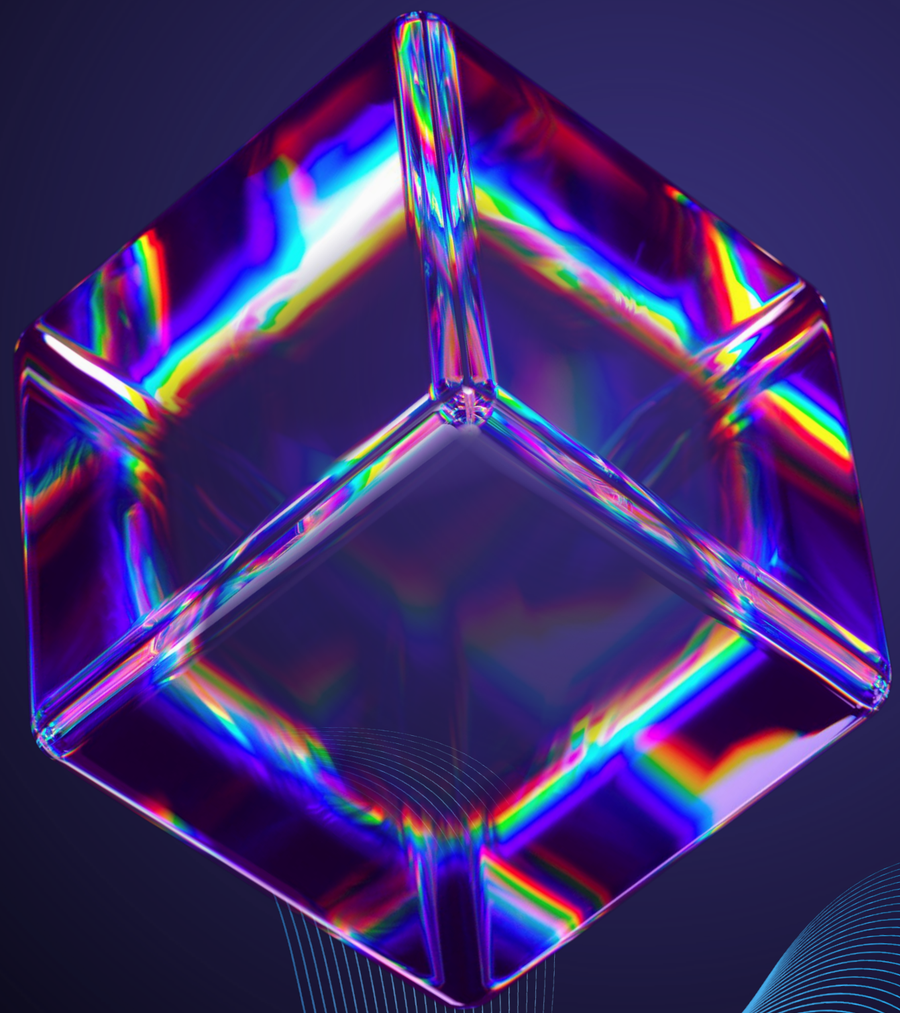
UTE UNIVERSITY of TECHNOLOGY & ENTREPRENEURSHIP

Smart POS & Inventory Management



Presented By

Lim Meng Huy
Chhum Chheanglim
Kang Soksambathratanak
Heng Longmeng

Lecturer:
SEK SOCHEAT

Class:
CSM

WHY THIS PROJECT IS NEEDED

Problem and Motivation

Current Problem

Small retail shops such as mini-marts still record sales and stock manually—on paper receipts or in spreadsheets kept separately by the shop owner. There is no shared system connecting what is actually sold at checkout to what remains on the shelf.

Who is affected

Cashiers cannot quickly check whether an item is still in stock while serving a customer, and owners cannot see accurate, up-to-date stock levels or daily sales totals without manually counting inventory at the end of the day.

