Inventory Analysis – Legacy POS Baseline

December 7, 2025

1 Scope and Objectives

- Establish authoritative list of inherited artefacts before restructuring.
- Classify each item as **Keep/Refactor**, **Replace**, or **Archive** to guide reengineering phases in the project brief.
- Capture immediate risks, tooling assumptions, and gaps to unblock reverse-engineering and documentation updates.

2 Repository Snapshot (Master Branch)

3 Module Inventory (from README + code survey)

4 Dependency Highlights

- File-based persistence hard-coded via relative paths across multiple classes → strong coupling, no abstraction.
- UI classes directly instantiate transaction classes, bypassing service or repository layers.
- Logging relies on appending to `Database/employeeLogfile.txt`, no rotation/security.
- Build pipeline tied to NetBeans; no standalone Gradle/Maven definition yet.

5 Asset Classification & Actions

6 Risks & Gaps

Environment: No JDK configured on workstation (`javac` missing) → must install to run/trace legacy app before deeper reverse engineering.

Data Integrity: Text files lack schema constraints; duplicates or malformed entries likely when migrating.

Testing Debt: Only one legacy test; need characterization tests before refactoring.

Security: Credentials stored plaintext; logging exposes sensitive info.

Category	**Asset(s)**	**Description**	**Status**
Source code	`src/*.java` (UI + domain + persistence blended)	Swing/AWT-based desktop POS with procedural flows, heavy coupling to `.txt` stores	Keep → Refactor
Tests	`tests/EmployeeTest.java` (stub)	Minimal characterization coverage	Keep → Expand
Build scripts	`build.xml`, `manifest.mf`, `nbproject/`	NetBeans + Ant project metadata	Keep (reference)
Executables	`SGTechnologies.jar`, `src.zip`, `jarFile.jar`	Legacy binaries for behavior reference	Archive
Data stores	`Database/*.txt`	Flat files for employees, rentals, inventory, logs,	Replace → migrate to RDBMS

Module	Responsibility	Dependencies	Notes
`POSSystem`	App bootstrap, login, role routing	`Employee`, text DB files	God-class; mixes UI, logic, IO
`Employee` / `EmployeeManagement`	Employee entity, CRUD, auth	DB text files	Passwords stored in plaintext
`Inventory`, `Item`	Stock access/update (Singleton)	`Database/itemDatabase.txt`	Inventory writes lack locking
`POS`, `POR`, `POH`, `PointOfSale`	Sales/Rental/Return workflows	Inventory, DB files	Abstract factory pattern
`Register`, `Sale`, `Rental`, `ReturnItem`	Transaction handling	Various DB files	Duplication across flows
UI forms `*_Interface`	Swing screens	Business modules	Direct file IO from UI events

Asset	**Classification**	**Action**
Core domain classes (`Employee`, `Item`, `Inventory`)	Reuse with refactor	Extract to service layer, add tests
UI Swing forms	Replace	Will be superseded by web UI
Flat-file databases	Replace	Model relational schema + migrations
Ant/NetBeans configs	Reference	Keep for provenance, but introduce modern build tool (Gradle/Maven)
Legacy docs (SAD, WBS, manuals)	Rework	Consolidate into updated documentation set with comparison tables
Executable JARs	Archive	Use only for behavior validation

7 Immediate Next Steps

1. Install/configure JDK 8+ and document build instructions.
2. Add Gradle/Maven wrapper to standardize builds/tests outside NetBeans.
3. Write high-level architecture diagram of current system using findings above.
4. Begin characterization tests around `POSSystem` login and `Inventory` operations to guard behavior pre-refactor.