# Design Specification Document

# Inventory and Billing Management System for General Store

## MERN Stack with Electron.js Implementation

**Prepared by:** Faiz Ullah Khan  
**Document Date:** 2025-10-13  
**Version:** 1.0

## Table of Contents

1. [Introduction](#Xe3d0fc0bea9a42ce7605565d0964033d7f6ee47)
   * 1.1 [Purpose](#Xe94c4103db6bb764aa3b798144b782dac27f424)
   * 1.2 [Scope](#X3378fde8998068a01e48267892e5cd923e9bc15)
   * 1.3 [References](#X8241e2c2f4870e183c18f4839d5b00887fbb26d)
2. [System Architecture](#Xadffa0c657789d188b7bad67251e99502d9d94d)
   * 2.1 [Architecture Overview](#Xe4af4147dc8f5cc65dd6ef32128ba32f5bdd3f5)
   * 2.2 [Component Diagram](#X2ee7709689d2642d5b76facf18762284e6bf546)
   * 2.3 [Deployment Model](#X80c4008e5e12e099ae31bcc06de1c28d2ec8607)
3. [Database Design](#X207c42dc9699dbfa2d40baada8c9a55d7a8e12f)
   * 3.1 [Database Management System](#X27933627512136a6fe9d382d46098a66a3fa6fa)
   * 3.2 [Data Models](#X4d22979c65beb20f20ef650ae7ee31458e3db37)
   * 3.3 [Schema Definitions](#Xa47d2066a23f94a8fd1dac11b671f86d22b2aa3)
   * 3.4 [Data Dictionary](#Xa3bf5026faf256edf63a43a8f8d51ee33fbc70e)
   * 3.5 [Database Optimization Strategy](#X10cbf623a50007c1a971962d0c423ee856f338d)
4. [User Interface Design](#X50654218cd8c97b63cc440d25cc759505961bd8)
   * 4.1 [Design Principles](#X9de8e5b45ce68f346b1cc38c139307b75b6baf5)
   * 4.2 [Screen Layout Templates](#X2cdde27ddb4f140a9c35de9d2561a30b674a7f0)
   * 4.3 [UI Component Library](#Xc86bd83ae38fbf23f6c56650be34bca50bca426)
   * 4.4 [Navigation Map](#X24eae82529d8854ba3f52092179627dfe3b3a83)
   * 4.5 [Screen Mockups](#Xbac23761218df4001ea4e27a392958c1c55dd8f)
5. [Module Design](#Xf1c5de4ef166a2e585c6d43a6ae34496e5e09cf)
   * 5.1 [Product Management Module](#X2f254005997bb0a8f0b3a66b6783f9bb92a6381)
   * 5.2 [Billing Module](#X7e1ebddb70e7727bce1cd57427949a34505d460)
   * 5.3 [Inventory Management Module](#Xa60969fd2cd3de9b5e4d2abddfa3a3dc2189c2f)
   * 5.4 [Reporting Module](#Xebf122658c78e7458fa4bb6c7aeb5d5b45ca286)
   * 5.5 [Supplier Management Module](#X0c570317bd6f17efc6b9374e53439a5fd324994)
   * 5.6 [User Authentication Module](#Xe9d653b35fb4caa83e27fe7a9ae0b9b2e28d0ae)
   * 5.7 [Backup/Restore Module](#Xfc515074df593a8cfe79ad8ce12aae359e3629e)
6. [Process Flows](#Xff05e0494e870022adbf43d834460be2fd5a492)
   * 6.1 [Billing Process](#Xf648dc69de968c372ad6f3221a0b44a67bcbfaa)
   * 6.2 [Inventory Update Process](#Xa8e7816e9d166fd6f4c1b5b7a5a17d6cfc4858a)
   * 6.3 [Reporting Process](#Xa08c9aae773227b5dd7d0c0d8776a9ab6d7db3f)
   * 6.4 [Backup Process](#Xe640ed74f22067c32361fbfc728cae1ae1501f3)
7. [Data Flow Diagrams](#X5bf9f1a22057d220651d1414e50127595ef0223)
   * 7.1 [Context Diagram](#X736e5a3d145b8169cd080fd5da588de68fe48e0)
   * 7.2 [Level 1 DFD](#X5a64faa108082812c3418ae261da3ed03ebaf33)
8. [Security Design](#X66f08edac5b0aa578e1635ff0ec4335c9892d6d)
   * 8.1 [Authentication Mechanism](#X68b024a256de1d002a26aa6a19af43ab11ffa97)
   * 8.2 [Data Protection](#X0fa53614a5bad9b8f678a4df33102c7002a7f95)
   * 8.3 [Audit Trail](#X3926d672c2100c40b5eaa3abbf6a04886b91bbe)
9. [Exception Handling](#Xdb013bb3339da315a74e25ead5a2218d3f8efaa)
   * 9.1 [Error Categories](#Xe6c18924ae67cb73efcffc9cbc19fda077c6334)
   * 9.2 [Error Messages](#Xf7df65163237d906aa8529838c8ba7fe40ed127)
   * 9.3 [Recovery Procedures](#X26ccb81a79ff534a549b52da9e5608019aa71aa)
10. [Testing Strategy](#X8cb50c8fa5db4e12befc846d9fbeb1d96bd9540)
    * 10.1 [Unit Testing](#Xd0765f71e3c344bbbbf70a7c72e4052aeb692f9)
    * 10.2 [Integration Testing](#X09732d93160514d246256c49b08320dcfb5f2ff)
    * 10.3 [System Testing](#Xe6ecfbeff2f3eff329f64496b79a4d2791a7617)
    * 10.4 [User Acceptance Testing](#Xb2d14832af0fc2de11070ad66ca18a6e35c0fa0)
11. [Implementation Plan](#Xa337785b92f16cf0c229f5fc23410f8e8a7e670)
    * 11.1 [Development Environment](#X421147532b039c74a036bc7b28488b2d4a54986)
    * 11.2 [Technology Stack](#X01de649acd1b5a419bc9d99fe8157f899a3d817)
    * 11.3 [Development Phases](#Xe6aaabd6306f81f0102588f38cf351c7b14a624)
    * 11.4 [Deployment Strategy](#X76c59a76e674215e0e1b716a5c0ecbd76b4c954)
12. [Appendices](#X30bfd1d3686d43b794f9c4bf827135f99f553e6)
    * 12.1 [Technical Libraries and Frameworks](#X518f40c5434e126148aaef7d84be941ce6c032e)
    * 12.2 [API Specifications](#Xd57156c244a238332d8dc54053b5a643b7fe90d)

