Software Construction and Development

Submitted by: Malik Muhammad Falak Sher, Hamza Nawaz, Shanzay Umer

Submitted to: Sir Ali Haider

Date: 10-January-2025

Department of Software Engineering, Lahore Garrison University

Project Overview: Inventory Management System (IMS)

The Inventory Management System (IMS) streamlines retail and warehouse operations, offering a centralized interface for staff, suppliers, product tracking, and customer transactions.

Dashboard

High-level system overview.

Employee Management

CRUD operations for staff records.

Product & Category

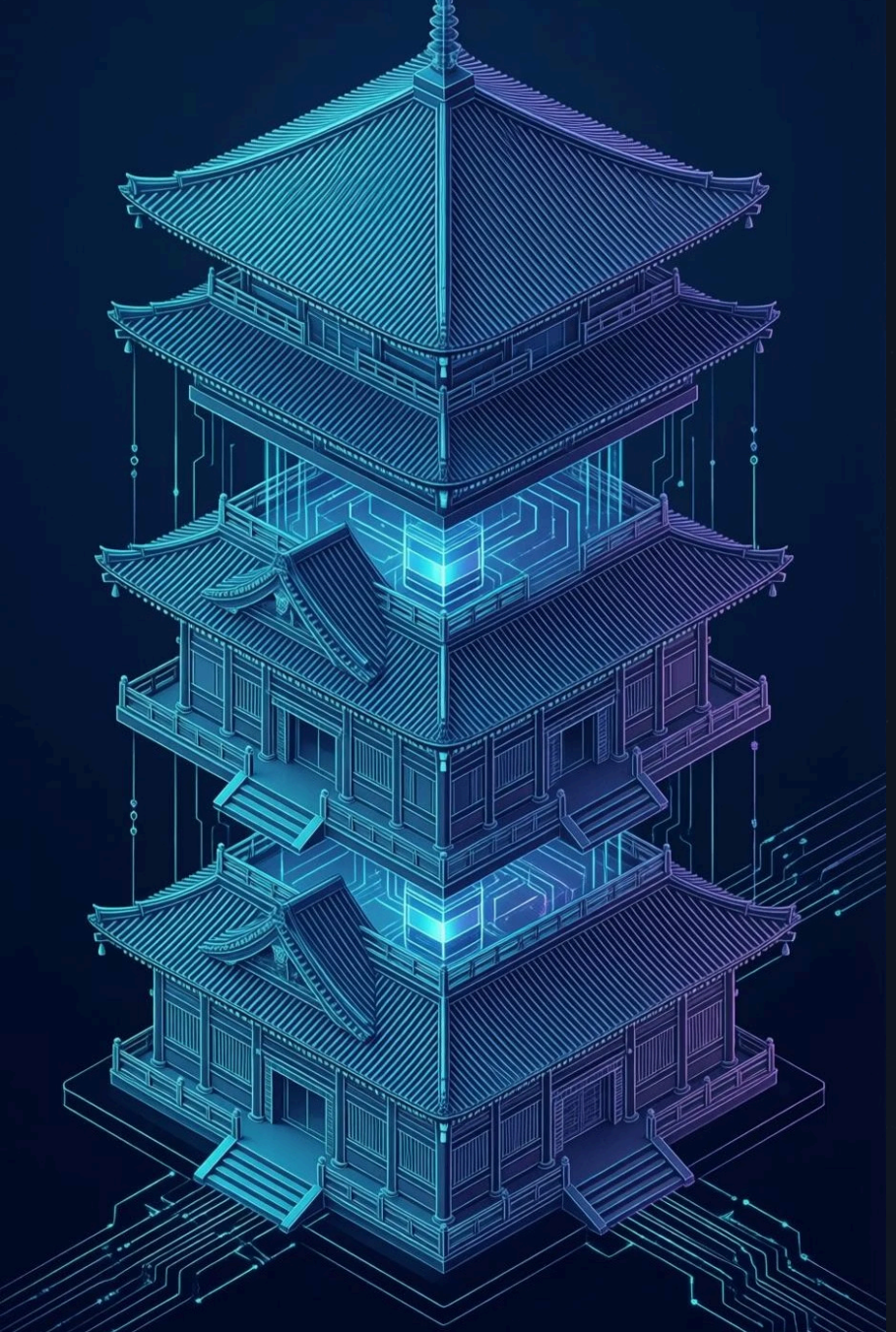
Inventory tracking with status.

Billing System

Transaction engine, calculator, receipt generator.

Sales History

File-based logging of past invoices.



System Architecture: 3-Tier Design

Our system employs a 3-Tier Architecture for clear separation of concerns, a primary goal during refactoring.

Presentation Layer (GUI)

Developed with Tkinter, handling user input and data display.

Business Logic Layer

Resides in database_manager.py, processing queries and calculations.

Data Tier

SQLite3 database (ims.db) for structured data and file system for receipts.

