Software Design Specifications

for

Pharmacy Inventory Management System 1.0

Prepared by:

Rishitha Janga - Project Manager
Jai Aditya - Database Admin
Anisha - frontend
Soumith - QA and tester
Sreeyansh - frontend
Raunak - backend
Manas
Sameer - backend

Document Information

Title: Software Design Specification - Pharmacy Inventory Management System	
Project Manager: Swapna S	**Document Version No: 1.0**
	Document Version Date: April 4th, 2025
Prepared By: Rishitha, Jai Aditya, Sameer, Raunak, Soumith, Sreeyansh, Manas, Anisha	**Preparation Date: April 4th, 2025**

Version History

Ver. No.	Ver. Date	Revised By	Description	Filename
1.0	04/04/25	Rishitha	Initial Draft	Pharmacy_SDS

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this Software Design Specification (SDS) document is to outline the software architecture and design decisions for the Pharmacy Inventory Management System. It serves as a blueprint for developers and stakeholders to understand system structure, modules, and interactions. The document is primarily intended for developers, testers, and project stakeholders.

1.2 Scope

This system automates pharmacy stock management, tracks expiry dates, manages sales, and facilitates monthly medicine delivery subscriptions based on uploaded prescriptions. It includes features for Admin and Pharmacist dashboards, patient data management, sales tracking, and integration with payment gateways (PhonePe, Paytm, GPay, Razorpay).

1.3 Definitions, Acronyms, and Abbreviations

- **SDS**: Software Design Specification
- **UI**: User Interface
- **API**: Application Programming Interface
- **POS**: Point of Sale
- **OTP**: One-Time Password
- **DB**: Database
- **Rx**: Prescription

1.4 References

- Pharmacy Inventory Management System Requirements Document
- ReactJS, Tailwind CSS Documentation
- Node.js and Express.js Documentation
- MongoDB Documentation

2 Use Case View

2.1 Use Case

Main Use Cases:

- **Admin Login and Dashboard** – manage stock, pharmacists, sales reports
- **Pharmacist Login and POS** – manage daily sales, generate bills
- **Customer Subscription** – upload prescriptions and receive monthly medicine kits
- **Inventory Management** – stock in/out, expiry tracking
- **Reports** – sales, stock, expiry, transactions

3. Design Overview

3.1 Design Goals and Constraints

- Responsive web app
- Scalable backend (Node.js + MongoDB)
- Easy access and authentication
- Secure payment gateway integration
- Intuitive UI using Tailwind CSS

3.2 Design Assumptions

- Admin will handle medicine data entry
- Pharmacists will handle billing
- Customers can access subscription service via frontend

3.3 Significant Design Packages

AdminPanel, PharmacistPanel, SubscriptionModule, InventoryModule, PaymentGatewayModule, DatabaseConfig

3.4 Dependent External Interfaces

The table below lists the public interfaces this design requires from other modules or applications.

External Application and Interface Name	**Module Using the Interface**	**Functionality/ Description**
Razorpay, Paytm, GPay, PhonePe	Payment API	Handles secure payment processing

3.5 Implemented Application External Interfaces (and SOA web services)

The table below lists the implementation of public interfaces this design makes available for other applications.

Interface Name	**Module Implementing the Interface**	**Functionality/ Description**
GET/api/ medicines	Inventory	Fetch medicine data
POSt/api/ prescription	Subscription	Upload Prescription
POST/api/pay	Payment	Payment Integration

4 Logical View

4.1 Design Model

Includes React components for the frontend and modular Express controllers for backend:

- React: Login.jsx, Dashboard.jsx, NewMedicine.jsx, etc.
- Express: medicineController.js, subscriptionController.js, etc.

