override;
    bool remove(int id) override;
    bool saveToFile() override;
    bool loadFromFile() override;
    int getCount() const override;

    // ============ Customer-Specific Methods ============

    /**
     * @brief Creates a new customer with generated ID
     * @param name Customer name
     * @param phone Phone number
     * @return Newly created Customer
     */
    Customer createCustomer(const std::string& name, const std::string&
phone);

    /**
     * @brief Searches customers by name (partial match)
     * @param searchTerm Search term
     * @return Vector of matching customers
     */
    std::vector<Customer*> searchByName(const std::string& searchTerm);

    /**
     * @brief Gets customer by phone number
     * @param phone Phone number to search
     * @return Pointer to customer, nullptr if not found
```

```
     */
    Customer* getByPhone(const std::string& phone);

    // ============ Loyalty Points Operations ============

    /**
     * @brief Adds loyalty points to a customer
     * @param customerId Customer ID
     * @param points Points to add
     * @return true if successful
     */
    bool addLoyaltyPoints(int customerId, int points);

    /**
     * @brief Redeems loyalty points from a customer
     * @param customerId Customer ID
     * @param points Points to redeem
     * @return true if successful (sufficient points)
     */
    bool redeemLoyaltyPoints(int customerId, int points);

    /**
     * @brief Gets current loyalty points for a customer
     * @param customerId Customer ID
     * @return Points balance, -1 if customer not found
     */
    int getLoyaltyPoints(int customerId) const;
};
```

```
#endif // CUSTOMERCONTROLLER_H
```

- Source code CustomerController.cpp

```
/**
* @file CustomerController.cpp
* @brief Implementation of CustomerController class
*/

#include "CustomerController.h"
#include "../utils/IdGenerator.h"
#include <algorithm>

CustomerController::CustomerController(FileManager& fileManager)
    : m_fileManager(fileManager)
{
}

std::vector<Customer> CustomerController::getAll() const {
    return m_customers;
}

Customer* CustomerController::getById(int id) {
    auto it = std::find_if(m_customers.begin(), m_customers.end(),
        [id](const Customer& c) {
            return c.getId() == id;
        });
```

```
    if (it != m_customers.end()) {
        return &(*it);
    }
    return nullptr;
}

bool CustomerController::add(const Customer& customer) {
    if (!customer.isValid()) {
        return false;
    }

    // Check for duplicate ID
    if (getById(customer.getId()) != nullptr) {
        return false;
    }

    // Update ID generator
    IdGenerator::getInstance().updateCounter("customer",
customer.getId());

    m_customers.push_back(customer);
    return true;
}

bool CustomerController::update(const Customer& customer) {
    Customer* existing = getById(customer.getId());
    if (!existing) {
        return false;
```

```
    }

    existing->setName(customer.getName());
    existing->setPhone(customer.getPhone());
    existing->setLoyaltyPoints(customer.getLoyaltyPoints());

    return true;
}

bool CustomerController::remove(int id) {
    auto it = std::find_if(m_customers.begin(), m_customers.end(),
        [id](const Customer& c) {
            return c.getId() == id;
        });

    if (it != m_customers.end()) {
        m_customers.erase(it);
        return true;
    }
    return false;
}

bool CustomerController::saveToFile() {
    std::vector<std::string> lines;
    lines.reserve(m_customers.size());

    for (const auto& customer : m_customers) {
        lines.push_back(customer.serialize());
```

```
    }

    return m_fileManager.writeLines(FILENAME, lines);
}

bool CustomerController::loadFromFile() {
    m_customers.clear();

    std::vector<std::string> lines = m_fileManager.readLines(FILENAME);

    for (const auto& line : lines) {
        if (line.empty()) continue;

        Customer customer;
        customer.deserialize(line);

        if (customer.isValid()) {
            IdGenerator::getInstance().updateCounter("customer",
customer.getId());
            m_customers.push_back(customer);
        }
    }

    return true;
}

int CustomerController::getCount() const {
    return static_cast<int>(m_customers.size());
```

```
}


Customer CustomerController::createCustomer(const std::string& name,
const std::string& phone) {
    int id = IdGenerator::getInstance().getNextId("customer");
    return Customer(id, name, phone, 0);
}


std::vector<Customer*> CustomerController::searchByName(const
std::string& searchTerm) {
    std::vector<Customer*> results;