## 1. Introduction

### 1.1 Purpose

This Design Specification Document provides the detailed technical design for the Inventory and Billing Management System as outlined in the Software Requirements Specification (SRS). It serves as a blueprint for developers to implement the system using the MERN stack (MongoDB, Express.js, React.js, Node.js) with Electron.js for desktop application capabilities.

### 1.2 Scope

This document covers the complete design of all system components including:

* System architecture and component relationships
* MongoDB database design
* React-based user interfaces
* Node.js backend services
* Electron.js desktop integration
* Module specifications and process flows
* Security implementation and testing approaches

The design focuses on a cross-platform desktop application using web technologies, designed for offline operation in a single-user environment.

### 1.3 References

* Software Requirements Specification for Inventory and Billing Management System v1.0
* Electron.js Application Development Guidelines
* MERN Stack Development Best Practices
* MongoDB Schema Design Patterns

## 2. System Architecture

### 2.1 Architecture Overview

The system will be built using a modified MERN stack architecture with Electron.js:

1. **Frontend Layer (Presentation)**:
   * React.js components for the user interface
   * Redux for state management
   * Material-UI and Tailwind CSS for component library
   * Electron.js for desktop application wrapper
2. **Backend Layer (Business Logic)**:
   * Node.js with Express.js for API services
   * Local API server running within Electron process
   * Service modules for core business logic
3. **Data Layer**:
   * MongoDB for NoSQL document storage
   * Mongoose for object data modeling
   * Local database instance embedded in the application

This architecture allows for:

* Offline operation with local database
* Rich user interface with React components
* Separation of concerns with clear layer boundaries
* Cross-platform compatibility (Windows, macOS, Linux)

