Document Restructuring
Legacy POS System

Reorganizing and Modernizing Technical Documentation
for the Existing File-Based POS Application

Version 1.0 2025-11-28

Status: **In Progress**

Contents

1 Overview

This document captures the **document restructuring phase** of the reengineering process. The goal is to reorganize, update, and recreate all technical documentation so that it accurately reflects the *actual* current system — not the originally intended design.

The legacy documentation is outdated, incomplete, and scattered. This restructuring establishes a modern, maintainable, and accurate documentation suite.

2 Legacy Documentation Inventory

2.1 Existing Documentation (from Documentation/ folder)

- **Inception Phase:** Business Rules.docx, Glossary.docx, Vision.docx, Use Cases Draft.docx, Supplementary Specification.docx, WBS and Responsibility Matrices
- **Elaboration Phase:** SAD.docx (multiple versions), Process diagrams (Activity, Class, Domain, Sequence), Package Diagram.png, Responsibility Matrix.xlsx
- **Construction Phase:** Beta Release notes, Developer Manual.docx, User Manual.docx, White box test documentation, Bug reports
- **Beta Release:** Change logs, updated manuals

2.2 Documentation Gaps Identified

1. No current architecture diagrams (existing ones do not match code)
2. Outdated class diagrams
3. Missing data model for file-based storage
4. No API/interface documentation
5. Incomplete test documentation
6. No deployment or configuration guide
7. Hardcoded paths and OS logic undocumented

3 Restructured Documentation Structure

Document	Location & Purpose
System Overview	`docs/system-overview.md` – High-level purpose and boundaries
Architecture	`docs/architecture/` – Reverse-engineered diagrams and flows
Data Model	`docs/data-model/` – File formats, data dictionary, migration plan
API/Interfaces	`docs/api/` – Public methods and UI contracts
Testing	`docs/testing/` – Strategy, coverage, test cases
Development	`docs/development/` – Build, setup, standards, refactoring log

4 Reverse-Engineered Diagrams

4.1 Class Diagram (Simplified – Reverse Engineered)

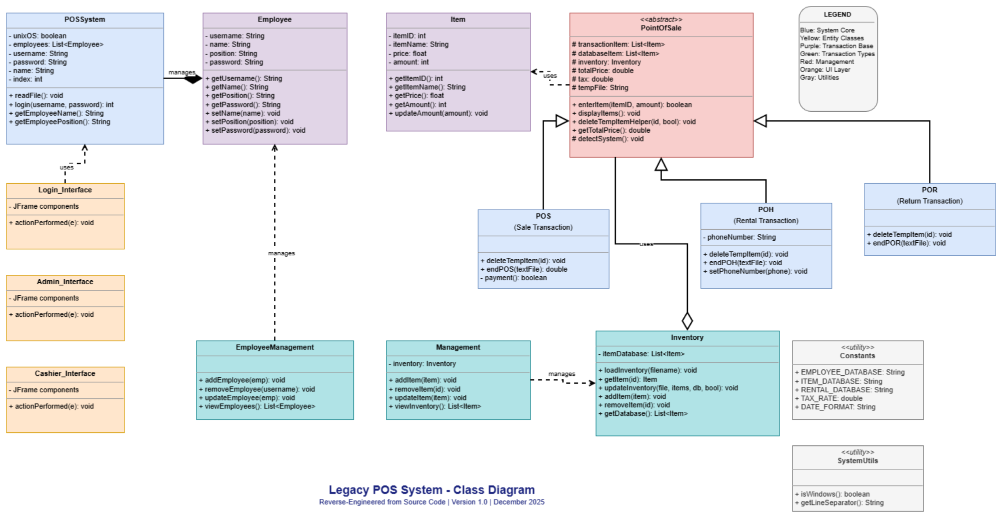

Figure 1: Reverse-Engineered Class Diagram (Current Actual Structure)