Why a Web system would solve it

Every sale is linked to stock

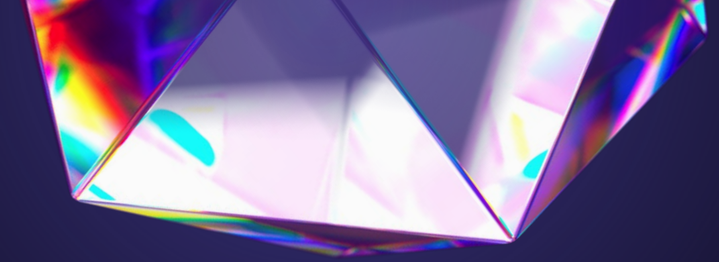
Checkout automatically deducts the exact quantity sold —no manual recount needed.

Real time, accurate inventory

The owner always sees the true stock count, not a number from yesterday's manual tally.

One shared source of truth

Admin, Admin Assistant, Inventory Manager, and Cashier all work from the same live data.



WHAT WE AIM TO ACHIEVE

Objectives

General Objective: Design and implement a web-based Point-of-Sale and Inventory Management System using Python, Flask, and MySQL that links every sale directly to stock levels through a clean, layered Object-Oriented design.

Secure role-based access

Authentication and authorization across all four roles: Admin, Admin Assistant, Inventory Manager, and Cashier.

Product & stock management

Full CRUD for products, with stock levels that stay accurate at all times.

Checkout workflow

A sales process that records each transaction and deducts stock automatically in real time.

Normalized MySQL database

A relational schema covering users, products, sales, sale items, and stock movements.

Layered OOP architecture

Models, services, and repositories, each verified with test cases — all within our 5-week timeline.

WHAT WE WILL AND WONT BUILD

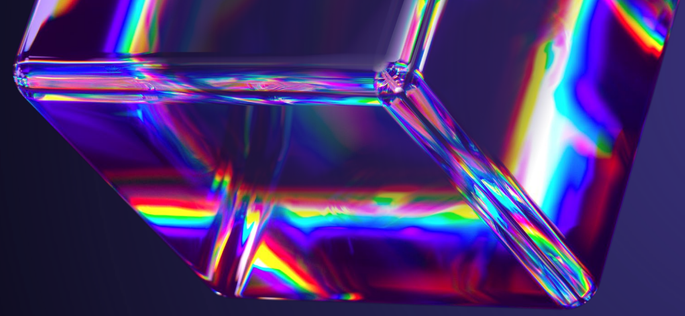
Scope &Limitation

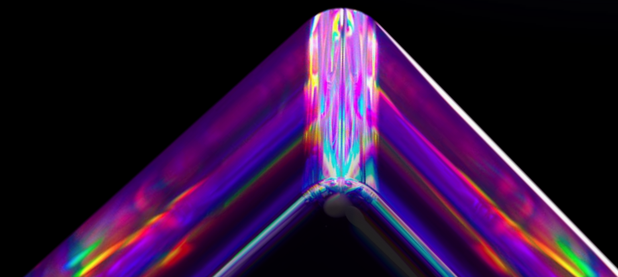
Included in Scope (Core)

- Four user roles : Admin, Admin Assistant, Inventory Manager, Cashier
- Product management (create, update, remove, view) by Admin
- Checkout workflow that records a sale and deducts stock automatically
- Stock/inventory tracking with low-stock visibility
- Product search/filter by name or category
- Sales report/summary page (daily totals, best-selling items)
- QR CODE payment
- Promotion and Discount (Percentage)

Excluded from Scope

- Bardcode hardware scanning (Manual product code entry only)
- Multi-branch / Multi-store support
- Online Payment gateway integration