### 2.2 Component Diagram

┌─────────────────────────────────────────────────────────────┐  
│ Electron.js Application │  
│ │  
│ ┌───────────────────────────────────────────────────────┐ │  
│ │ React.js Frontend (Renderer) │ │  
│ │ ┌─────────────┐ ┌─────────────┐ ┌─────────────────┐ │ │  
│ │ │ Dashboard │ │ Billing │ │Product Management│ │ │  
│ │ │ Component │ │ Component │ │ Component │ │ │  
│ │ └─────────────┘ └─────────────┘ └─────────────────┘ │ │  
│ │ ┌─────────────┐ ┌─────────────┐ ┌─────────────────┐ │ │  
│ │ │ Inventory │ │ Reports │ │ Settings │ │ │  
│ │ │ Component │ │ Component │ │ Component │ │ │  
│ │ └─────────────┘ └─────────────┘ └─────────────────┘ │ │  
│ └───────────────────────┬───────────────────────────────┘ │  
│ │ Redux/Context API │  
│ ▼ │  
│ ┌───────────────────────────────────────────────────────┐ │  
│ │ Node.js Backend (Main Process) │ │  
│ │ ┌─────────────┐ ┌─────────────┐ ┌─────────────────┐ │ │  
│ │ │ Product │ │ Billing │ │Inventory Service │ │ │  
│ │ │ Service │ │ Service │ │ │ │ │  
│ │ └─────────────┘ └─────────────┘ └─────────────────┘ │ │  
│ │ ┌─────────────┐ ┌─────────────┐ ┌─────────────────┐ │ │  
│ │ │ Reporting │ │ Supplier │ │ Backup Service │ │ │  
│ │ │ Service │ │ Service │ │ │ │ │  
│ │ └─────────────┘ └─────────────┘ └─────────────────┘ │ │  
│ └───────────────────────┬───────────────────────────────┘ │  
│ │ Mongoose ODM │  
│ ▼ │  
│ ┌───────────────────────────────────────────────────────┐ │  
│ │ MongoDB Database │ │  
│ │ (Embedded local instance or connection to local DB) │ │  
│ └───────────────────────────────────────────────────────┘ │  
└─────────────────────────────────────────────────────────────┘

### 2.3 Deployment Model

The application will be deployed as a standalone Electron desktop application:

* **Installation**: Electron packaged installer for Windows, macOS, and Linux
* **Updates**: Auto-updater module for version control and updates
* **Local File Storage**: Application data stored in structured folders:
  + Database files in %AppData%\StoreManager\Database\ (Windows) or equivalent on other platforms
  + Configuration in user data directory
  + Backups in user-selected location or default backup directory
  + Logs in application-specific log directory

## 3. Database Design

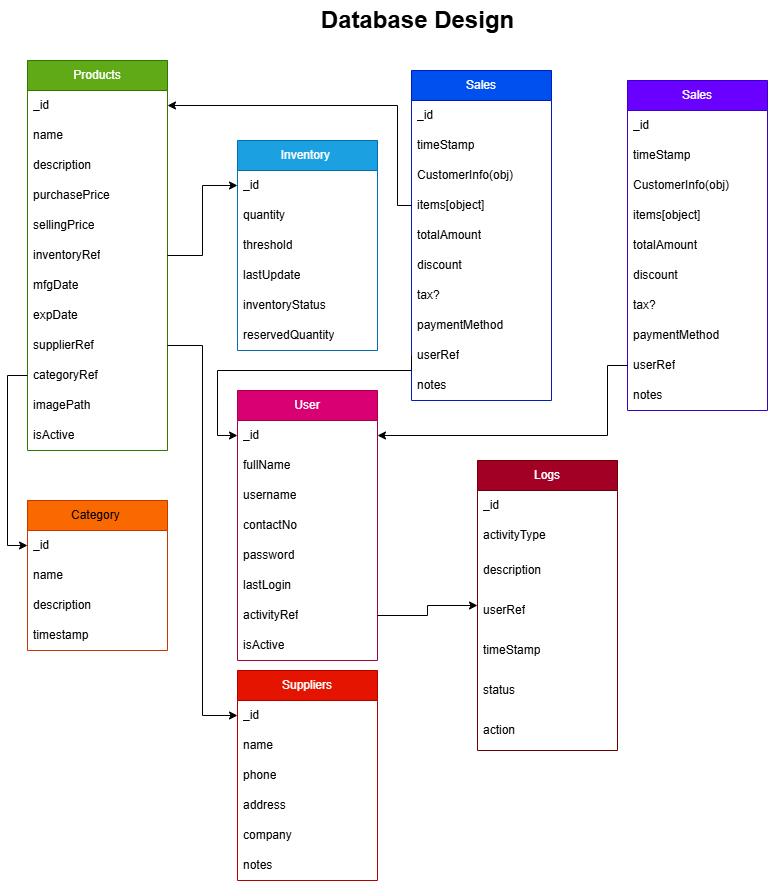
### 3.1 Database Management System

The system will use MongoDB as the NoSQL document database due to:

* Schema flexibility for evolving business requirements
* JSON-like document storage that maps well to JavaScript objects
* Easy integration with Node.js via Mongoose
* Embedded MongoDB capability for offline operation
* Simple backup and restore operations
* Good performance for the expected data volume

### 3.2 Data Models

The system will use the following MongoDB collections (equivalent to tables in relational databases):



### 3.3 Schema Definitions

#### Products Schema

const ProductSchema = new mongoose.Schema({  
 name: { type: String, required: true },  
 description: { type: String },  
 purchasePrice: { type: Number, required: true },  
 sellingPrice: { type: Number, required: true },  
 quantity: { type: Number, default: 0 },  
 unit: { type: String },  
 reorderLevel: { type: Number },  
 mfgDate: { type: Date },  
 expDate: { type: Date },  
 batchNumber: { type: String },  
 supplier: { type: mongoose.Schema.Types.ObjectId, ref: 'Supplier' },  
 categories: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Category' }],  
 imagePath: { type: String },  
 isActive: { type: Boolean, default: true },  
}, { timestamps: true });

#### Categories Schema

const CategorySchema = new mongoose.Schema({  
 name: { type: String, required: true },  
 description: { type: String },  
}, { timestamps: true });

#### Suppliers Schema

const SupplierSchema = new mongoose.Schema({  
 name: { type: String, required: true },  
 contactPerson: { type: String },  
 phone: { type: String },  
 email: { type: String },  
 address: { type: String },  
 notes: { type: String },  
 isActive: { type: Boolean, default: true },  
}, { timestamps: true });

#### Users Schema

const UserSchema = new mongoose.Schema({  
 username: { type: String, required: true, unique: true },  
 passwordHash: { type: String, required: true },  
 lastLogin: { type: Date },  
 isActive: { type: Boolean, default: true },  
}, { timestamps: true });

#### Sales Schema

const SaleItemSchema = new mongoose.Schema({  
 product: { type: mongoose.Schema.Types.ObjectId, ref: 'Product', required: true },  
 quantity: { type: Number, required: true },  
 unitPrice: { type: Number, required: true },  
 totalPrice: { type: Number, required: true },  
 profit: { type: Number },  
});  
  
const SaleSchema = new mongoose.Schema({  
 dateTime: { type: Date, default: Date.now },  
 customer: { type: String },  
 items: [SaleItemSchema],  
 totalAmount: { type: Number, required: true },  
 discount: { type: Number, default: 0 },  
 tax: { type: Number, default: 0 },  
 netAmount: { type: Number, required: true },  
 paymentMethod: { type: String, default: 'Cash' },  
 user: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },  
 notes: { type: String },  
}, { timestamps: true });

#### Inventory\_Log Schema

const InventoryLogSchema = new mongoose.Schema({  
 product: { type: mongoose.Schema.Types.ObjectId, ref: 'Product', required: true },  
 dateTime: { type: Date, default: Date.now },  
 oldQuantity: { type: Number, required: true },  
 newQuantity: { type: Number, required: true },  
 reason: { type: String },  
 referenceId: { type: mongoose.Schema.Types.ObjectId },  
 referenceType: { type: String }, // 'Sale', 'Adjustment', etc.  
}, { timestamps: true });

#### Inventory\_Adjust Schema

const InventoryAdjustSchema = new mongoose.Schema({  
 product: { type: mongoose.Schema.Types.ObjectId, ref: 'Product', required: true },  
 dateTime: { type: Date, default: Date.now },  
 quantityChange: { type: Number, required: true },  
 reason: { type: String },  
 user: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },  
}, { timestamps: true });

### 3.4 Data Dictionary

| Collection | Field | Data Type | Description |
| --- | --- | --- | --- |
| Products | \_id | ObjectId | Unique identifier for products |
| Products | name | String | Name of the product |
| Products | description | String | Detailed description of the product |
| Products | purchasePrice | Number | Cost price of the product |
| Products | sellingPrice | Number | Retail price of the product |
| Products | quantity | Number | Current stock level |
| Products | unit | String | Unit of measurement (e.g., kg, piece) |
| Products | reorderLevel | Number | Minimum quantity before reorder alert |
| Products | mfgDate | Date | Manufacturing date |
| Products | expDate | Date | Expiration date |
| Products | batchNumber | String | Manufacturer batch number |
| Products | supplier | ObjectId | Reference to supplier document |
| Products | categories | [ObjectId] | Array of references to category documents |
| Products | imagePath | String | Path to product image file |
| Products | isActive | Boolean | Whether product is active in system |
| Products | createdAt | Date | Record creation timestamp |
| Products | updatedAt | Date | Record last update timestamp |

