Islamic University-Gaza Faculty of Information Technology Dr. Abdelkareem Alashqar	بسم الله الرحمن الرحيم 	SW Engineering Assignment 6

Main Document

For

Point of Sale (POS) Management System

Supervised by:

Abdelkareem Alashqar

Submitted in partial fulfillment of the requirements for

Software Engineering Course _CSCI2313

Web and Multimedia Engineering Course_WDMM3314

By

Yazan. Kh .H .Owda 120220544

January 2026

Links:

GitHub Repository Link:

yazanowda102004-debug/POS_Managment-System

Gogal drive Link:

https://drive.google.com/drive/folders/1QIOEkdlkeGD7FlzTpqlZ2iP2iPiV64zt?usp=drive_link

Abstract

This document presents a comprehensive engineering plan, analysis, and design for the **Point of Sale (POS) Management System**. The system is designed to automate daily retail operations, including **fast sales processing**, **real-time inventory tracking**, and **automated financial reporting**. To ensure the steady delivery of core functionalities and effective risk management, the project follows an **Incremental Development Model**.

The planning phase details the project schedule using **Network Diagrams**, identifying a **31-week minimum duration** and establishing the **Critical Path** (Activities A, D, E, F, I, J) to ensure timely delivery. The system's architecture is modeled using **UML diagrams**, including Use Case, Class, and Sequence diagrams, to define interactions between stakeholders such as Cashiers, Managers, and System Administrators.

Finally, the document details the technical implementation of core functions—such as **barcode scanning** and **stock management**—using **[PHP]** following the **Facade design pattern**. This is supported by a rigorous testing report covering unit and system testing to guarantee that the final product meets all operational and security requirements, including data encryption and high system reliability.

Table of Contents

1. Chapter 1: Project Plan

1.1 Project Overview

The Point of Sale (POS) Management System is an integrated software solution developed to automate and optimize the daily transactional and inventory workflows within retail environments. Its primary goal is to eliminate reliance on manual record-keeping and inefficient sales processing by digitizing critical store operations. The system features an intelligent inventory tracking module that provides real-time updates on stock levels, product categories, and low-stock alerts, a high-speed sales processing component for handling customer transactions through barcode scanning, and a comprehensive reporting module for real-time financial and revenue analysis. By centralizing these functions, the POS System enhances transactional accuracy, increases operational productivity, and ensures a seamless experience for both cashiers and customers.

1.2 Problem Statement

The current reliance on traditional methods and manual recording in the retail store results in significant operational inefficiencies and high administrative costs. The following are the key problems justifying the need for the **Point of Sale (POS) Management System**:

- **Excessive Reliance on Manual Sales Processing:** The use of manual logs for recording sales transactions leads to long customer wait times and increased risk of calculation errors. This manual entry carries a high risk of data loss and transcription errors during busy hours.
- **Inefficient Inventory Tracking:** Manual tracking of stock levels often results in inventory conflicts, such as selling out-of-stock products or failing to identify low-stock items in time. This lack of real-time updates reduces the productivity of sales processing and leads to customer dissatisfaction due to stock shortages.
- **Complicated Financial and Revenue Tracking:** Manual handling of sales receipts, daily revenue, and cash flow is prone to human error and consumes excessive administrative time. Without automation, tracking specific transaction histories or payment statuses becomes difficult and unreliable.
- **Lack of Real-Time Operational Insights:** The current manual system does not allow for effective summarization of sales trends or top-selling products. This prevents management from obtaining accurate and timely information for making critical decisions regarding restocking, pricing, and promotional offers.

1.3 Project Objectives

The main objective of the **POS Management System** project is to develop a software application that automates and supports all core retail transactions and inventory management tasks within the store. This is achieved by:

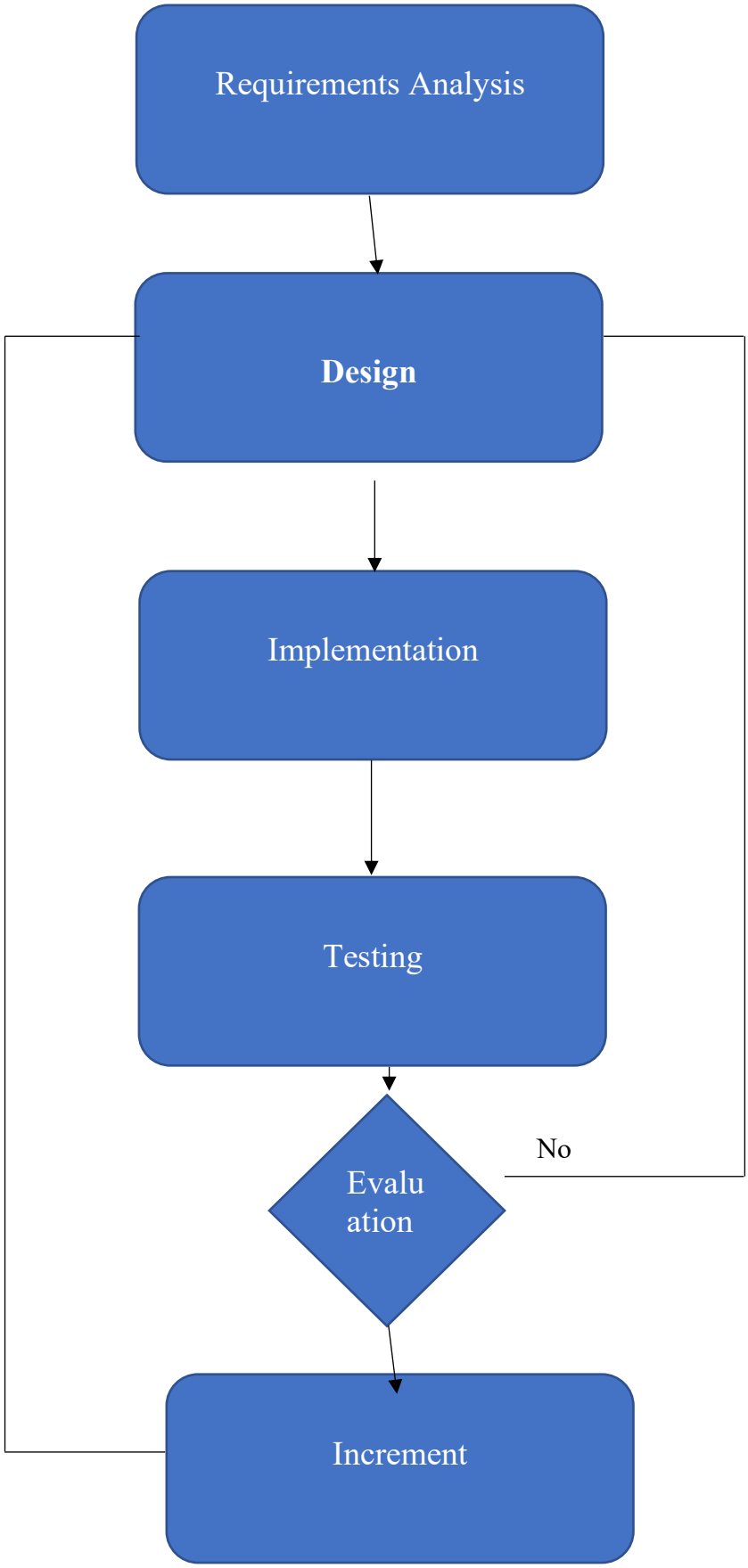
- **Automate and Optimize Sales Workflow:** To implement an efficient and integrated sales system that supports fast transaction processing through barcode scanning and automated price calculation, thereby increasing cashier productivity and reducing customer wait times.
- **Digitize Inventory and Product Records:** To establish a secure and reliable digital database for product details, categories, and stock levels, thus eliminating reliance on manual ledger entries and paper-based tracking.
- **Improve Data Accuracy and Real-Time Reporting:** To minimize manual data entry errors during sales and stocktakes, and provide accurate, real-time reports that assist management in making timely decisions regarding inventory restocking and sales trends.
- **Increase Operational Efficiency and Reduce Costs:** To achieve better operational efficiency and reduce administrative expenses by minimizing manual record-keeping labor hours and eliminating errors associated with traditional paper-based methods.