    // Convert search term to lowercase for case-insensitive search
    std::string lowerSearch = searchTerm;
    std::transform(lowerSearch.begin(), lowerSearch.end(),
lowerSearch.begin(), ::tolower);

    for (auto& customer : m_customers) {
        std::string lowerName = customer.getName();
        std::transform(lowerName.begin(), lowerName.end(),
lowerName.begin(), ::tolower);

        if (lowerName.find(lowerSearch) != std::string::npos) {
            results.push_back(&customer);
        }
    }

    return results;
}
```

```
Customer* CustomerController::getByPhone(const std::string& phone) {
    auto it = std::find_if(m_customers.begin(), m_customers.end(),
        [&phone](const Customer& c) {
            return c.getPhone() == phone;
        });

    if (it != m_customers.end()) {
        return &(*it);
    }
    return nullptr;
}

bool CustomerController::addLoyaltyPoints(int customerId, int points) {
    Customer* customer = getById(customerId);
    if (!customer || points <= 0) {
        return false;
    }

    customer->addPoints(points);
    return true;
}

bool CustomerController::redeemLoyaltyPoints(int customerId, int points)
{
    Customer* customer = getById(customerId);
    if (!customer) {
        return false;
```

```
    }

    return customer->redeemPoints(points);
}

int CustomerController::getLoyaltyPoints(int customerId) const {
    auto it = std::find_if(m_customers.begin(), m_customers.end(),
        [customerId](const Customer& c) {
            return c.getId() == customerId;
        });

    if (it != m_customers.end()) {
        return it->getLoyaltyPoints();
    }
    return -1;
}
```

- Source code ProductController.h

```
/**
* @file ProductController.h
* @brief Controller for managing products (Coffee and Snacks)
*
* SOLID Principles Applied:
* - Single Responsibility: Only manages product operations
* - Dependency Inversion: Depends on FileManager abstraction
* - Open/Closed: Can work with any Product subclass
*/
```

```cpp
#ifndef PRODUCTCONTROLLER_H
#define PRODUCTCONTROLLER_H

#include "IController.h"
#include "../models/Product.h"
#include "../utils/FileManager.h"
#include <memory>
#include <vector>

/**
* @class ProductController
* @brief Manages product CRUD operations and persistence
*
* This controller handles both Coffee and Snack products using
* polymorphism. Products are stored as unique_ptr for memory safety.
*/
class ProductController {
private:
   std::vector<std::unique_ptr<Product>> m_products;  ///< Product collection
   FileManager& m_fileManager;                        ///< File I/O handler
   static constexpr const char* FILENAME = "products.txt";

public:
   /**
    * @brief Constructs controller with FileManager dependency
    * @param fileManager Reference to FileManager for I/O
    */
```

```
    explicit ProductController(FileManager& fileManager);

    /**
     * @brief Destructor
     */
    ~ProductController() = default;

    // ============ CRUD Operations ============

    /**
     * @brief Gets all products as raw pointers
     * @return Vector of product pointers
     */
    std::vector<Product*> getAll() const;

    /**
     * @brief Gets a product by ID
     * @param id Product ID
     * @return Pointer to product, nullptr if not found
     */
    Product* getById(int id);

    /**
     * @brief Gets products by type
     * @param type "coffee" or "snack"
     * @return Vector of matching products
     */
    std::vector<Product*> getByType(const std::string& type) const;
```

```
    /**
     * @brief Adds a new product (takes ownership)
     * @param product Unique pointer to product
     * @return true if successful
     */
    bool add(std::unique_ptr<Product> product);

    /**
     * @brief Updates an existing product
     * @param product Product with updated data
     * @return true if found and updated
     */
    bool update(const Product& product);

    /**
     * @brief Removes a product by ID
     * @param id Product ID to remove
     * @return true if found and removed
     */
    bool remove(int id);

    // ============ Factory Methods ============

    /**
     * @brief Creates a new Coffee product
     * @param name Product name
     * @param price Price in IDR
```

```
     * @param shotSize Shot size (single/double)
     * @return Unique pointer to new Coffee
     */
    std::unique_ptr<Product> createCoffee(const std::string& name, double
price,
                                          const std::string& shotSize =
"single");