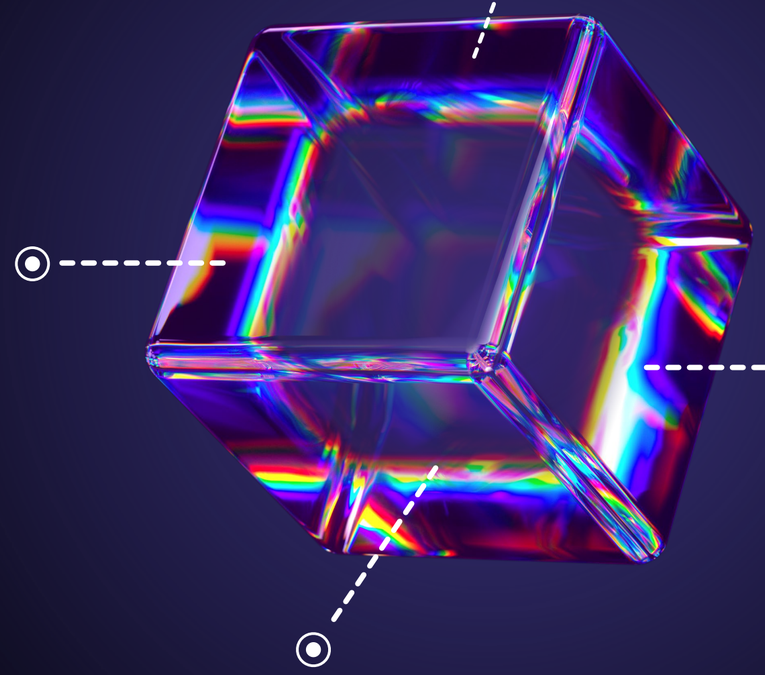


Who uses the system

Target Users & Roles

Admin

Full system access: manages user accounts, products, stock, and views all sales reports.



Admin Assistant

Supports the Admin day-to-day — views sales reports and manages cashier accounts, but cannot delete products or change system settings.

Cashier

Processes checkout transactions at the point of sale and views current stock while serving customers.

Inventory Manager

Manages products and stock levels directly — add, update, remove products, adjust stock, view stock reports.

Proposed

Proposed System Features

01

Authentication & role -based authorization

Secure login/logout with password hashing and session management across all four roles.

02

Product & stock management

Full CRUD for products; every sale automatically updates the stock quantity.

03

Checkout / sales workflow

Supports multiple items per sale, validates stock availability, and calculates totals.

04

Search & filter

Find products quickly by name or category during checkout or stock review.

05

Sales report

Daily sales totals and best-selling items, visible to Admin and Admin Assistant.

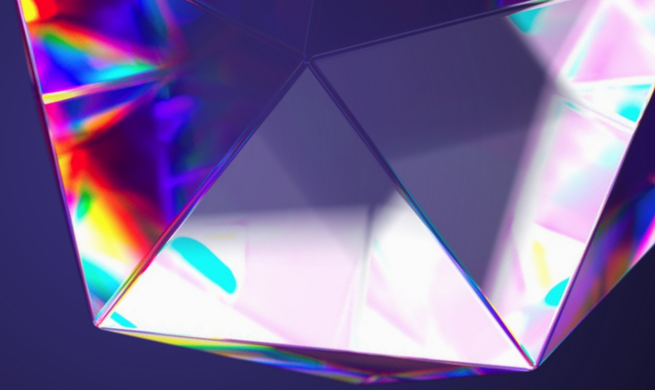
06

Percentage discount & exportable report

Discount at checkout and a printable/exportable sales report.

CLASSES, RESPONSIBILITIES, RELATIONSHIPS

OOP Design Plan



User (Base class)

Admin

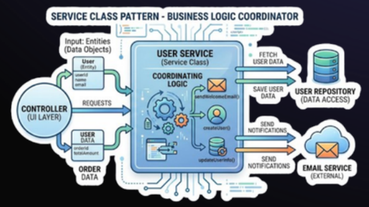
Assistant Admin

Inventory Manager

Cashier

Domain / Model Classes

Product, Sale, SaleItem, StockMovement represent core data and behavior.