1.4 Software Process Model

The **Incremental Process Model** will be used for developing the **Point of Sale (POS) Management System**. This model allows the system to be built and delivered in small releases, or increments, with each release adding new functionality to the system.

Justification for Choosing the Incremental Model:

- **Early Deliverables:** A usable subset of the system (such as the **Product Inventory and Lookup module**) can be delivered early in the project lifecycle. This provides initial value to the retail store and allows the cashiers and staff to familiarize themselves with the digital scanning and inventory system before the full sales and reporting modules are complete.
- **Risk Mitigation:** It reduces the risk of requirements volatility by gathering continuous feedback from the store management and staff after each increment. High-priority features, such as **Barcode Sales Processing and Real-Time Stock Updates**, are delivered and tested first, ensuring that the core business operations are supported as early as possible.


Requirements Analysis
Design
Implementation
Testing
Evalu
ation
No
Increment

1.5 Team Organization

The development of the **POS Management System** is managed by a structured team where each member has defined roles and responsibilities to ensure the project's success. The team organization is divided as follows:

- **Project Manager / Systems Analyst:** Responsible for overseeing the project schedule, managing risks, and gathering requirements from the store owners. They ensure that the system design aligns with the business goals of the retail shop.
- **Backend Developers:** Focused on implementing the core logic of the system using **PHP**. Their main tasks include developing the sales processing engine, inventory database connections, and applying the **Facade design pattern** for system integration.
- **Frontend Developers:** Responsible for creating a user-friendly and responsive interface for cashiers and managers, ensuring that barcode scanning and product lookups are intuitive and fast.
- **Quality Assurance (QA) / Testers:** Tasked with designing and executing test cases. They perform unit testing for individual functions and system testing to ensure the hardware (scanners/printers) works perfectly with the software.

1.6 Risk Analysis

A risk assessment was performed to identify and analyze potential adverse circumstances that may affect the POS project. Risks are ranked based on the combination of their Probability and Impact (Low, Mid, or High):

#	Risk Description	Probability	Impact	Rank	Risk Plan
1	**(Requirements Risk)** Changes to store requirements that require major design rework.	Mid	High	High	Derive traceability information to assess change impact; maximize information hiding in the design.
2	**(Technology Risk)** Hardware integration issues (e.g., barcode scanners or printers	Mid	High	High	Conduct early hardware compatibility testing and use standardized APIs to ensure

#	Risk Description	Probability	Impact	Rank	Risk Plan
	failing to sync with PHP).				seamless communication.
3	**(Estimation Risk)** The time required to develop the real-time stock update and billing modules is underestimated.	High	Mid	High	Monitor the **Critical Path** (31 weeks) closely and use slack time from non-critical activities as buffers.
4	**(Data Security Risk)** Unauthorized access to financial transaction data or inventory records.	Low	High	Mid	Implement role-based access control (RBAC) and database encryption to ensure data integrity.

1.7 Hardware and Software Resource Requirements

The development and deployment of the **POS Management System** require a specific set of hardware and software resources to ensure high performance and system reliability.

Hardware Resources: The system is designed to run on a central workstation or PC (Intel i5 or higher) that acts as the main terminal. Essential peripherals include **Barcode Scanners** for rapid product identification, **Thermal Receipt Printers** for transaction documentation, and a stable **Local Area Network (LAN)** to connect the terminal to the database and printer infrastructure.

Software Resources: On the software side, the system is built using the **PHP** programming language for backend logic and **MySQL** as the primary database management system for secure data storage. The development environment utilizes **Visual Studio Code** as the main IDE, while a local server like **XAMPP/Apache** is used for hosting and testing the application. The system is compatible with modern operating systems such as **Windows 10/11** and is accessed through standard web browsers like Google Chrome.

1.8 Project Schedule

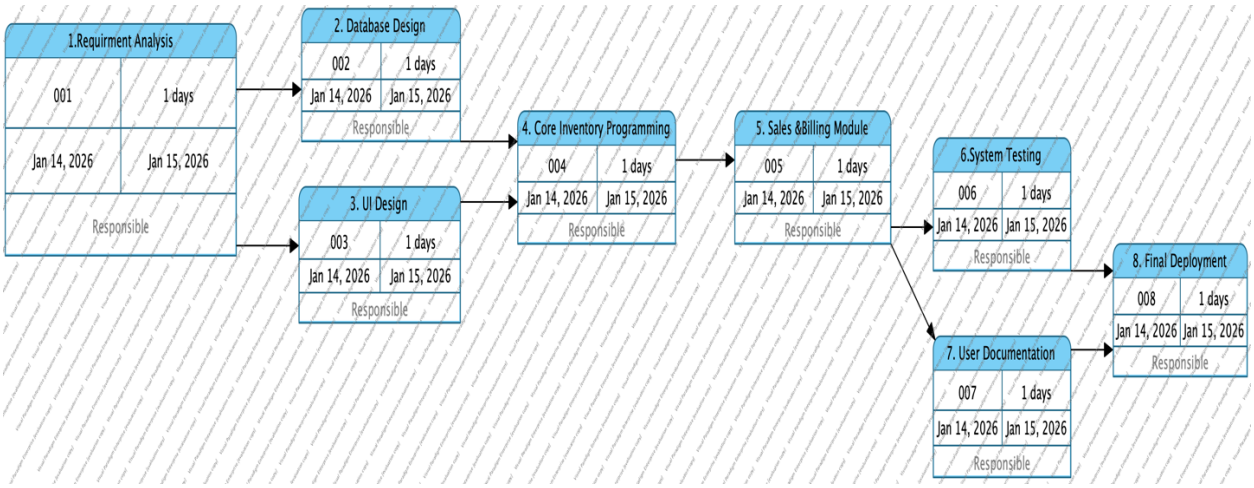
#	Task Name	Expected Time (Weeks)	Preceding Task(s)
1	Requirements Collection & Analysis	4	-
2	Database Design (Products, Users, Sales)	3	1
3	User Interface (UI) Design	4	1
4	Core Modules Programming (Inventory)	6	2, 3
5	Sales Module & Barcode Programming	5	4
6	Testing & System Integration	3	5
7	User Documentation Creation	2	4
8	Training & Deployment	4	6, 7