    /**
     * @brief Creates a new Snack product
     * @param name Product name
     * @param price Price in IDR
     * @param category Category (pastry/sandwich/other)
     * @return Unique pointer to new Snack
     */
    std::unique_ptr<Product> createSnack(const std::string& name, double
price,
                                         const std::string& category =
"other");

    // ============ File I/O ============

    /**
     * @brief Saves all products to file
     * @return true if successful
     */
    bool saveToFile();

    /**
```

```
    * @brief Loads all products from file
    * @return true if successful
    */
   bool loadFromFile();

   // ============ Query ============

   /**
    * @brief Gets total product count
    * @return Number of products
    */
   int getCount() const;

   /**
    * @brief Gets count by product type
    * @param type "coffee" or "snack"
    * @return Number of products of that type
    */
   int getCountByType(const std::string& type) const;
};

#endif // PRODUCTCONTROLLER_H
```

- Source code ProductController.cpp

```
/**
* @file ProductController.cpp
* @brief Implementation of ProductController class
*/
```

```cpp
#include "ProductController.h"
#include "../models/Coffee.h"
#include "../models/Snack.h"
#include "../utils/IdGenerator.h"
#include <algorithm>

ProductController::ProductController(FileManager& fileManager)
    : m_fileManager(fileManager)
{
}

std::vector<Product*> ProductController::getAll() const {
    std::vector<Product*> result;
    result.reserve(m_products.size());

    for (const auto& product : m_products) {
        result.push_back(product.get());
    }

    return result;
}

Product* ProductController::getById(int id) {
    auto it = std::find_if(m_products.begin(), m_products.end(),
        [id](const std::unique_ptr<Product>& p) {
            return p->getId() == id;
        });
```

```
    if (it != m_products.end()) {
        return it->get();
    }
    return nullptr;
}

std::vector<Product*> ProductController::getByType(const std::string&
type) const {
    std::vector<Product*> result;

    for (const auto& product : m_products) {
        if (product->getType() == type) {
            result.push_back(product.get());
        }
    }

    return result;
}

bool ProductController::add(std::unique_ptr<Product> product) {
    if (!product || !product->isValid()) {
        return false;
    }

    // Update ID generator with this ID
    IdGenerator::getInstance().updateCounter("product",
product->getId());
```

```
    m_products.push_back(std::move(product));
    return true;
}

bool ProductController::update(const Product& product) {
    Product* existing = getById(product.getId());
    if (!existing) {
        return false;
    }

    existing->setName(product.getName());
    existing->setPrice(product.getPrice());
    existing->setExtraField(product.getExtraField());

    return true;
}

bool ProductController::remove(int id) {
    auto it = std::find_if(m_products.begin(), m_products.end(),
        [id](const std::unique_ptr<Product>& p) {
            return p->getId() == id;
        });

    if (it != m_products.end()) {
        m_products.erase(it);
        return true;
    }
    return false;
```

```
}


std::unique_ptr<Product> ProductController::createCoffee(const
std::string& name, double price,
                                                          const
std::string& shotSize) {
    int id = IdGenerator::getInstance().getNextId("product");
    return std::make_unique<Coffee>(id, name, price, shotSize);
}


std::unique_ptr<Product> ProductController::createSnack(const
std::string& name, double price,
                                                        const
std::string& category) {
    int id = IdGenerator::getInstance().getNextId("product");
    return std::make_unique<Snack>(id, name, price, category);
}


bool ProductController::saveToFile() {
    std::vector<std::string> lines;
    lines.reserve(m_products.size());


    for (const auto& product : m_products) {
        lines.push_back(product->serialize());
    }


    return m_fileManager.writeLines(FILENAME, lines);
}
```

```cpp
bool ProductController::loadFromFile() {
    m_products.clear();

    std::vector<std::string> lines = m_fileManager.readLines(FILENAME);

    for (const auto& line : lines) {
        if (line.empty()) continue;