(Additional data dictionary entries would follow for all collections and fields)

### 3.5 Database Optimization Strategy

1. **Indexing Strategy**:
   * Create indexes on frequently queried fields:
   * // Products collection indexes  
     ProductSchema.index({ name: 'text' }); // Text search on product name  
     ProductSchema.index({ supplier: 1 }); // Lookup by supplier  
     ProductSchema.index({ categories: 1 }); // Lookup by category  
       
     // Sales collection indexes  
     SaleSchema.index({ dateTime: -1 }); // Sort by date descending  
     SaleSchema.index({ 'items.product': 1 }); // Lookup by product in items
2. **Query Optimization**:
   * Use appropriate Mongoose projection and population
   * Implement MongoDB aggregation pipeline for complex queries
   * Batch operations for bulk updates/inserts
   * Implement pagination for large result sets
3. **Data Caching**:
   * In-memory cache for frequently accessed data
   * Local storage caching for UI state persistence
   * Application-level caching of lookup data
4. **Database Maintenance**:
   * Periodic compaction of database files
   * Regular validation of document integrity
   * Automated backup scheduling

## 4. User Interface Design

### 4.1 Design Principles

1. **Simplicity First**
   * Clear, uncluttered layouts
   * Focus on essential information
   * Progressive disclosure of complex features
2. **Consistency**
   * Uniform control placement across all screens
   * Consistent color scheme and typography
   * Standard button sizes and placements
3. **Visual Hierarchy**
   * Important elements have visual prominence
   * Related items are visually grouped
   * Clear section separation with appropriate spacing
4. **Error Prevention**
   * Validate input as it's entered
   * Provide clear feedback on actions
   * Confirm destructive operations
5. **Accessibility**
   * High contrast between text and background
   * Readable font sizes (minimum 16px)
   * ARIA attributes for screen reader compatibility

### 4.2 Screen Layout Templates

Each screen in the application will follow one of these layout templates:

1. **Dashboard Layout**

┌─────────────────────────────────────────────────────────┐  
│ Logo User | Logout │  
├─────────────┬───────────────────────────────────────────┤  
│ │ │  
│ │ │  
│ Navigation │ Content Area │  
│ Menu │ │  
│ │ │  
│ │ │  
├─────────────┴───────────────────────────────────────────┤  
│ Status Bar │  
└─────────────────────────────────────────────────────────┘

1. **Data Entry Layout**

┌─────────────────────────────────────────────────────────┐  
│ Logo User | Logout │  
├─────────────┬───────────────────────────────────────────┤  
│ │ Form Title Actions │  
│ ├───────────────────────────────────────────┤  
│ Navigation │ │  
│ Menu │ Form Fields │  
│ │ │  
│ │ │  
│ ├───────────────────────────────────────────┤  
│ │ Action Buttons │  
├─────────────┴───────────────────────────────────────────┤  
│ Status Bar │  
└─────────────────────────────────────────────────────────┘

1. **List/Grid Layout**

┌─────────────────────────────────────────────────────────┐  
│ Logo User | Logout │  
├─────────────┬───────────────────────────────────────────┤  
│ │ List Title Actions │  
│ ├───────────────────────────────────────────┤  
│ Navigation │ Search/Filter Controls │  
│ Menu ├───────────────────────────────────────────┤  
│ │ │  
│ │ Data Grid / List View │  
│ │ │  
│ ├───────────────────────────────────────────┤  
│ │ Pagination Summary │  
├─────────────┴───────────────────────────────────────────┤  
│ Status Bar │  
└─────────────────────────────────────────────────────────┘

### 4.3 UI Component Library

The application will use Material-UI as the primary component library:

1. **Input Controls**
   * TextField: For single-line text input
   * NumberField: For numeric input with validation
   * DatePicker: Calendar-style date selection
   * Select/Autocomplete: For selection from predefined options
   * SearchField: Text input with search functionality
2. **Information Display**
   * DataGrid: For tabular data display with sorting and filtering
   * Card: For compact information display
   * Charts: Using React-ChartJS for visual data representation
   * Badges/Chips: To show status information
3. **Navigation**
   * Drawer: Side navigation menu
   * Tabs: Secondary navigation within a section
   * Breadcrumbs: For hierarchical navigation
   * Buttons: Various button types with consistent styling
4. **Notifications**
   * Alert: For important messages
   * Snackbar: For temporary feedback
   * Dialog: For confirmations and user decisions
   * LinearProgress/CircularProgress: For long-running operations

### 4.4 Navigation Map

┌───────────────┐  
│ Dashboard │  
└───────┬───────┘  
 │  
┌───────┴───────┬────────────┬─────────────┬─────────────┬────────────┐  
│ │ │ │ │ │  
▼ ▼ ▼ ▼ ▼ ▼  
┌───────────┐ ┌────────┐ ┌─────────┐ ┌──────────┐ ┌──────────┐ ┌──────────┐  
│ Products │ │ Billing│ │Inventory│ │ Reports │ │Suppliers │ │ Settings │  
└─────┬─────┘ └───┬────┘ └────┬────┘ └────┬─────┘ └────┬─────┘ └────┬─────┘  
 │ │ │ │ │ │  
┌─────┴─────┐ ┌──┴────┐ ┌────┴────┐ ┌────┴─────┐ ┌────┴─────┐ ┌────┴─────┐  
│Add Product│ │New Bill│ │ Stock │ │ Sales │ │ Add │ │ User │  
│Edit Product│ │Bill List│ │ Adjust │ │ Reports │ │ Supplier │ │ Profile │  
│List Products│└────────┘ │Low Stock│ │Inventory │ │Supplier │ │ Backup │  
└───────────┘ │ History│ │ Reports │ │ List │ │ Restore │  
 └─────────┘ │Profit/Loss│└──────────┘ └──────────┘  
 └──────────┘

### 4.5 Screen Mockups

(Detailed mockups for key screens would be included here)

1. **Dashboard Screen**
2. **Product Management Screen**
3. **Billing Screen**
4. **Inventory Management Screen**
5. **Reports Screen**

## 5. Module Design

### 5.1 Product Management Module

**Purpose**: Manage all aspects of product data and categorization.

**Key Components**:

* ProductService: Backend service for product operations
* ProductModel: Mongoose model for product data
* ProductContext: React context for product state management
* ProductForm: React component for adding/editing products
* ProductList: React component for displaying products

**Main Functions**:

1. addProduct(product): Add new product to inventory
2. updateProduct(id, product): Update existing product details
3. deleteProduct(id): Mark product as inactive
4. getProductById(id): Retrieve specific product
5. searchProducts(query): Search products by name/description
6. getLowStockProducts(): Get products below reorder level
7. assignProductCategory(productId, categoryId): Categorize a product

**Events/Actions**:

* PRODUCT\_ADDED: Dispatched when new product is added
* PRODUCT\_UPDATED: Dispatched when product details change
* PRODUCT\_DELETED: Dispatched when product is deleted
* LOW\_STOCK\_ALERT: Dispatched when product falls below reorder level

### 5.2 Billing Module

**Purpose**: Handle all sales transactions and billing operations.

**Key Components**:

* BillingService: Backend service for sales operations
* SaleModel: Mongoose model for sales data
* BillingContext: React context for billing state management
* BillingForm: React component for creating bills
* SalesList: React component for listing sales

**Main Functions**:

1. createNewSale(): Initialize a new sale transaction
2. addItemToSale(saleId, productId, quantity): Add product to current sale
3. removeItemFromSale(saleId, itemId): Remove item from current sale
4. updateItemQuantity(saleId, itemId, quantity): Change item quantity
5. applyDiscount(saleId, discountAmount): Apply discount to sale
6. calculateTotals(saleId): Calculate subtotal, tax, and final amount
7. completeSale(saleId, paymentMethod): Finalize sale and process inventory
8. printReceipt(saleId): Generate printable receipt

**Events/Actions**:

* SALE\_CREATED: Dispatched when new sale is created
* SALE\_UPDATED: Dispatched when items are added/removed
* SALE\_COMPLETED: Dispatched when sale is finalized
* INVENTORY\_CHANGED: Dispatched when sale affects inventory levels