1.8.2 Gantt Chart

Task Name	Start Week	Duration (Weeks)
Requirements Analysis	1	4
Database & UI Design	5	3
Inventory Module	8	6
Sales & Billing Module	14	4
System Testing	18	3

Task Name	Start Week	Duration (Weeks)
Documentation	21	2
Final Deployment	23	1

1.8.3 Network Diagram



Chapter 2: System Requirements

2.1 System Overall Description

The **Point of Sale (POS) Management System** is a comprehensive, centralized software solution designed to streamline and automate all core administrative, inventory, and financial operations within a retail store. The system aims to eliminate reliance on paper-based workflows by digitizing product records, sales processing, and billing, thereby enhancing data accessibility and operational efficiency.

The system's main functional components include:

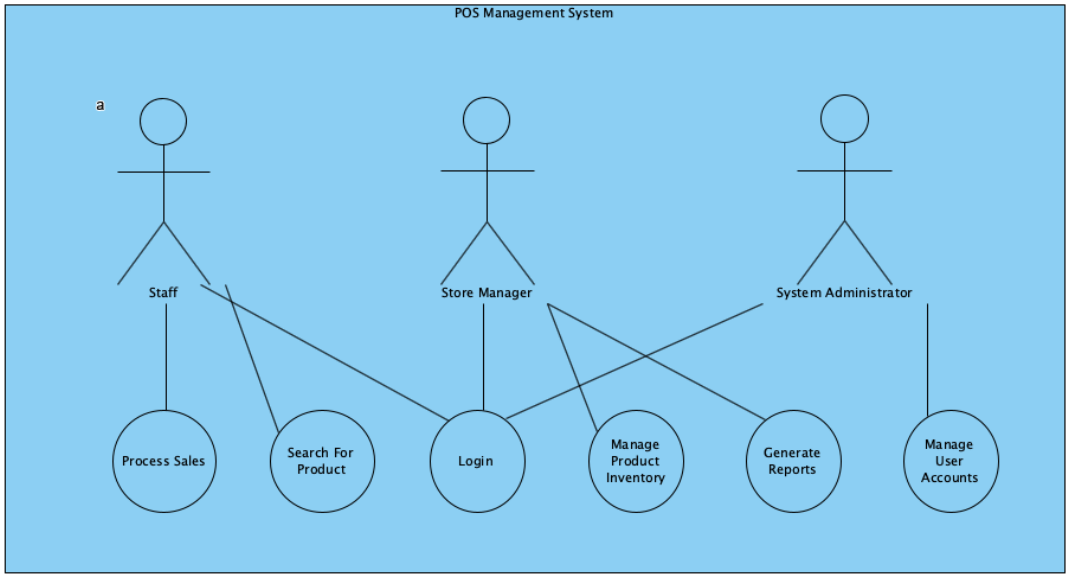
- **Inventory Management:** Allows staff to register new products and manage their bibliographic information, such as name, category, and SKU.
- **Stock Tracking:** Provides a real-time tracking system for managing stock levels, checking item availability, and sending automated low-stock alerts.
- **Sales and Billing:** Enables staff to securely record customer purchases using barcode scanners, process payments, apply discounts, and generate digital invoices.
- **Customer Records:** Handles the storage of customer purchase history, contact information, and loyalty program data.
- **Reporting and Administration:** Allows the administrator to manage user accounts (cashiers and managers) and generate performance and financial reports, such as daily sales trends and total inventory value.

2.2 Main Use Case Diagram

The diagram should include:

- **Actors:** POS Staff (Cashier), Store Manager, System Administrator.

- **Main Use Cases:** Manage Product Inventory, Process Customer Sales, Search for Products, Generate Financial Reports, Manage User Accounts.



2.3 System Specific Requirements

The following sections detail the core functional requirements of the POS Management System using structured scenario tables.

2.3.1 Function Scenario for Sell Books

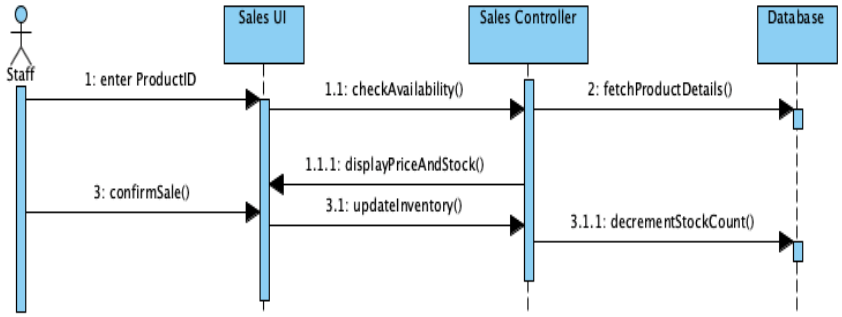
Feature	Description
ID	1
Name	**Process Sales**
Description	Allows the cashier to scan items, calculate totals, and process a payment transaction for a customer.
Priority	**Must have (1)**
Actors	Cashier (Staff)

Feature	Description
Pre-Conditions	1. The product must exist in the system inventory. 2. The cashier must be logged in.
Flow of Events	Details
Normal Flow	1. The cashier selects the "New Sale" function. 2. The system displays the sales transaction screen. 3. The cashier scans the product barcode (or enters ISBN/ID), and the system retrieves the price and stock availability. 4. The cashier adds the desired quantity to the cart. 5. The system calculates the total price, including taxes and discounts. 6. The cashier selects the payment method (Cash/Card). 7. The system confirms the payment and saves the transaction record.

Feature	Description
	8. The system generates and prints a sales receipt for the customer.
Alternative Flow	Alternative flow of events 1: In Step 3, if the product is out of stock, the system displays an "Insufficient Stock" warning and prevents adding it to the cart.
Post-Conditions	1. A "Sale Record" is created in the database. 2. The product's inventory quantity is automatically decreased in the database.