        // Parse type from the line to determine product type
        std::vector<std::string> fields = FileManager::splitLine(line,
'|');
        if (fields.size() < 4) continue;

        std::string type = fields[3];
        std::unique_ptr<Product> product;

        if (type == "coffee") {
            product = std::make_unique<Coffee>();
        } else if (type == "snack") {
            product = std::make_unique<Snack>();
        } else {
            continue;  // Unknown type
        }

        product->deserialize(line);

        if (product->isValid()) {
            // Update ID generator
```

```
            IdGenerator::getInstance().updateCounter("product",
product->getId());
            m_products.push_back(std::move(product));
        }
    }

    return true;
}

int ProductController::getCount() const {
    return static_cast<int>(m_products.size());
}

int ProductController::getCountByType(const std::string& type) const {
    return static_cast<int>(std::count_if(m_products.begin(),
m_products.end(),
        [&type](const std::unique_ptr<Product>& p) {
            return p->getType() == type;
        }));
}
```

- Source code TransactionController.h

```
/**
* @file TransactionController.h
* @brief Controller for managing transactions
*
* SOLID Principles Applied:
* - Single Responsibility: Only manages transaction operations
```

```
* - Dependency Inversion: Depends on abstractions (FileManager, other
controllers)
*/

#ifndef TRANSACTIONCONTROLLER_H
#define TRANSACTIONCONTROLLER_H

#include "IController.h"
#include "../models/Transaction.h"
#include "../utils/FileManager.h"
#include "ProductController.h"
#include "CustomerController.h"
#include <vector>

/**
* @class TransactionController
* @brief Manages transaction operations and coordinates with other
controllers
*
* This controller handles the complete transaction flow, including
* cart management, points calculation, and finalization.
*/
class TransactionController : public IController<Transaction> {
private:
    std::vector<Transaction> m_transactions;     ///< Completed
transactions
    Transaction m_currentTransaction;            ///< Active
cart/transaction
    FileManager& m_fileManager;                  ///< File I/O handler
```

```
    ProductController& m_productController;      ///< Product operations
    CustomerController& m_customerController;    ///< Customer operations
    static constexpr const char* FILENAME = "transactions.txt";

public:
    /**
     * @brief Constructs controller with dependencies
     * @param fileManager Reference to FileManager
     * @param productController Reference to ProductController
     * @param customerController Reference to CustomerController
     */
    TransactionController(FileManager& fileManager,
                          ProductController& productController,
                          CustomerController& customerController);

    /**
     * @brief Destructor
     */
    ~TransactionController() override = default;

    // ============ IController Interface Implementation ============

    std::vector<Transaction> getAll() const override;
    Transaction* getById(int id) override;
    bool add(const Transaction& transaction) override;
    bool update(const Transaction& transaction) override;
    bool remove(int id) override;
    bool saveToFile() override;
```

```
    bool loadFromFile() override;
    int getCount() const override;

    // ============ Current Transaction (Cart) Operations ============

    /**
     * @brief Starts a new transaction
     * @param customerId Customer ID (0 for guest)
     */
    void startNewTransaction(int customerId = 0);

    /**
     * @brief Gets the current active transaction
     * @return Reference to current transaction
     */
    Transaction& getCurrentTransaction();

    /**
     * @brief Gets the current active transaction (const)
     * @return Const reference to current transaction
     */
    const Transaction& getCurrentTransaction() const;

    /**
     * @brief Adds a product to the current transaction
     * @param productId Product ID to add
     * @param quantity Quantity (default: 1)
     * @return true if product found and added
```

```
     */
    bool addToCart(int productId, int quantity = 1);

    /**
     * @brief Removes a product from the current transaction
     * @param productId Product ID to remove
     * @return true if removed
     */
    bool removeFromCart(int productId);

    /**
     * @brief Updates quantity of a product in cart
     * @param productId Product ID
     * @param quantity New quantity
     * @return true if updated
     */
    bool updateCartQuantity(int productId, int quantity);

    /**
     * @brief Clears the current cart
     */
    void clearCart();

    /**
     * @brief Sets the customer for current transaction
     * @param customerId Customer ID
     */
    void setCurrentCustomer(int customerId);
```

```
/**
 * @brief Sets points to use for current transaction
 * @param points Points to redeem
 * @return true if customer has enough points
 */
bool setPointsToUse(int points);