### 5.3 Inventory Management Module

**Purpose**: Track and manage product inventory levels.

**Key Components**:

* InventoryService: Backend service for inventory operations
* InventoryLogModel: Mongoose model for inventory history
* InventoryAdjustModel: Mongoose model for inventory adjustments
* InventoryContext: React context for inventory state management
* InventoryForm: React component for inventory adjustments

**Main Functions**:

1. adjustInventory(productId, quantityChange, reason): Manual inventory adjustment
2. getInventoryHistory(productId): Retrieve inventory history for product
3. logInventoryChange(productId, oldQty, newQty, reason): Record changes
4. checkLowStockItems(): Identify items below reorder level
5. getExpiringItems(daysThreshold): Find items nearing expiration
6. reconcileInventory(actualCounts): Update inventory after physical count

**Events/Actions**:

* INVENTORY\_ADJUSTED: Dispatched when manual adjustment occurs
* LOW\_STOCK\_DETECTED: Dispatched when item falls below threshold
* EXPIRY\_APPROACHING: Dispatched when items are nearing expiry date

### 5.4 Reporting Module

**Purpose**: Generate business reports and analytics.

**Key Components**:

* ReportingService: Backend service for report generation
* ReportContext: React context for report state management
* ReportFilters: React component for report parameter selection
* ReportViewer: React component for displaying generated reports
* ChartComponents: React components for visual data representation

**Main Functions**:

1. generateSalesReport(startDate, endDate): Create sales report
2. generateProfitReport(startDate, endDate): Create profit/loss report
3. generateInventoryReport(): Create current inventory status report
4. generateProductPerformanceReport(): Analyze product sales performance
5. exportReportToPdf(report): Export report to PDF format
6. exportReportToExcel(report): Export report to Excel format

**Events/Actions**:

* REPORT\_GENERATED: Dispatched when report creation completes
* EXPORT\_COMPLETED: Dispatched when report export finishes

### 5.5 Supplier Management Module

**Purpose**: Manage supplier information and relationships.

**Key Components**:

* SupplierService: Backend service for supplier operations
* SupplierModel: Mongoose model for supplier data
* SupplierContext: React context for supplier state management
* SupplierForm: React component for adding/editing suppliers
* SupplierList: React component for displaying suppliers

**Main Functions**:

1. addSupplier(supplier): Add new supplier
2. updateSupplier(id, supplier): Update supplier details
3. deleteSupplier(id): Mark supplier as inactive
4. getSupplierById(id): Retrieve specific supplier
5. getProductsBySupplier(supplierId): List products from specific supplier
6. searchSuppliers(query): Search suppliers by name/contact

**Events/Actions**:

* SUPPLIER\_ADDED: Dispatched when new supplier is added
* SUPPLIER\_UPDATED: Dispatched when supplier details change
* SUPPLIER\_DELETED: Dispatched when supplier is deleted

### 5.6 User Authentication Module

**Purpose**: Handle user authentication and security.

**Key Components**:

* AuthService: Backend service for authentication
* UserModel: Mongoose model for user data
* AuthContext: React context for authentication state
* LoginForm: React component for user login
* ProfileSettings: React component for user profile management

**Main Functions**:

1. authenticateUser(username, password): Validate user credentials
2. changePassword(userId, oldPassword, newPassword): Update password
3. getCurrentUser(): Get currently logged-in user
4. logUserActivity(userId, activity): Track user actions
5. logoutUser(): End user session

**Events/Actions**:

* USER\_LOGGED\_IN: Dispatched on successful authentication
* USER\_LOGGED\_OUT: Dispatched when user logs out
* PASSWORD\_CHANGED: Dispatched when password is updated
* AUTHENTICATION\_FAILED: Dispatched on failed login attempt

### 5.7 Backup/Restore Module

**Purpose**: Manage system data backup and restoration.

**Key Components**:

* BackupService: Backend service for backup operations
* BackupContext: React context for backup state management
* BackupForm: React component for backup configuration
* RestoreForm: React component for restore operations

**Main Functions**:

1. createBackup(destination): Create database backup
2. restoreFromBackup(backupFile): Restore system from backup
3. scheduleAutomaticBackup(interval): Configure scheduled backups
4. getBackupHistory(): List previous backup operations
5. validateBackupFile(backupFile): Check backup file integrity

**Events/Actions**:

* BACKUP\_STARTED: Dispatched when backup begins
* BACKUP\_COMPLETED: Dispatched when backup finishes
* RESTORE\_STARTED: Dispatched when restore begins
* RESTORE\_COMPLETED: Dispatched when restore finishes