2.3.1.1 Sequence Diagram For [name of function]

d [y2]

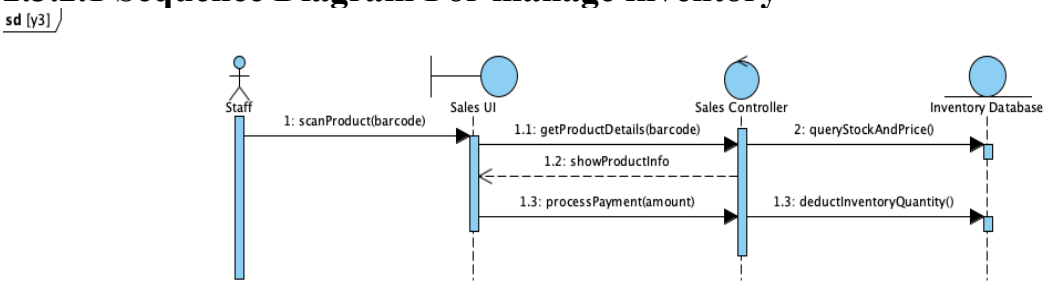


2.3.2 Function Scenario For Manage Inventory

Feature	Description
ID	2
Name	**Manage Inventory (Product Records)**
Description	Allows the Store Manager to add new product titles, update existing stock levels, and categorize products.
Priority	**Must have (1)**
Actors	Store Manager
Pre-Conditions	1. The manager must be logged in. 2. The manager must have administrative permissions.
Flow of Events	Details
Normal Flow	1. The manager selects the "Manage Inventory" function from the dashboard. 2. The system displays the inventory list (Product Name, SKU/Barcode, and Price). 3. The manager selects "Add New Product" or chooses an existing item to update.

Feature	Description
	4. The manager enters/updates product details (Name, Category, Price, and Barcode). 5. The manager enters the quantity received for the stock. 6. The manager clicks "Save & Update". 7. The system validates the data and saves the record to the database. 8. The system displays a confirmation message: "Inventory Updated Successfully".
Alternative Flow	Alternative flow of events 1: In Step 7, if a duplicate Barcode/SKU is detected for a new entry, the system alerts the manager that the product already exists and offers to update its quantity instead of creating a new record.
Post-Conditions	1. A new or updated "Product Record" is stored in the database. 2. The system inventory levels are accurately reflected in the search results.

2.3.2.1 Sequence Diagram For manage inventory

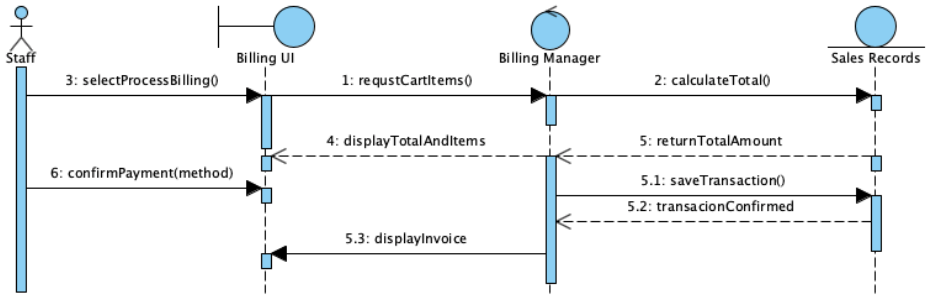


2.3.3 Function Scenario for Process Billing

Feature	Description
ID	3
Name	**Process Billing**
Description	Allows the POS staff to calculate totals, finalize the sale, and generate a payment receipt for the customer.
Priority	**Should Have (2)**
Actors	Staff/Cashier
Pre-Conditions	Customer must have selected items for purchase and items must be in the cart.
Flow of Events	Details
Normal Flow	1. Staff selects "Proceed to Checkout" or "Process Billing". 2. System displays the list of products, individual prices, and the grand total.

Feature	Description
	3. Staff selects payment method (e.g., Cash, Credit Card). 4. Staff inputs the amount received; the system calculates the change if necessary. 5. Staff confirms and generates the invoice. 6. System records the successful payment in the sales log.
Alternative Flow	Alternative flow of events 1: If the payment transaction fails (e.g., card declined), the system displays an error message and allows the staff to try another payment method.
Post-Conditions	Invoice is printed, and payment records are saved in the financial database.

2.3.2.2 Sequence Diagram For process billing

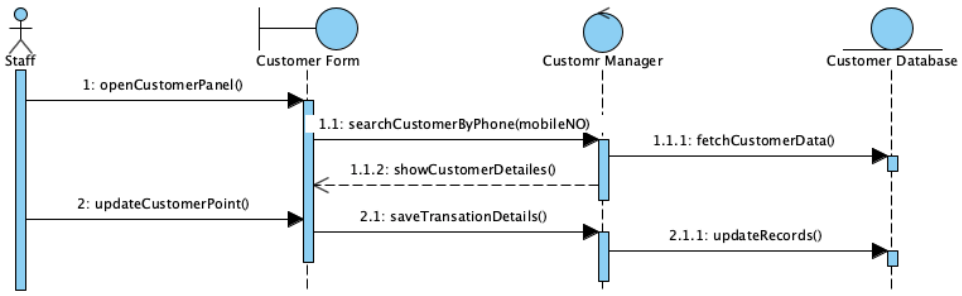


2.3.4 Function Scenario for Manage Customers

Feature	Description
ID	4
Name	**Manage Customers**
Description	Allows POS staff to add, update, and view customer profiles and purchase history.
Priority	**Must have (1)**
Actors	Staff/Cashier
Pre-Conditions	Staff member must be logged into the system.
Flow of Events	Details
Normal Flow	1. Staff selects the "Manage Customers" function from the main menu. 2. The system displays the searchable customer list. 3. Staff adds a new customer profile or updates existing contact information (Name, Phone, Email).

Feature	Description
	4. The system validates and saves the customer data to the database.
Alternative Flow	Alternative flow of events 1: If required data (e.g., Phone Number) is missing or in an incorrect format, the system displays an error message and highlights the required fields.
Post-Conditions	Customer information is stored or updated successfully in the system database.

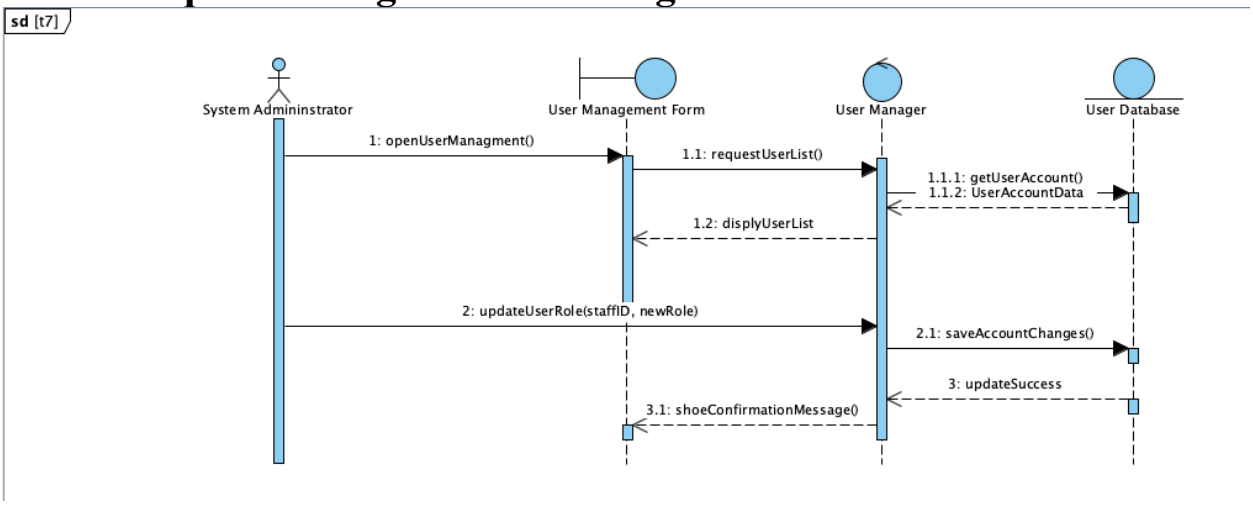
2.3.2.3 Sequence Diagram For Manage Customers