/**
 * @brief Completes the current transaction
 * @return true if transaction was completed successfully
 *
 * This will:
 * 1. Finalize the transaction
 * 2. Deduct loyalty points if used
 * 3. Add earned loyalty points to customer
 * 4. Add transaction to history
 * 5. Clear the cart
 */
bool completeTransaction();

/**
 * @brief Cancels the current transaction
 */
void cancelTransaction();

/**
 * @brief Checks if cart has items
```

```
     * @return true if cart is not empty
     */
    bool hasItemsInCart() const;

    // ============ Query Methods ============

    /**
     * @brief Gets transactions for a specific customer
     * @param customerId Customer ID
     * @return Vector of transactions for that customer
     */
    std::vector<Transaction*> getByCustomerId(int customerId);

    /**
     * @brief Gets transactions for a date range
     * @param startDate Start date (YYYY-MM-DD)
     * @param endDate End date (YYYY-MM-DD)
     * @return Vector of transactions in range
     */
    std::vector<Transaction*> getByDateRange(const std::string&
startDate,
                                             const std::string&
endDate);

    /**
     * @brief Calculates total revenue from all transactions
     * @return Total revenue in IDR
     */
    double getTotalRevenue() const;
```

```
    /**
     * @brief Gets recent transactions
     * @param count Number of recent transactions to get
     * @return Vector of recent transactions
     */
    std::vector<Transaction*> getRecent(int count);
};

#endif // TRANSACTIONCONTROLLER_H
```

- Source code TransactionController.cpp

```
/**
* @file TransactionController.cpp
* @brief Implementation of TransactionController class
*/

#include "TransactionController.h"
#include "../utils/IdGenerator.h"
#include <algorithm>

TransactionController::TransactionController(FileManager& fileManager,
                                             ProductController&
productController,
                                             CustomerController&
customerController)
    : m_fileManager(fileManager)
    , m_productController(productController)
    , m_customerController(customerController)
```

```
{
}


std::vector<Transaction> TransactionController::getAll() const {
    return m_transactions;
}


Transaction* TransactionController::getById(int id) {
    auto it = std::find_if(m_transactions.begin(), m_transactions.end(),
        [id](const Transaction& t) {
            return t.getId() == id;
        });

    if (it != m_transactions.end()) {
        return &(*it);
    }
    return nullptr;
}


bool TransactionController::add(const Transaction& transaction) {
    if (!transaction.isValid()) {
        return false;
    }

    IdGenerator::getInstance().updateCounter("transaction",
transaction.getId());
    m_transactions.push_back(transaction);
    return true;
```

```
}

bool TransactionController::update(const Transaction& transaction) {
    Transaction* existing = getById(transaction.getId());
    if (!existing) {
        return false;
    }

    *existing = transaction;
    return true;
}

bool TransactionController::remove(int id) {
    auto it = std::find_if(m_transactions.begin(), m_transactions.end(),
        [id](const Transaction& t) {
            return t.getId() == id;
        });

    if (it != m_transactions.end()) {
        m_transactions.erase(it);
        return true;
    }
    return false;
}

bool TransactionController::saveToFile() {
    std::vector<std::string> lines;
    lines.reserve(m_transactions.size());
```

```
    for (const auto& transaction : m_transactions) {
        lines.push_back(transaction.serialize());
    }

    return m_fileManager.writeLines(FILENAME, lines);
}

bool TransactionController::loadFromFile() {
    m_transactions.clear();

    std::vector<std::string> lines = m_fileManager.readLines(FILENAME);

    for (const auto& line : lines) {
        if (line.empty()) continue;

        Transaction transaction;
        transaction.deserialize(line);

        if (transaction.isValid()) {
            // Populate product names and prices for items
            for (auto& item :
const_cast<std::vector<TransactionItem>&>(transaction.getItems())) {
                Product* product =
m_productController.getById(item.getProductId());
                if (product) {
                    item.setProductName(product->getName());
                    item.setUnitPrice(product->getPrice());
                }
```

```
            }

            IdGenerator::getInstance().updateCounter("transaction",
transaction.getId());
            m_transactions.push_back(transaction);
        }
    }

    return true;
}

int TransactionController::getCount() const {
    return static_cast<int>(m_transactions.size());
}

void TransactionController::startNewTransaction(int customerId) {
    m_currentTransaction = Transaction();
    m_currentTransaction.setCustomerId(customerId);
    m_currentTransaction.setCurrentDateTime();
}

