

Forward Engineering Web-Based POS System

Modern Reimplementation Using
Spring Boot + React + PostgreSQL

Version 1.0 2025-11-28

Status: **Ready for Implementation**

Contents

1 Overview	2
2 Technology Stack Selection	2
2.1 Backend: Spring Boot	2
2.2 Frontend: React + TypeScript	2
2.3 Database: PostgreSQL	2
3 Architecture Design	2
3.1 Layered Architecture Diagram	3
4 Project Structure	3
4.1 Backend Structure (Spring Boot)	3
4.2 Frontend Structure (React + TypeScript)	4
5 Key Features Implementation	5
6 API Design	5
6.1 RESTful Endpoints Summary	5
7 Design Patterns Applied	5
8 Testing Strategy	6
9 Deployment Strategy	6
10 Migration from Legacy System	6
11 Improvements Over Legacy System	6

1 Overview

This document defines the **forward engineering phase** of the POS reengineering project: building a modern, scalable, secure, and maintainable **web-based Point of Sale system** from the ground up using industry-standard technologies.

The new system replaces the fragile file-based desktop application with a robust full-stack web architecture while preserving all business logic and achieving feature parity — and significant improvements.

2 Technology Stack Selection

2.1 Backend: Spring Boot

Rationale: Java ecosystem reuse, mature framework, excellent JPA/Security/REST support, built-in testing, production-ready.

2.2 Frontend: React + TypeScript

Rationale: Component-based UI, type safety, virtual DOM performance, rich ecosystem (Material-UI, React Router), hot reload, industry dominance.

2.3 Database: PostgreSQL

Rationale: ACID compliance, rich data types, excellent Spring Data JPA integration, high concurrency, open source.