2.3.5 Function Scenario for Manage Users

Feature	Description
ID	5
Name	**Manage Users**
Description	Allows the system administrator to manage staff accounts, passwords, and access roles.
Priority	**Should have (2)**
Actors	System Administrator
Pre-Conditions	Administrator must be logged into the system with full privileges.
Flow of Events	Details
Normal Flow	1. Admin selects the "Manage Users" function from the administration panel. 2. The system displays the list of existing users (Staff and Managers). 3. Admin adds a new user, updates an existing account, or deletes a user profile. 4. The system validates the input and saves the changes to the user database.
Alternative Flow	Alternative flow of events 1: If the user data is invalid (e.g., username already taken or weak password), the system displays an error message.
Post-Conditions	User accounts and roles are updated successfully in the system.

2.3.2.4 Sequence Diagram for Manage Users



2.4 List of Classes

The primary classes (objects) identified from the use cases are categorized using the Boundary, Control, Entity (BCE) pattern.

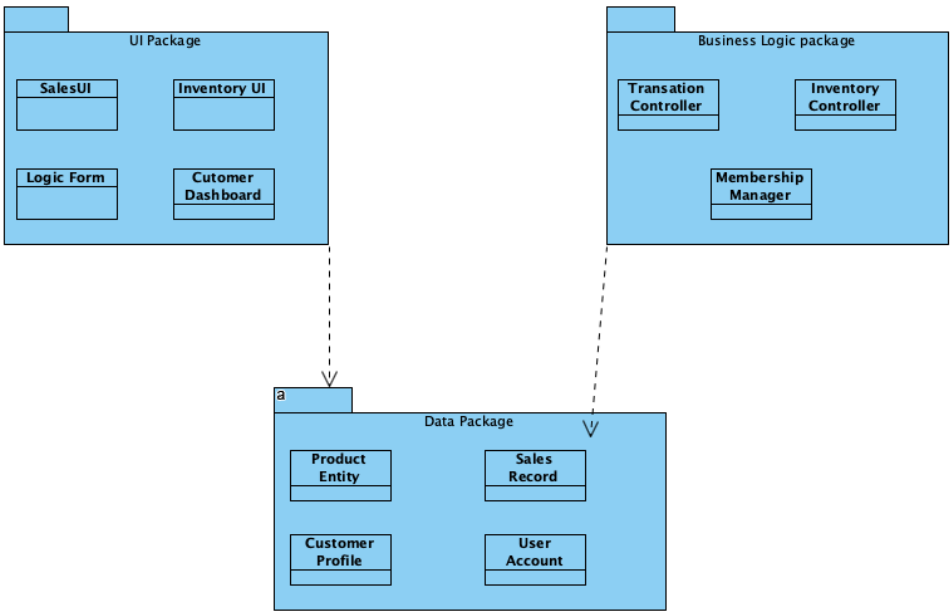
2.4.1 Interface (Boundary) classes

These classes represent the interaction points between the users and the POS system.

- **Sales Interface:** The main screen where the cashier scans items and calculates totals.
- **Inventory Management Screen:** Used by the manager to add, update, and track product stock.
- **Login Form:** The initial access point for all system users to authenticate their roles.
- **Customer Management Form:** The interface for registering new customers and viewing their purchase history.
- **Billing & Receipt Interface:** The screen used to finalize payments and generate customer invoices.
- **Reports Dashboard:** A screen for the manager to view sales performance and stock alerts.

1. Chapter 3: System Design

1.1 Application Architectural Using Package Diagram



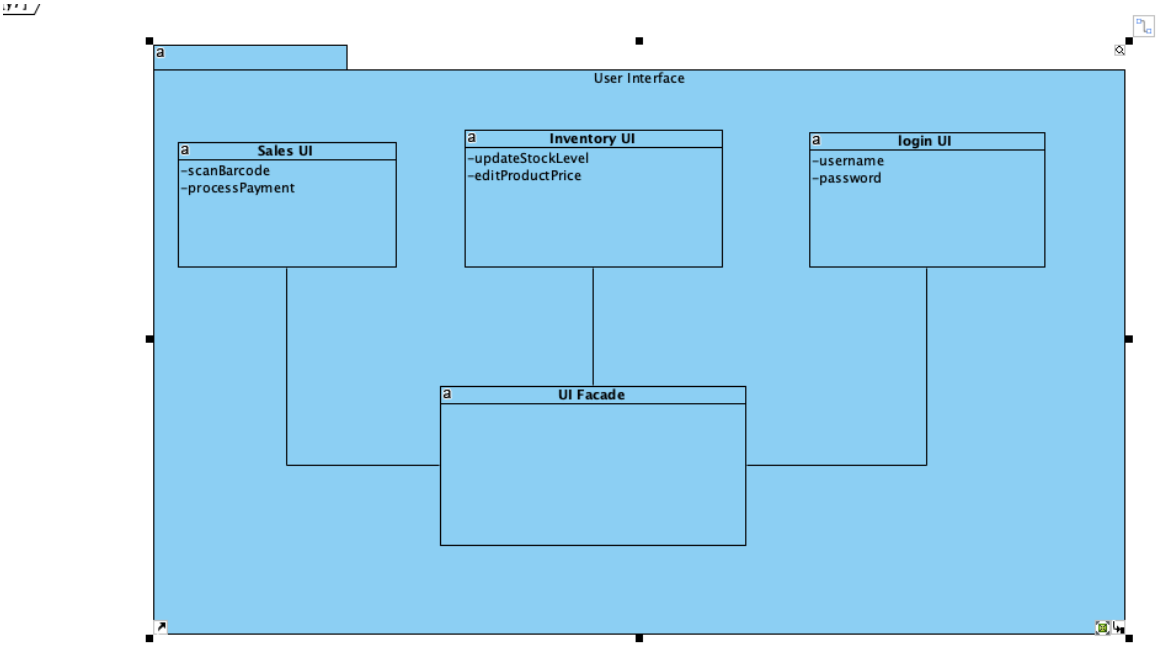
3.2 Modules (Packages) Descriptions

The **POS Management System** follows a three-tier architecture that separates the system into three main layers: User Interface, Application Logic, and Data Layer.