4.2 Use Case Realization

Use Case 1: Admin Login and Dashboard Access

High-Level Interaction (Sequence Diagram Summary)

1. Admin enters credentials on the login page (Login.jsx)
2. React frontend sends credentials via POST /api/admin/login
3. Backend (authController.js) verifies login with MongoDB
4. On success, JWT is returned and stored
5. Admin redirected to Dashboard.jsx, which loads data using APIs

Lower-Level Collaboration

- Login.jsx → triggers authController.loginAdmin()
- authController.js → calls User.findOne({ role: 'admin' })
- JWT is signed and sent back
- Dashboard.jsx loads modules: stock, sales reports, etc.

Use Case 2: Pharmacist POS & Billing

High-Level Interaction

1. Pharmacist logs in via PharmacistLogin.jsx
2. Redirected to PharmacistPOS.jsx
3. Selects medicine and quantity
4. Calls POST /api/sales
5. Backend reduces stock, generates bill, and returns invoice
6. Invoice displayed and optionally downloaded

Lower-Level Collaboration

- PharmacistPOS.jsx → saleController.createTransaction()
- saleController.js → checks stock → updates quantity
- Invoice is generated using invoiceController.js

Use Case 3: Customer Subscription Process

High-Level Interaction

1. Customer signs up → logs in
2. Navigates to subscription page → uploads prescription
3. Calls POST /api/prescriptions
4. Backend stores file, associates with user ID
5. Scheduler sets monthly delivery reminder
6. Admin dashboard shows pending dispatches

Lower-Level Collaboration

- SubscriptionForm.jsx → triggers subscriptionController.uploadRx()
- Rx stored in /uploads folder with MongoDB reference
- dispatchJob.js checks subscription DB monthly
- Admin notified via dashboard

Use Case 4: Inventory Management

High-Level Interaction

1. Admin navigates to NewMedicine.jsx
2. Fills form and submits → POST /api/medicines
3. Backend creates entry in MongoDB
4. Admin views list in PharmacistInventory.jsx

Lower-Level Collaboration

- NewMedicine.jsx → calls medicineController.addMedicine()
- Checks for duplicates → adds with expiry, price, quantity
- Inventory module updates in real-time

Use Case 5: Reports Generation

High-Level Interaction

1. Admin opens Reports.jsx
2. Selects report type (sales, expiry, stock, transaction)
3. Sends API request like GET /api/reports/sales
4. Backend aggregates MongoDB data
5. Returns formatted report

Lower-Level Collaboration

- Reports.jsx → reportController.generateReport()
- Controller runs filters, aggregates, and returns JSON
- Data is displayed in chart/table form

5 Data View

5.1 Domain Model

Entities: User, Medicine, Prescription, Transaction, Report

5.2 Data Model (persistent data view)

5.2.1 Data Dictionary

Field	Type	Description
`medicineName`	String	Name of medicine
`expiryDate`	Date	Expiry date
`quantity`	Number	Available quantity
`userType`	String	Admin or Pharmacist
`subscription`	Boolean	Whether user is subscribed

6 Exception Handling

- Invalid Login → Error: 401 Unauthorised
- Out-of-Stock Medicine → Message: Medicine Unavailable
- Payment Failure → Display appropriate error message
- DB connection error → Retry mechanism enabled

7 Configurable Parameters

This table describes the simple configurable parameters (name / value pairs).

Parameter	Description	Dynamic
`PORT`	Server port number	Yes
`MONGODB_URI`	MongoDB connection string	Yes
`JWT_SECRET`	Secret key for auth	No
`PAYMENT_API_KEY`	Razorpay/PhonePe key	No

8 Quality of Service

8.1 Availability

- 99.9% uptime targeted
- Downtime only during scheduled maintenance

8.2 Security and Authorisation

- JWT-based authentication
- Role-based access for Admin and Pharmacist
- Secure payment using gateway APIs

8.3 Load and Performance Implications

- Supports up to 1000 transactions/day
- MongoDB optimised with indexing
- API request throttling in place

8.4 Monitoring and Control

- Admin panel shows stock alerts
- Expiry warnings displayed in red
- Server logs maintained for every transaction