Service Classes

AuthService, SalesService, InventoryService, ReportService —apply business rules, e.g. rejecting a sale that exceeds available stock.

Repository Classes

UserRepository, ProductRepository, SaleRepository, StockMovementRepository only these touch MySQL directly.

HOW THE LAYERS CONNECT

System Architecture

Templates UI

Jinja2 pages render data received from routes; no business logic lives in a template.

Browser

The user's web browser sends requests (e.g. submitting a checkout form).

Flask Routes

Receive HTTP requests and call the right service—no business logic here.

MySQL

Stores users, products, sales, sale items, and stock movements.

Services

Apply business rules, e.g. rejecting a sale that exceeds stock.

Repositories

Execute parameterized SQL queries—the only layer that touches the DB.

★★★★★

Lim Meng Huy

“ Each layer has exactly one responsibility. Routes never contain SQL. Services never talk to the browser directly. Repositories never contain business rules. This separation is what lets the team test, debug, and extend one layer without breaking the others.” HUY said.

ERD DRAFT AND KEY TABLES

Database Plan

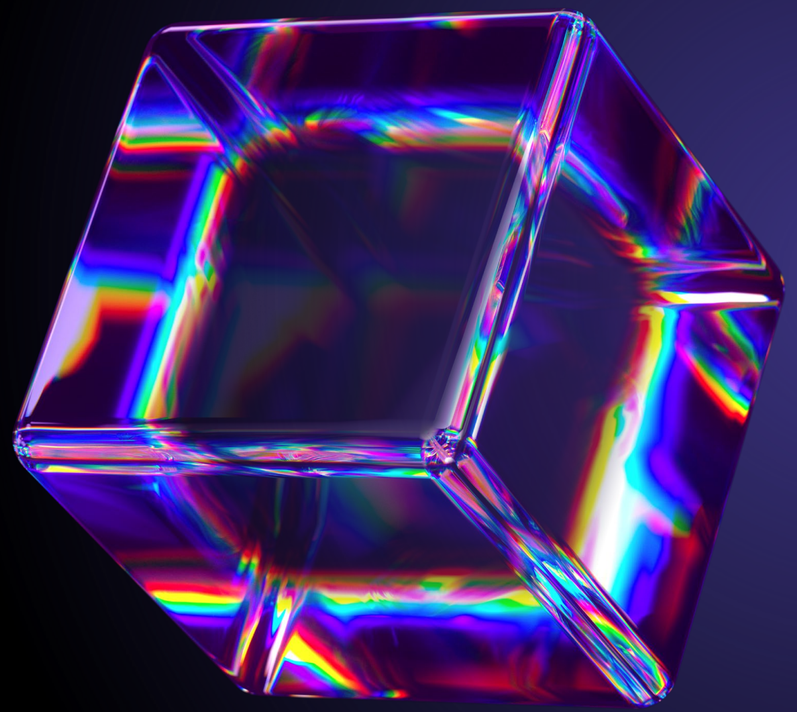
users	id (PK), name, email (unique), password_hash, role (admin / admin_assistant / inventory_manager / cashier)
roles	id (PK), name (admin / admin_assistant / inventory_manager / cashier)
permissions	id (PK), cashier_id (FK→users.id), total_amount, created_at
role_permissions	role_id (FK→roles.id), permission_id (FK→permissions.id) — junction table defining which permissions each role holds
products	id (PK), name, category, price, quantity_in_stock
sales	id (PK), cashier_id (FK → users.id), discount_percent, payment_method, total_amount, created_at
sale_items	id (PK), sale_id (FK→sales.id), product_id (FK→products.id), quantity, unit_price
stock_movements	id (PK), product_id (FK→products.id), change_amount, reason, created_at