- **The User Interface package** handles all user interactions and screens, such as the **Sales Interface**, **Inventory Management Screen**, and **Customer Dashboard**.
- **The Application Logic package** contains the core business logic and system controllers, including the **Transaction Controller**, **Inventory Manager**, and **Membership Controller**.

- **The Data package** is responsible for managing data entities and database operations for **Products**, **Customers**, **User Accounts**, and **Sales Records**.
- **This separation** improves system maintainability, scalability, and ensures that changes in the UI do not directly affect the database structure

3.2.1 User Interface package



3.2.1.1 Sales Form Class

Instance: `Sales Form : Sales Form`

Attributes:

- `productId : int`
- `barcode : String`
- `quantity : int`
- `unitPrice : double`
- `totalAmount : double`

Methods:

- `displaySalesForm() : void`

- searchProduct(barcode) : void
- addItemsToCart() : void
- calculateTotal() : double
- processPayment() : boolean
- printReceipt() : void

3.2.1.2 Inventory Screen Class

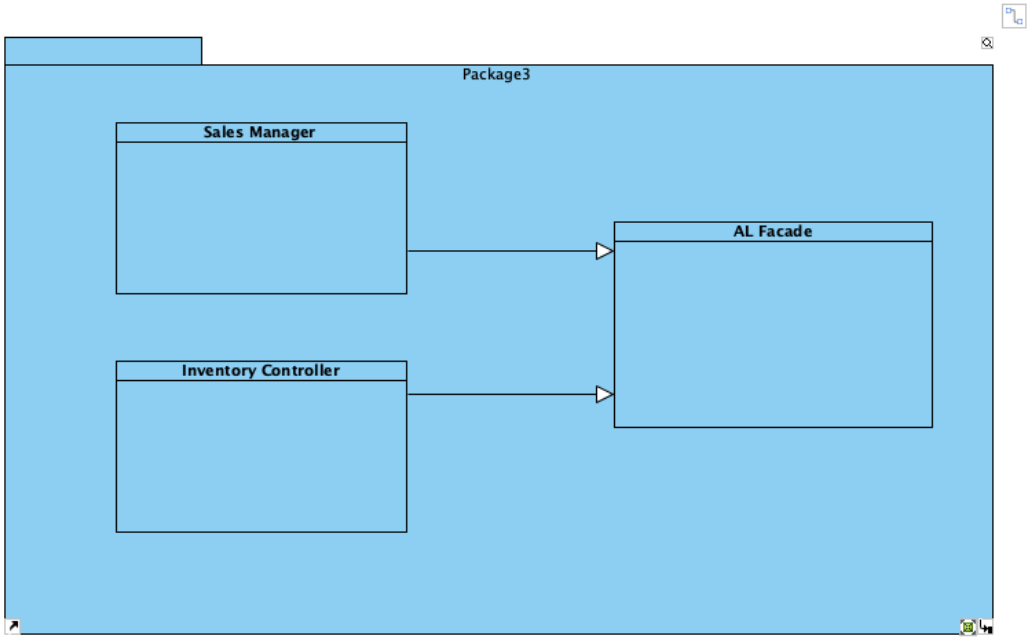
Instance: Inventory Screen : Inventory Screen

Attributes:

- productId : int
- productName : String
- supplierName : String
- barcode : String
- stockLevel : int

- **Methods:**
  - displayInventory() : void
  - filterByCategory(category) : void
  - updateStock(productId, amount) : void
  - viewProductDetails(productId) : void
  - generateLowStockAlert() : void

3.2.2 Business Logic package



3.2.2.1 Sales Manager class

Instance: `Sales Manager : Sales Manager`

Attributes:

- `cartItems : List<Product>`
- `totalPrice : double`

Methods:

- `checkProductAvailability(barcode, quantity) : boolean`
- `processSaleTransaction(cartItems, paymentMethod) : Sale`
- `calculateTotalAmount(items) : double`
- `saveSaleRecord(sale) : boolean`

3.2.2.2 Inventory Controller class

Instance: `Inventory Controller : Inventory Controller`

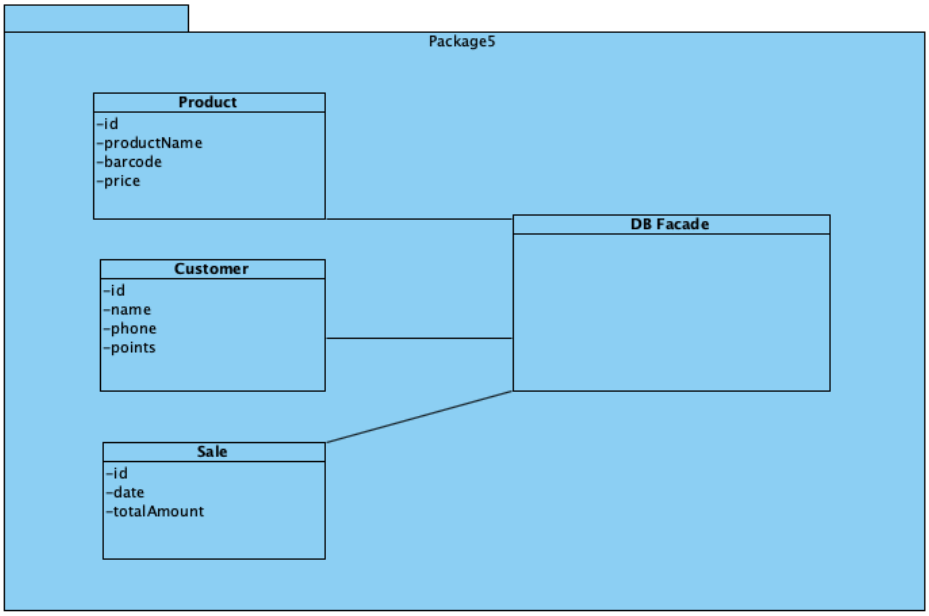
Attributes:

- `ProductList : List<Product>`
- `lowStockThreshold : int`

Methods:

- `searchProductByName(name) : List<Product>`
- `updateProductDetails(productId, newDetails) : boolean`
- `checkStockAvailability(productId) : int`
- `addNewProduct(productData) : void`
- `generateStockReport() : Report`

3.2.3 Data Package package



3.2.3.1 Product class

Instance: `product : Product`

Attributes:

- roductId : int
- productName : String
- barcode : String
- supplierName : String
- price : double

- **Methods:**

- getProductDetails() : String
- updatePrice(newPrice) : void
- updateStock(quantity) : void

3.2.3.2 Sale class

Instance: `sale : Sale`

Attributes:

- `saleId : int`
- `saleDate : Date`
- `totalAmount : double`
- `paymentStatus : String`
- `customerName : String`

Methods:

- `getSaleSummary() : String`
- `calculateTax() : double`
- `voidTransaction() : boolean`

4. Chapter 4: System Implementation and Testing

4.1 Coding and Coding Standards

The implementation of the **POS Management System** will adhere to the three-tier architecture designed in Chapter 3. To ensure high-quality, maintainable code, the following coding standards will be strictly followed:

1. Implementation Language and Methodology:

- A robust, widely-adopted programming language (**Java**) will be used for the Business Logic and Data Layers.
- The **Object-Oriented Programming (OOP)** methodology will be applied to ensure code reusability, modularity, and a strong separation of concerns between classes.