Component Details: Dashboard & Database Manager

The Dashboard (dashboard.py)

Serves as the entry point, using a polling mechanism to update product, employee, and sales counts every 200ms without UI freezing.

Refactoring: Reduced 5 SQL queries/second to a single db.get_dashboard_counts() call to minimize database overhead.

Database Manager (database_manager.py)

Critical for Software Process Improvement (SPI), centralizing all SQL code.

- **Code Reusability:** Single get_connection() function.
- **Security:** Parameterized Queries prevent SQL Injection attacks.

The Billing Engine (billClass.py)

This component manages the complex relationship between inventory and sales.

1

Stock Synchronization

Automatically subtracts sold quantity from product stock; sets status to "Inactive" if stock hits zero.

2

Calculator Logic

Provides on-the-spot math for sales representatives using sanitized string evaluation.

Software Engineering Principles Applied

1 Lehman's Laws of Software Evolution

Law of Continuing Change: System evolved from basic tracking to include sales history (sales.py).

Law of Increasing Complexity: Linking billing to product database increased dependencies, requiring robust error handling.

2 Exception Handling & Robustness

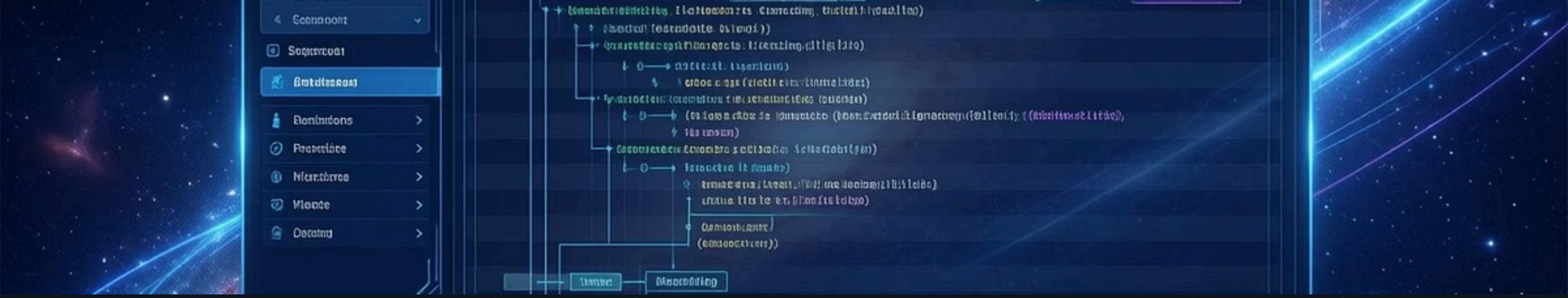
Moved from "Direct Use" to "Safe Use" by wrapping database and file I/O in try-except blocks.

```
try:
  fp = open(bill_path, 'w')
  fp.write(content)
except IOError as e:

messagebox.showerror("File Error", f"Could not save bill: {e}")
```

3 Automated Unit Testing

Implemented test_billing_logic.py to verify discount and Net Pay calculations for accuracy.



Version Control History (SCM)

Git was utilized to manage the project's lifecycle and track key milestones.

1

Alpha

Basic CRUD for Employees.

2

Beta

Billing engine and Product stock linking.

3

Refactor Phase

Migration of raw SQL strings to database_manager.py.

4

Final Polish

Implementing absolute paths for cross-environment compatibility.

Justifying Lehman's Law

Law of Continuing Change

Software must adapt or become less useful. Our project evolved from basic database creation to complex features like "Billing" and "Sales Reports," demonstrating continuous adaptation.

Law of Increasing Complexity

Complexity increases unless managed. We used **Modularization**, splitting code into separate classes (employee.py, product.py) to keep the system manageable.

Software Process Improvement (SPI) & Peer Reviews

SPI: Version Control (Git)

Identified process weakness (lack of safety net for code changes) and implemented Git for change history, improving configuration management.

Code Level SPI: Defect Prevention

Removed "Command Error" in dashboard.py (Line 54) to prevent crashes, improving code quality.

Peer Reviews: Automated Code Inspection

A walkthrough of dashboard.py found a TclError, resolved by removing an invalid parameter, preventing a runtime crash.

Team Roles & Learning Outcomes

Backend Developer	Designed SQLite schema and SQL queries (Falak Sher).
Frontend Developer	Built Tkinter GUI layout (Hamza Nawaz).
QA Engineer	Fixed bugs, ensured deployment, implemented unit testing (Shanzay Umer).

Final Missing Link: Unit and Automated Testing were run, resulting in an "OK" report.

Learning Outcomes

- **Refactoring:** Not just cleaning, but making code testable.
- **User Experience (UX):** Depends on background processes like the live-updating dashboard.
- **Data Integrity:** Paramount, achieved through atomic database transactions.