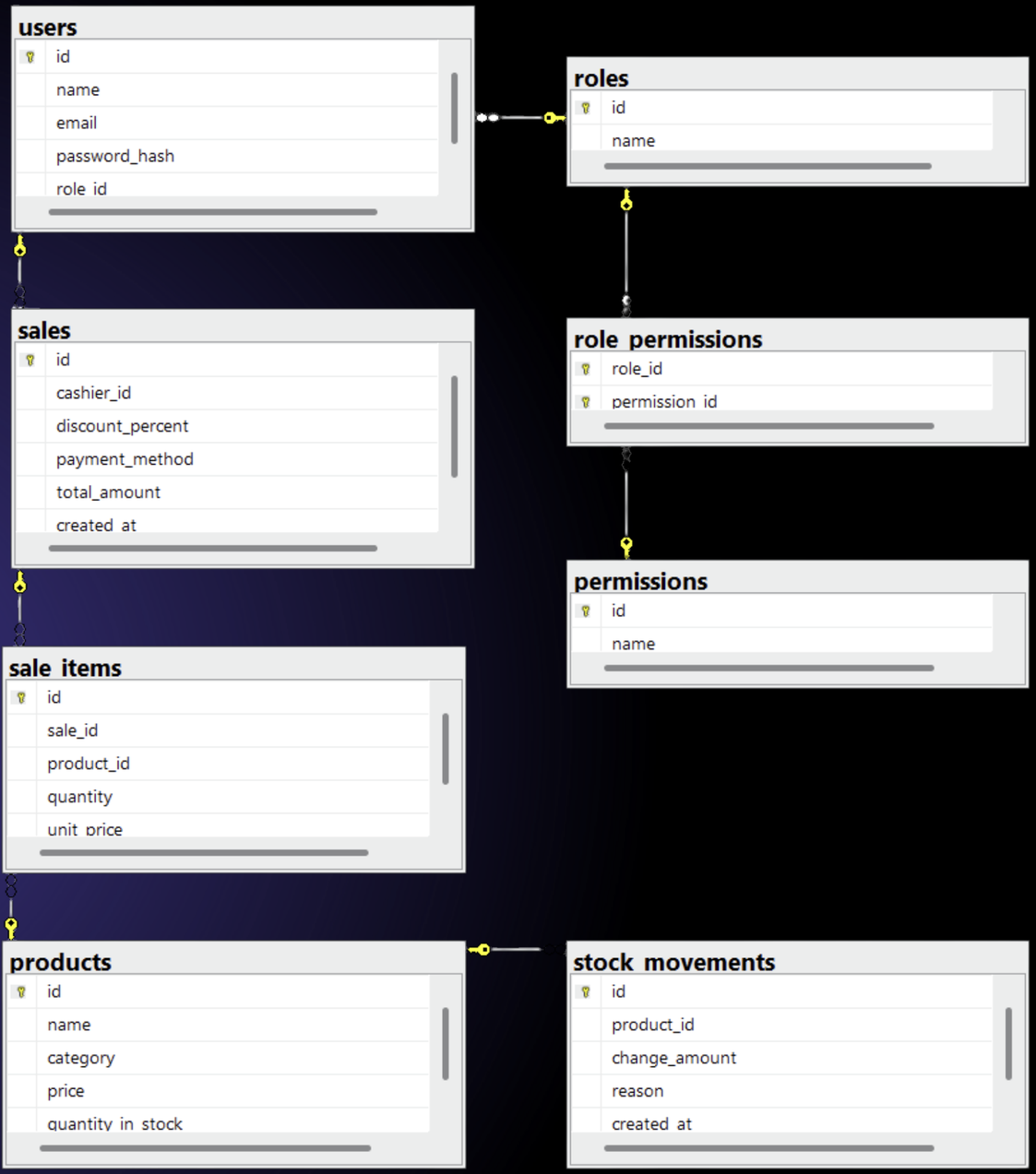
Design Standards

- Normalized schema with foreign keys enforcing valid relationships
- Normalized schema with foreign keys enforcing valid relationships
- Parameterized queries only—no string concatenation with user input
- Passwords stored only as hashes, never plain text

Timeline

Database Diagram





Our 5 Week Sprint Plan(ಥ _ ಥ)

Methodology & Timeline

1

Week 1

Finalize scope, ERD & domain classes, Git repo, Flask project structure, MySQL schema.

