S/W Detailed Level Design

Project Name	Photo Studio Management System		
Block Name	Release 2		
Author	Team	**Approver**	Anna Kyselova
Team	Cebanu Dan (@Nik0gda) Nikita Springis (@nikitaspringis21) Manana Bebia (@Mabebi29) Anna Sukhanishvili (@annasulkh)		

Contents

■ Revision History

Version	Date	Revised contents	Author	Approver
1.0	23.09.2025	Release 1	Team	Anna Kyselova
1.1	26.10.2025	Release 2	Team	Anna Kyselova

■ Terms and Abbreviations

Term	Description
DIP	Dependency Inversion Principle
RAII	Resource Acquisition Is Initialization
UML	Unified Modeling Language
SRP	Single Responsibility Principle
OCP	Open/Closed Principle

■ References

1. SW Requirements Specification

1. Overview

This document provides the detailed technical design for the Photo Studio Management System, a C++ application that automates the workflow of a photo studio. The system manages client orders for photo printing and film developing services, tracks employee responsibilities across three roles (Receptionist, Photographer, Administrator), manages consumable inventory, and generates comprehensive reports.

The system supports both regular and express orders (with 25% surcharge), tracks consumable usage during photo processing, and provides end-of-day reporting capabilities. The design follows SOLID principles with clear separation of concerns, dependency injection, and polymorphic behavior to ensure maintainability and extensibility.

Key stakeholders include studio receptionists who create orders, photographers who process orders and track consumables, administrators who manage inventory, and studio management who review reports. The system replaces manual paper-based tracking with an automated console-based application.

2. System Overview / Architectural Context

The system follows a layered architecture with clear separation between presentation, business logic, and data management:

Presentation Layer:

- ConsoleDisplay: Handles all user interface output
- IDisplay interface: Abstracts display functionality for testing and future UI implementations

Business logic layer:

- Manager Classes: OrderManager, ConsumableManager, ReportManager
- Employee Classes: Receptionist, Photographer, Administrator (polymorphic hierarchy)
- Domain Services: Order processing, inventory management, report generation

Domain layer:

- Core Entities: Order/ExpressOrder, Client, Service, Consumable
- Value Objects: OrderItem, ConsumableUsage, Report
- Enumerations: OrderStatus, ServiceType, ReportType

Infrastructure layer:

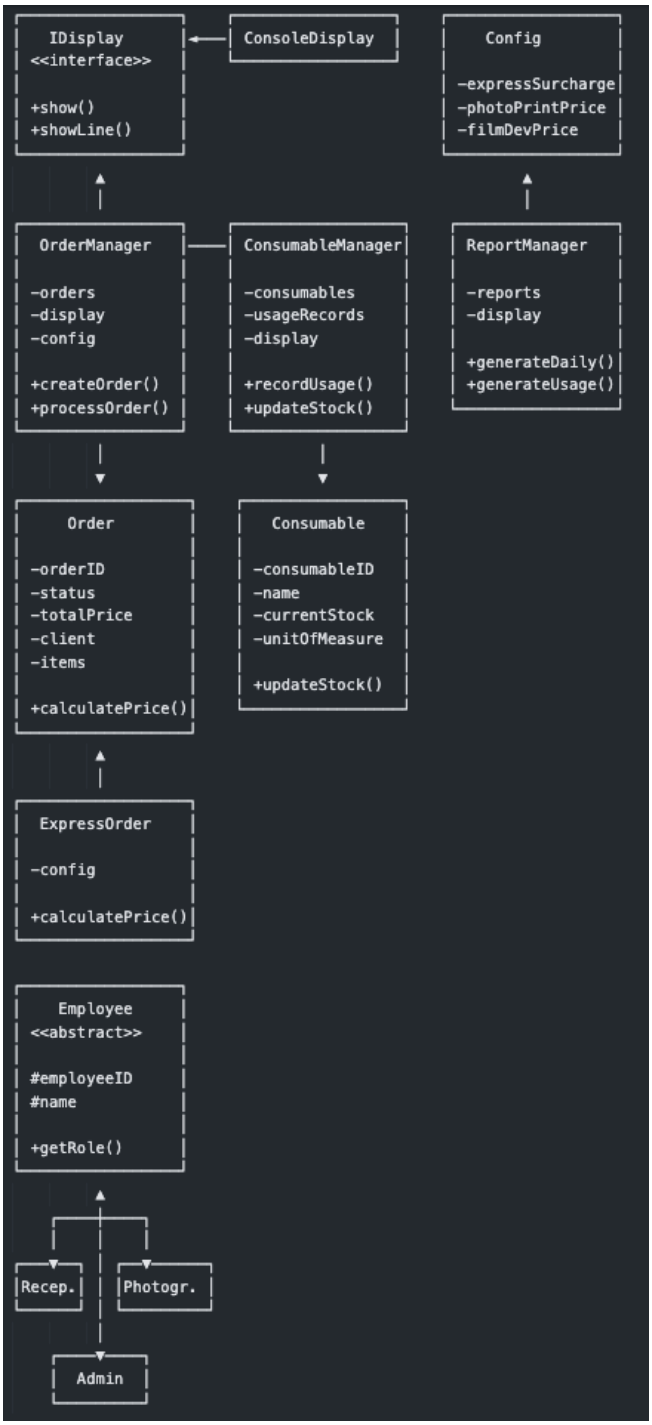
- Configuration: Config class for business rules and pricing
- Types: Centralized type definitions and enumerations