Transaction& TransactionController::getCurrentTransaction() {
    return m_currentTransaction;
}

const Transaction& TransactionController::getCurrentTransaction() const
{
    return m_currentTransaction;
}
```

```
bool TransactionController::addToCart(int productId, int quantity) {
    Product* product = m_productController.getById(productId);
    if (!product || quantity ⩽ 0) {
        return false;
    }

    TransactionItem item(productId, product→getName(),
product→getPrice(), quantity);
    m_currentTransaction.addItem(item);
    return true;
}

bool TransactionController::removeFromCart(int productId) {
    return m_currentTransaction.removeItem(productId);
}

bool TransactionController::updateCartQuantity(int productId, int
quantity) {
    return m_currentTransaction.updateItemQuantity(productId, quantity);
}

void TransactionController::clearCart() {
    m_currentTransaction.clearItems();
}

void TransactionController::setCurrentCustomer(int customerId) {
    m_currentTransaction.setCustomerId(customerId);
    // Reset points usage when customer changes
```

```
    m_currentTransaction.setPointsUsed(0);
}

bool TransactionController::setPointsToUse(int points) {
    int customerId = m_currentTransaction.getCustomerId();
    if (customerId <= 0) {
        return false;
    }

    Customer* customer = m_customerController.getById(customerId);
    if (!customer) {
        return false;
    }

    // Check if customer has enough points
    if (points > 0 && !customer->canRedeemPoints(points)) {
        return false;
    }

    m_currentTransaction.setPointsUsed(points);
    return true;
}

bool TransactionController::completeTransaction() {
    if (!m_currentTransaction.hasItems()) {
        return false;
    }
```

```
    // Generate ID
    int id = IdGenerator::getInstance().getNextId("transaction");
    m_currentTransaction.setId(id);
    m_currentTransaction.setCurrentDateTime();
    m_currentTransaction.recalculate();

    int customerId = m_currentTransaction.getCustomerId();
    int pointsUsed = m_currentTransaction.getPointsUsed();
    int pointsEarned = m_currentTransaction.getPointsEarned();

    // Handle loyalty points
    if (customerId > 0) {
        // Deduct used points
        if (pointsUsed > 0) {
            if (!m_customerController.redeemLoyaltyPoints(customerId,
pointsUsed)) {
                return false;  // Failed to redeem points
            }
        }

        // Add earned points
        if (pointsEarned > 0) {
            m_customerController.addLoyaltyPoints(customerId,
pointsEarned);
        }
    }

    // Add to history
    m_transactions.push_back(m_currentTransaction);
```

```
    // Clear cart for next transaction
    startNewTransaction(0);

    return true;
}

void TransactionController::cancelTransaction() {
    m_currentTransaction = Transaction();
}

bool TransactionController::hasItemsInCart() const {
    return m_currentTransaction.hasItems();
}

std::vector<Transaction*> TransactionController::getByCustomerId(int
customerId) {
    std::vector<Transaction*> result;

    for (auto& transaction : m_transactions) {
        if (transaction.getCustomerId() == customerId) {
            result.push_back(&transaction);
        }
    }

    return result;
}
```

```cpp
std::vector<Transaction*> TransactionController::getByDateRange(const
std::string& startDate,
                                                                 const
std::string& endDate) {
    std::vector<Transaction*> result;

    for (auto& transaction : m_transactions) {
        std::string date = transaction.getDateTime().substr(0, 10);  //
YYYY-MM-DD
        if (date ⩾ startDate && date ⩽ endDate) {
            result.push_back(&transaction);
        }
    }

    return result;
}

double TransactionController::getTotalRevenue() const {
    double total = 0.0;
    for (const auto& transaction : m_transactions) {
        total += transaction.getTotal();
    }
    return total;
}

std::vector<Transaction*> TransactionController::getRecent(int count) {
    std::vector<Transaction*> result;
```

```
   int start = std::max(0, static_cast<int>(m_transactions.size()) -
count);


   for (int i = static_cast<int>(m_transactions.size()) - 1; i ⩾ start;
--i) {
       result.push_back(&m_transactions[i]);
   }


   return result;
}
```

- Repositori: github.com/mfachrizalg/skena