2

Week 2

Authentication, role-based access, and Product management (Admin CRUD).

3

Week 3

Checkout/sales workflow (Cashier) with automatic stock deduction and validation.

4

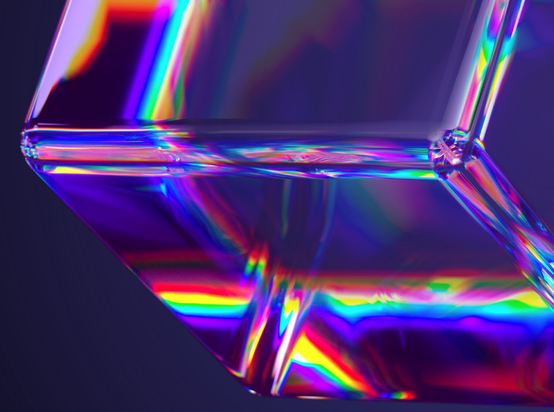
Week 4

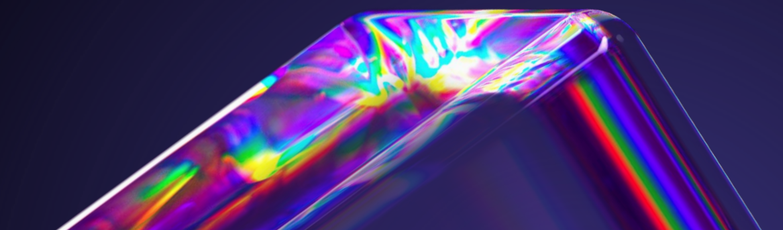
Search/filter, sales report page; begin stretch goals if on schedule.

5

Week 5

Test cases, bug fixes, screenshots, final report, defense/demo prep.





WHAT WE WILL DELIVER

Expected Output

Working Flask Application

Adeployable, demonstrable web app connected to a live MySQL database.

Normalized Database

Full schema with seed data for users, products, sales, and stock movements.

Documented Test Cases

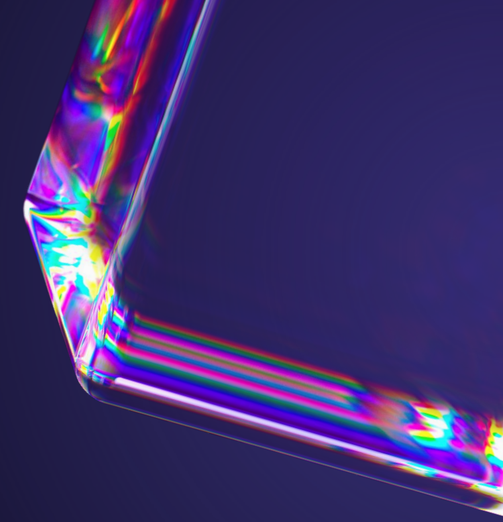
At least 10 test cases with evidence and screenshots covering core business rules.

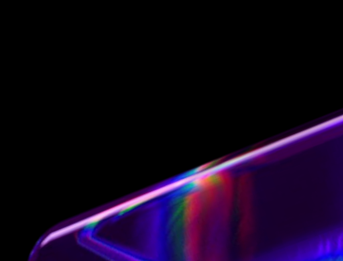
Final Report

Complete documentation of our design, implementation, and testing process.

Live Demonstration

Afull sale transaction shown end-to-end, from checkout to updated stock and reports.





Thank you!

Have any Question?

+855 6711967

UTEMinimart@ute.com