Dependency flow:

Presentation -> Business Logic -> Domain -> Infrastructure

Each layer interacts only with the layer directly below it, ensuring loose coupling and high cohesion. The system uses dependency injection throughout, with the IDisplay interface enabling easy testing and future UI changes.

3. UML Class Diagram (Technical Design)



4. Class Specifications

Class	Type	Description	Attributes	Methods
Order	Entity	Represents a client order for photo services	orderID (string), completionTime (string), status(OrderStatus), totalPrice(double), isPaid(bool), client(Client*), items(vector<OrderItem*>)	calculatePrice(), updateStatus(), recordPayment(), addItem(), getters
ExpressOrder	Entity	Express order with surcharge	config(Config*)	calculatePrice() override
Client	Entity	Represents a studio client	clientID(string), surname(string)	getSurname(), getID()
Employee	Abstract	Base class for all employees	employeeID(string), name(string)	getName(), getID(), getRole()
Receptionist	Employee	Handles order creation and reports	Inherited	createOrder(), generateDailyRevenueReport() getRole()
Photographer	Employee	Processes orders and tracks consumables	Inherited	viewAssignedOrders(), submitConsumablesReport(), getRole()
Administrator	Employee	Manages inventory and reviews reports	Inherited	manageConsumablesStock(), reviewConsumablesReports(), getRole()
OrderManager	Manager	Manages order lifecycle	orders(vector<Order*>), display(IDisplay*), config(Config*)	createOrder(), addItemToOrder(), processOrder(), completeOrder(), calculateTotalRevenue()
Consumable Manager	Manager	Tracks inventory and usage	consumables(vector<Consumable*>),	addConsumable(), recordUsage(),

			usageRecords(vector<ConsumableUsage>), display(IDisplay*)	updateStock(), findConsumableByName()
ReportManager	Manager	Generates and stores reports	reports(vector<Report*>), display(IDisplay*)	generateDailyRevenueReport(), generateConsumablesUsageRepo getAllReports()
Consumable	Entity	Represents studio consumables	consumableID(string), name(string), currentStock(int), unitOfMeasure(string)	updateStock(), getCurrentStock(), getName(), getConsumableID()
Service	Entity	Represents available services	serviceID(string), name(string), basePrice(double), type(ServiceType)	getBasePrice(), getName(), getServiceID(), getType()
OrderItem	Value Obj	Individual items in an order	itemID(string), quantity(int), unitPrice(double)	getSubtotal(), getItemID(), getQuantity(), getUnitPrice()
Config	Config	Business rules and pricing	expressSurchargeRate(double), photoPrintingBasePrice(double), filmDevelopingBasePrice(double)	Getters and setters for all rates ar prices

5. Interfaces and Abstractions

Interface	Purpose	Key Methods	Planned For (Release)
IDisplay	Abstract output interface for UI flexibility	show(message), showLine(message)	Release 2
Employee	Abstract base for polymorphic employee behaviour	getRole()	Release 2
Order	Virtual base for polymorphic pricing	calculatePrice()	Release 1

6. Function Responsibilities

Class	Method	Purpose	Input	Output	Notes
OrderManager	createOrder()	Factory method for order creation	orderID, client, completionTime, isExpress	Order*	Creates regular or express order based on flag
OrderManager	calculateTotalRevenue()	Sums revenue from all paid orders	None	double	Iterates through orders, sums paid totals
ExpressOrder	calculatePrice()	Applies 25% surcharge to base price	None	double	Overrides base implementation
ConsumableManager	recordUsage()	Updates inventory when consumables used	ConsumableUsage	void	Automatically reduces stock
ReportManager	generateDailyRevenueReport()	Creates revenue summary	OrderManager*	void	Aggregates order data
Employee	getRole()	Returns employee type	None	string	Pure virtual, implemented by subclasses

7. Operation Flow

Primary order processing flow:

1. Client Interaction: Client approaches receptionist with service request
2. Order Creation: Receptionist -> OrderManager.createOrder() -> Order/ExpressOrder created
3. Item Addition: OrderManager.addItemToOrder() -> OrderItem created and added
4. Price Calculation: Order.calculatePrice() -> Base price or surcharge applied
5. Order Processing: OrderManager.processOrder() -> Status updated to IN_PROGRESS
6. Photographer Assignment: Photographer.viewAssignedOrders() -> Order retrieved
7. Consumable Usage: Photographer.submitConsumablesReport() -> ConsumableManager.recordUsage()
8. Inventory Update: ConsumableManager updates stock automatically
9. Order Completion: OrderManager.completeOrder() -> Status updated to COMPLETED
10. Payment Processing: OrderManager.recordPayment() -> Payment recorded

End-of-day reporting flow:

1. Revenue Report: Receptionist -> ReportManager.generateDailyRevenueReport()
2. Usage Report: Administrator -> ReportManager.generateConsumablesUsageReport()
3. Report Storage: Reports stored in ReportManager for history

Data flow:

ConsoleDisplay <-> Manager Classes <-> Domain Entities <-> Configuration

8. Enumerations & Constants

Name	Value / Type	Description
OrderStatus	Enum class	PENDING, IN_PROGRESS, COMPLETED, CANCELLED
ServiceType	Enum class	PHOTO_PRINTING, FILM_DEVELOPING
ReportType	Enum class	DAILY_REVENUE, CONSUMABLES_USAGE
EXPRESS_SURCHARGE_RATE	0.25 (double)	25% surcharge for express orders
PHOTO_PRINTING_BASE_PRICE	15.99 (double)	Base price for photo printing service
FILM_DEVELOPING_BASE_PRICE	25.50 (double)	Base price for film developing service

9. Validation Rules & Future Work

Rule / Planned Feature	Description	Target Release
Order Validation	Validate orderID uniqueness, client existence	Release 2
Stock Validation	Prevent negative stock levels, low stock warning	Release 2
Payment Validation	Ensure order completed before payment recorded	Release 2

GUI Interface	Replace console with graphical interface	Release 3
Database integration	Replace in-memory storage with persistent database	Release 3
Multi-photographer Support	Support multiple photographers and order assignment	Release 3
Customer Management	Extended client profiles with contact information	Release 3
Advanced Reporting	Date range reports, profit analysis, trend tracking	Release 3
Email notifications	Automated client notifications for order status	Release 4

10. Traceability Matrix

Requirement (SRS)	Class / Method (DLD)
"Client places order with receptionist"	Receptionist::createOrder(), OrderManager::createOrder()
"Record client surname and completion time"	Client class, Order constructor
"Express orders have 25% surcharge"	ExpressOrder::calculatePrice(), Config::getExpressSurchargeRate()
"Payment made upon completion"	Order::recordPayment(), OrderManager::recordPayment()
"Photographer uses consumables"	Photographer::submitConsumablesReport() ConsumableManager::recordUsage()
"Administrator accounts for consumables"	Administrator::manageConsumablesStock(), ConsumableManager

Requirement (SRS)	Class / Method (DLD)
“Receptionist reports on revenue”	Receptionist::generateDailyRevenueReport(), ReportManager::generateDailyRevenueReport()
“Photographer reports consumed materials”	Photographer::submitConsumablesReport(), ReportManager::generateConsumablesUsageReport()
“Two forms: client and photographer”	Order creation workflow, Order status tracking

11. Code Structure and File Mapping

Class	File
Order	src/orders/Order.cpp / src/orders/Order.h
ExpressOrder	src/orders/ExpressOrder.cpp / src/orders/ExpressOrder.h
Client	src/entities/Client.cpp / src/entities/Client.h
Employee	src/employees/Employee.cpp / src/employees/Employee.h
Receptionist	src/employees/Receptionist.cpp / src/employees/Receptionist.h
Photographer	src/employees/Photographer.cpp / src/employees/Photographer.h
Administrator	src/employees/Administrator.cpp / src/employees/Administrator.h
OrderManager	src/managers/OrderManager.cpp / src/managers/OrderManager.h
ConsumableManager	src/managers/ConsumableManager.cpp / src/managers/ConsumableManager.h
ReportManager	src/managers/ReportManager.cpp / src/managers/ReportManager.h
Consumable	src/entities/Consumable.cpp / src/entities/Consumable.h
Service	src/entities/Service.cpp / src/entities/Service.h
OrderItem	src/entities/OrderItem.cpp / src/entities/OrderItem.h

Class	File
ConsumableUsage	src/entities/ConsumableUsage.cpp / src/entities/ConsumableUsage.h
Report	src/entities/Report.cpp / src/entities/Report.h
Config	src/config/Config.cpp / src/config/Config.h
IDisplay	src/interfaces/IDisplay.h
ConsoleDisplay	src/implementations/ConsoleDisplay.cpp / src/implementations/ConsoleDisplay.h
Types	src/types/Types.h
Main Application	src/main.cpp

12. Revision History

Date	Version	Change Summary	Author
26.10.2025	1.0	Initial DLD creation	Team