2. Naming Conventions:

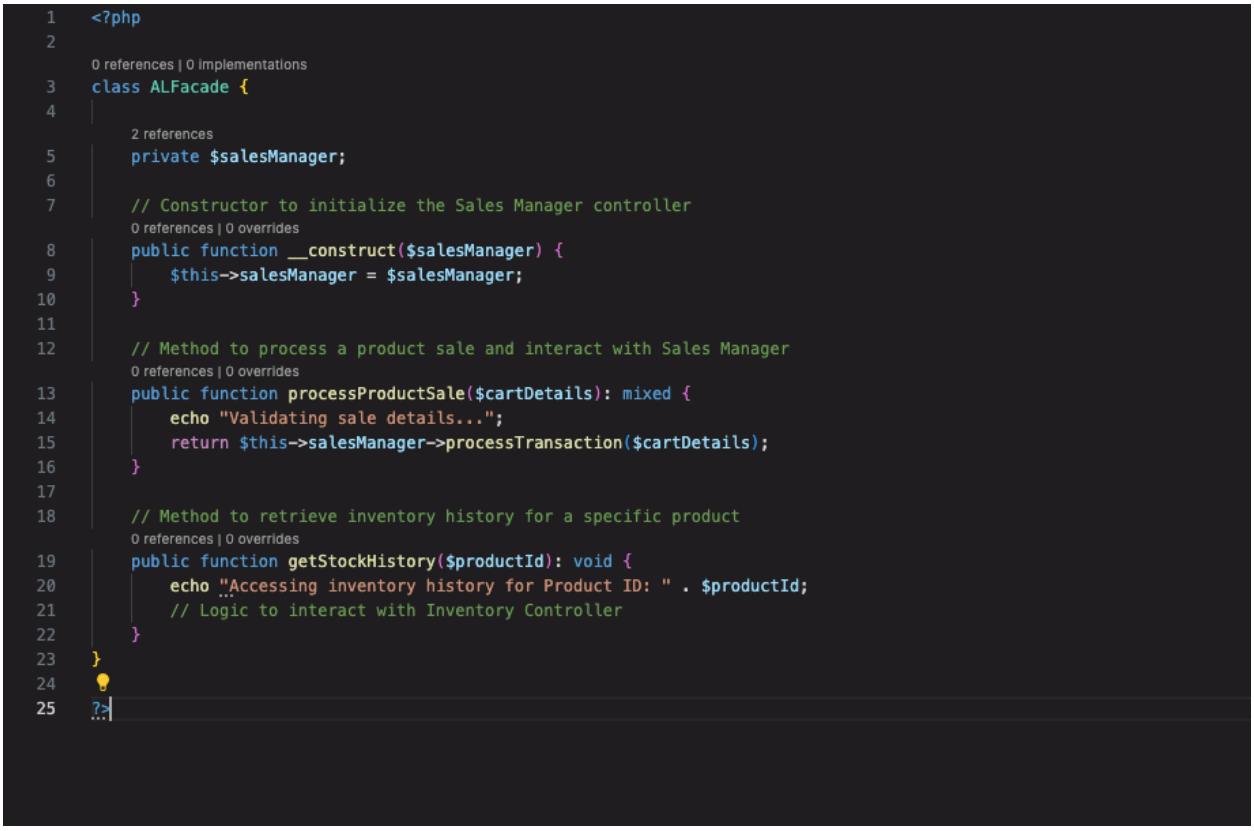
- **Classes and Constants** must use Pascal Case (e.g., `SalesManager`, `ProductForm`, `InventoryController`).
- **Variables and Methods** must use camelCase (e.g., `calculateTotalPrice`, `productId`, `updateStockLevel`).
- All names must be descriptive and clearly reflect the function or purpose of the element.

3. Documentation and Comments:

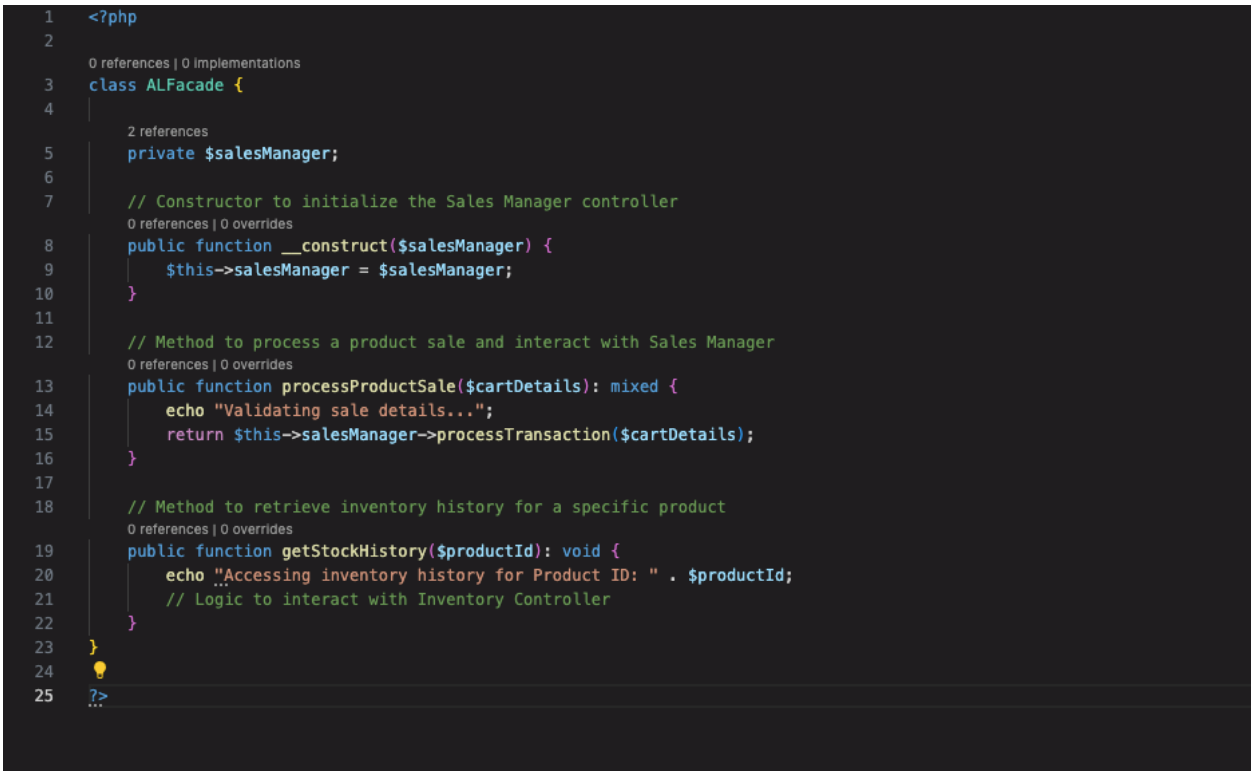
- **Inline Comments** must be used to explain complex logic or non-obvious code sections.
- **Standard documentation formats** (like Javadocs) must be used to clearly specify the purpose, parameters, and return values for all public methods and functions.