3 Architecture Design

3.1 Layered Architecture Diagram

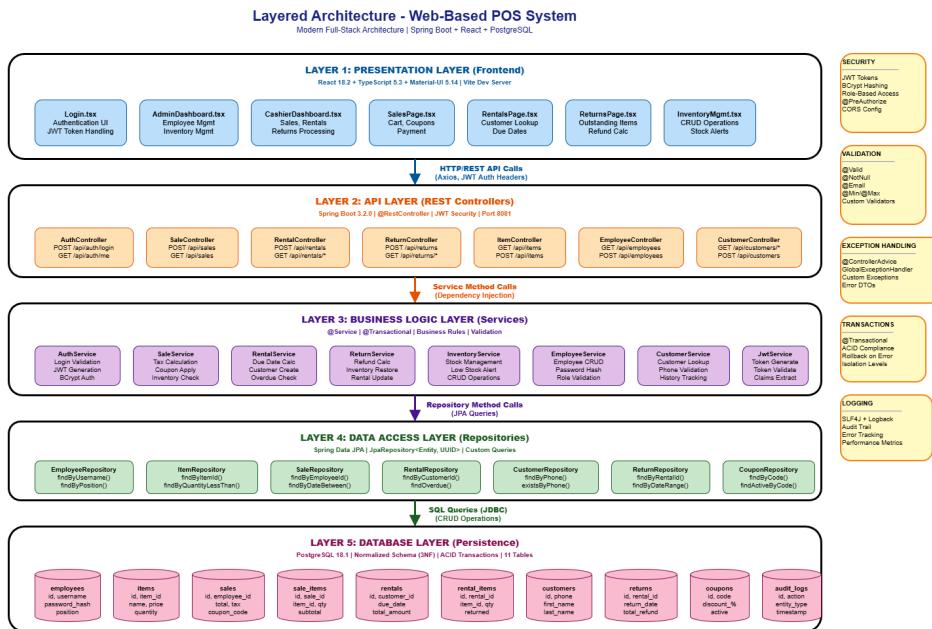


Figure 1: Layered Architecture of the New Web-Based POS System

4 Project Structure

4.1 Backend Structure (Spring Boot)

```
pos-backend/
src/main/java/com/sgtech/pos/
    PosApplication.java
    config/
        SecurityConfig.java
        WebConfig.java
    controller/
        AuthController.java
        SaleController.java
        RentalController.java
        ReturnController.java
        InventoryController.java
        EmployeeController.java
    service/
        AuthService.java
        SaleService.java
        RentalService.java
        ReturnService.java
        InventoryService.java
        EmployeeService.java
    repository/
        EmployeeRepository.java
        ItemRepository.java
        SaleRepository.java
```

```
RentalRepository.java  
CustomerRepository.java  
ReturnRepository.java  
model/  
    Employee.java  
    Item.java  
    Sale.java  
    Rental.java  
    Customer.java  
    Return.java  
dto/  
    LoginRequest.java  
    SaleRequest.java  
    RentalRequest.java  
    ReturnRequest.java  
exception/  
    GlobalExceptionHandler.java  
    ResourceNotFoundException.java  
src/main/resources/  
    application.properties  
    application-dev.properties  
pom.xml
```

4.2 Frontend Structure (React + TypeScript)

```
pos-frontend/  
src/  
    App.tsx  
    index.tsx  
    components/  
        Login.tsx  
        Dashboard.tsx  
        SaleForm.tsx  
        RentalForm.tsx  
        ReturnForm.tsx  
        InventoryList.tsx  
    pages/  
        CashierDashboard.tsx  
        AdminDashboard.tsx  
        SalesPage.tsx  
        RentalsPage.tsx  
        ReturnsPage.tsx  
    services/  
        api.ts  
        authService.ts  
        saleService.ts  
        rentalService.ts  
        inventoryService.ts  
    hooks/  
        useAuth.ts  
        useInventory.ts  
types/  
    Employee.ts  
    Item.ts  
    Sale.ts  
    Rental.ts
```

```
utils/  
constants.ts  
formatters.ts  
package.json  
tsconfig.json
```

5 Key Features Implementation

- **Authentication:** JWT + BCrypt + Role-based access (Admin/Cashier)
- **Sales:** Real-time cart, tax, coupons, receipt generation
- **Rentals:** Customer lookup, due dates, overdue tracking
- **Returns:** Outstanding rental lookup, refund calculation
- **Inventory:** Real-time stock, low-stock alerts
- **Employee Management:** CRUD (Admin only)

6 API Design

6.1 RESTful Endpoints Summary

Method	Endpoint
POST	/api/auth/login
GET	/api/auth/me
POST	/api/sales
GET	/api/sales?employeeId=&startDate=&endDate=
POST	/api/rentals
GET	/api/rentals/customer/{phone}
GET	/api/rentals/outstanding/{phone}
POST	/api/returns
GET	/api/items
GET	/api/items/low-stock
GET	/api/customers/phone/{phone}
GET	/api/employees (Admin)
POST	/api/employees (Admin)

7 Design Patterns Applied

- Repository Pattern
- Service Layer Pattern
- DTO Pattern
- Dependency Injection (Spring IoC)
- Singleton (Spring beans)

- Strategy Pattern (future payment methods)

8 Testing Strategy

- **Backend:** JUnit 5, @SpringBootTest, MockMvc, @DataJpaTest
- **Frontend:** Jest + React Testing Library, MSW for API mocking
- **E2E (optional):** Cypress

9 Deployment Strategy

- **Development:** Local Spring Boot (8080) + React dev server (3000)
- **Production:**
 - Backend: Dockerized JAR on AWS/Heroku
 - Frontend: Static build on Netlify/Vercel
 - Database: Managed PostgreSQL (RDS, Supabase, etc.)

10 Migration from Legacy System

1. Run data migration scripts (from .txt files to PostgreSQL)
2. Validate all records migrated correctly
3. Parallel run with legacy system during transition
4. Gradual user migration
5. Full decommissioning of legacy desktop app

11 Improvements Over Legacy System

Legacy System	New Web System
File-based .txt storage	Normalized PostgreSQL database
Plain-text passwords	BCrypt + JWT authentication
Single-user desktop app	Multi-user web access
No concurrency control	ACID transactions
Hardcoded paths	Configurable via properties
Swing UI (1990s)	Modern React + Material-UI
No testing	Full unit/integration/E2E tests
Not scalable	Cloud-ready, horizontally scalable

Document Version: 1.0
Date: 2025-11-28
Status: Ready for Implementation