4.2 Sequence Diagram: Sale Transaction

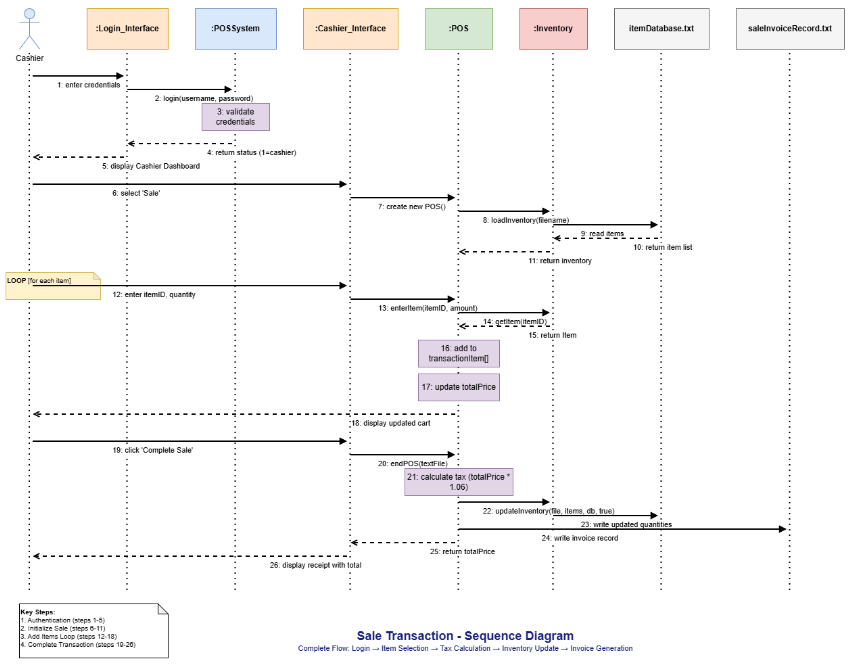
Cashier
:Login_Interface
:POSSystem
:Cashier_Interface
:POS
:Inventory
itemDatabase.txt
saleInvoiceRecord.txt
1: enter credentials
2: login(username, password)
3: validate credentials
4: return status (1=cashier)
5: display Cashier Dashboard
6: select 'Sale'
7: create new POS()
8: loadInventory(filename)
9: read items
10: return item list
11: return inventory
LOOP [for each item]
12: enter itemID, quantity
13: enterItem(itemID, amount)
14: getItem(itemID)
15: return Item
16: add to transactionItem[]
17: update totalPrice
18: display updated cart
19: click 'Complete Sale'
20: endPOS(textFile)
21: calculate tax (totalPrice * 1.06)
22: updateInventory(file, items, db, true)
23: write updated quantities
24: write invoice record
25: return totalPrice
26: display receipt with total
Key Steps:
1. Authentication (steps 1-5)
2. Initialize Sale (steps 6-11)
3. Add Items Loop (steps 12-18)
4. Complete Transaction (steps 19-26)
Sale Transaction - Sequence Diagram
Complete Flow: Login → Item Selection → Tax Calculation → Inventory Update → Invoice Generation

Figure 2: Sequence Diagram – Complete Sale Transaction Flow

5 Data Dictionary

5.1 Employee Database Format

File: `Database/employeeDatabase.txt`
Format: `username position firstName lastName password`
Example: `110001 Admin Harry Larry 1`

5.2 Item Database Format

File: `Database/itemDatabase.txt`
Format: `itemID itemName price amount`
Example: `1000 Potato 1.0 249`

5.3 User/Rental Database Format

File: `Database/userDatabase.txt`
Format: `phoneNumber itemID1,date1,returned1 itemID2,date2,returned2 ...`
Example: `1234567890 1022,6/31/11,false 1023,7/15/11,true`

6 Operational Scenarios

6.1 Scenario 1: Cashier Login and Process Sale

1. Cashier launches app and logs in with valid credentials
2. System returns status 1 (cashier)
3. Cashier selects "Sale" → new `POS` instance created
4. Item database loaded into memory
5. Items scanned, total updated
6. Transaction completed → tax applied, inventory reduced, invoice written

6.2 Scenario 2: Process Rental

1. Customer phone entered → checked/created in `userDatabase.txt`
2. Items added to rental cart
3. Total calculated → inventory reduced
4. Rental record appended to user file

6.3 Scenario 3: Process Return

1. Customer phone entered → outstanding rentals displayed
2. Cashier selects items to return
3. Inventory restored, rental entries marked `true`

7 Known Issues Documented

- Plain-text passwords (critical security flaw)
- No file locking → race conditions
- No transaction support → partial writes possible
- Entire files loaded on every operation
- Hardcoded paths and OS-specific logic
- Tight coupling → difficult to test
- Swallowed exceptions and poor error messages

8 Documentation Standards Established

- All new documentation in **Markdown** (Git-friendly)
- Diagrams in **PlantUML/Mermaid** source + exported PNG/PDF
- Version control for all docs
- Consistent naming and folder structure
- Cross-references and glossary

9 Next Steps

1. Generate accurate class diagrams from source code (using PlantUML C4 or javadoc)
2. Create sequence diagrams for all major flows
3. Fully document all file formats with examples
4. Write deployment and configuration guide
5. Establish coding standards
6. Begin systematic refactoring log

Document Version: 1.0
Date: 2025-11-28
Status: In Progress