Users > macair > Desktop > yazan > yyy > ...

```php
<?php


class UIFacade {

    private $alFacade;

    // Constructor to initialize the Application Logic Facade

    public function __construct($alFacade) {
        $this->alFacade = $alFacade;
    }

    // Method to display the main Login Screen for Cashiers

    public function showLoginPage(): void {
        echo "Displaying POS Login Screen...";
    }

    // Method to submit a new sale transaction to the logic layer

    public function submitSaleOrder($cartDetails): mixed {
        // Logic to pass POS UI data to the Application Logic layer
        return $this->alFacade->processProductSale($cartDetails);
    }

    // Method to display the Inventory Management Screen

    public function showInventoryScreen(): void {
        echo "Loading Product Inventory...";
    }
}

?>
```



```php
<?php

0 references | 0 implementations
class ALFacade {

    2 references
    private $salesManager;

    // Constructor to initialize the Sales Manager controller
    0 references | 0 overrides
    public function __construct($salesManager) {
        $this->salesManager = $salesManager;
    }

    // Method to process a product sale and interact with Sales Manager
    0 references | 0 overrides
    public function processProductSale($cartDetails): mixed {
        echo "Validating sale details...";
        return $this->salesManager->processTransaction($cartDetails);
    }

    // Method to retrieve inventory history for a specific product
    0 references | 0 overrides
    public function getStockHistory($productId): void {
        echo "Accessing inventory history for Product ID: " . $productId;
        // Logic to interact with Inventory Controller
    }
}

?>
```



```php
<?php

0 references | 0 implementations
class ALFacade {

    2 references
    private $salesManager;

    // Constructor to initialize the Sales Manager controller
    0 references | 0 overrides
    public function __construct($salesManager) {
        $this->salesManager = $salesManager;
    }

    // Method to process a product sale and interact with Sales Manager
    0 references | 0 overrides
    public function processProductSale($cartDetails): mixed {
        echo "Validating sale details...";
        return $this->salesManager->processTransaction($cartDetails);
    }

    // Method to retrieve inventory history for a specific product
    0 references | 0 overrides
    public function getStockHistory($productId): void {
        echo "Accessing inventory history for Product ID: " . $productId;
        // Logic to interact with Inventory Controller
    }
}

?>
```

Users > macair > Desktop > yazan > yyy > ...

```php
<?php

/**
 * Database Connection Facade
 * POS Management System - January 2026
 */

0 references | 0 implementations
class DBFacade {

    /**
     * Initializes connection to the POS database
     */
    0 references | 0 overrides
    public function connect(): void {
        // Database connection string logic
        echo "Connected to POS System Database Successfully.";
    }

    /**
     * Executes saving data into a specific table (Products or Sales)
     */
    0 references | 0 overrides
    public function saveRecord($table, $data): bool {
        echo "Saving data into table: " . $table;
        return true;
    }
}
💡
?>
```

4.2 System Testing

A comprehensive testing process will be employed to ensure the **POS System** meets all functional and non-functional requirements specified in Chapter 2. The testing process includes three main phases:

1. Unit Testing:

- **Goal:** To test individual program units (methods or classes) in isolation to ensure they function correctly according to design.
- **Example:** Testing the `processSaleTransaction()` method within the **SalesManager** class to confirm it successfully calculates the total and updates the product stock in the database.

2. System Testing:

- **Goal:** To test the integrated components of the system as a whole and verify that non-functional requirements (like performance and security) are met.
- **Example:** Testing the end-to-end workflow of searching for a product by barcode, adding it to the cart, and completing a sale, verifying that the UI interacts correctly with the Business Logic and that the data is persisted in the database.

3. Acceptance Testing:

- **Goal:** To verify that the system satisfies the customer's (store staff and managers) expectations and meets the final business requirements.
- **Example:** Store cashiers and inventory managers will use the system in a real-world scenario to confirm ease of use, speed, and overall functionality before final deployment.

4.3 Source Code Repository

The complete source code, database scripts, and project files for the **POS Management System** are hosted on GitHub for version control and academic review.

Github Link:

Youtube Link:

1.2.1 Test Cases

#	Function	Test Steps	Test Data	Expected Results	Actual Results	Status
1	**User Login**	Access Login page .1 Call .2 `()showLoginPage`	N/A	Display "Displaying POS Login "...Screen	As Expected	Pass
2	**Add New Product**	Create Product object .1 Call .2 `()addNewProduct`	Name: Milk Barcode: 628123	Display "Product created successfully ".y	As Expected	Pass
3	**Process Sale**	Input sale details .1 Call .2 `)processProductSale(`	Date: 17-01-2026	Display "Validating sale "...details	As Expected	Pass
4	**DB Connection**	Initialize DB Connection .1 `()connect` Call .2	N/A	Display "Connected to POS System ".Database	As Expected	Pass

Status	Actual Results	Expected Results	Test Data	Test Steps	Function	#
Pass	As Expected	Return "True" and confirm data saving	Table: Sales	Submit sale record .1 Call .2 `()saveSaleRecord`	**Data Storage**	**5**
Pass	As Expected	Display "Current stock level for Product "502: 15	Product ID: 502	Enter Product ID .1 Call .2 `checkStockAvailability()`	**Check Stock**	**6**

1. References

[1] Ian Sommerville, “Software Engineering”, 10th Edition, Pearson Education, 2016.

[2] Object Management Group (OMG), “Unified Modeling Language (UML) Specification”, Available at: https://www.omg.org/spec/UML/, [Accessed: Jan 2026].

[3] Roger S. Pressman, “Software Engineering: A Practitioner’s Approach”, 9th Edition, McGraw-Hill Education, 2019.

[4] PHP Documentation Group, "PHP: Hypertext Preprocessor Manual", Available at: https://www.php.net/docs.php, [Accessed: Jan 2026].

[5] Visual Paradigm, "UML Modeling Tool for Software Design", Available at: https://www.visual-paradigm.com/, [Accessed: Jan 2026].

[6] Microsoft, "Visual Studio Code - Code Editing. Redefined", Available at: https://code.visualstudio.com/, [Accessed: Jan 2026].