## 6. Process Flows

### 6.1 Billing Process

┌─────────┐ ┌──────────────┐ ┌───────────────┐  
│ Start │────►│ Create New │────►│ Add Items to │  
└─────────┘ │ Sale │ │ Sale │  
 └──────────────┘ └───────┬───────┘  
 │  
 ┌──────────────┐ ┌───────▼───────┐  
 │ Complete │◄────┤ Calculate │  
 │ Sale │ │ Totals │  
 └───────┬──────┘ └───────────────┘  
 │  
 ┌───────────────┼────────────────┐  
 │ │ │  
┌───────▼──────┐ ┌──────▼───────┐ ┌──────▼───────┐  
│ Print Receipt │ │Update Inventory│ │Record Sale in │  
└───────┬──────┘ └──────┬───────┘ │ Database │  
 │ │ └──────────────┘  
 │ │  
┌───────▼──────┐ ┌──────▼───────┐  
│ Give Receipt │ │ Update Sales │  
│ to Customer │ │ Reports │  
└──────────────┘ └──────────────┘

### 6.2 Inventory Update Process

┌─────────┐ ┌──────────────┐ ┌───────────────┐  
│ Start │────►│ Inventory │──Manual─────►│ Enter Quantity│  
└─────────┘ │ Update Source│ │ Adjustment │  
 └──────┬───────┘ └───────┬───────┘  
 │ │  
 │ ▼  
 │ ┌───────────────┐  
 │ │ Enter Reason │  
 │ │ for Change │  
 │ └───────┬───────┘  
 │ │  
 │ ▼  
 │ ┌───────────────┐  
 │ │ Validate │  
 │ │ Adjustment │  
 │ └───────┬───────┘  
 │ │  
 ▼ ▼  
 ┌──────────────┐ ┌────────────────┐  
 │ Sale Process │────────────►│Update Product │  
 └──────────────┘ │ Quantity │  
 └───────┬────────┘  
 │  
 ▼  
 ┌────────────────┐  
 │Record in │  
 │Inventory Log │  
 └───────┬────────┘  
 │  
 ▼  
 ┌────────────────┐  
 │Check for Low │  
 │Stock Condition │  
 └───────┬────────┘  
 │  
 ┌───────▼────────┐  
 │ Generate Alert │  
 │ (if needed) │  
 └────────────────┘

### 6.3 Reporting Process

┌─────────┐ ┌──────────────┐ ┌───────────────┐  
│ Start │────►│ Select Report│────►│ Select Report │  
└─────────┘ │ Type │ │ Parameters │  
 └──────────────┘ └───────┬───────┘  
 │  
 ┌───────▼───────┐  
 │ Generate │  
 │ Report │  
 └───────┬───────┘  
 │  
 ┌────────────────────┼────────────────────┐  
 │ │ │  
 ┌───────▼──────┐ ┌──────▼───────┐ ┌──────▼──────┐  
 │ View Report │ │ Print Report │ │Export Report │  
 │ on Screen │ │ │ │ │  
 └──────────────┘ └──────────────┘ └──────────────┘

### 6.4 Backup Process

┌─────────┐ ┌──────────────┐ ┌───────────────┐  
│ Start │────►│Select Backup │────►│ Choose Backup │  
└─────────┘ │ Type │ │ Destination │  
 └──────────────┘ └───────┬───────┘  
 │  
 ┌───────▼───────┐  
 │ Validate │  
 │ Destination │  
 └───────┬───────┘  
 │  
 ┌───────▼───────┐  
 │ Perform │  
 │ Backup │  
 └───────┬───────┘  
 │  
 ┌────────────────────┼────────────────┐  
 │ │ │  
 ┌───────▼──────┐ ┌──────▼───────┐ ┌──────▼──────┐  
 │Verify Backup │ │Record Backup │ │Show Success │  
 │ (optional) │ │in Log │ │ Message │  
 └──────────────┘ └──────────────┘ └─────────────┘

## 7. Data Flow Diagrams

### 7.1 Context Diagram

┌───────────────┐  
 │ │  
 Product Data │ │ Billing Information  
 ┌───────────────►│ Inventory ├───────────────┐  
 │ │ and │ │  
 │ │ Billing │ │  
 │ │ System │ │  
 │ │ │ │  
 │ └───┬─────┬─────┘ │  
 │ │ │ │  
 │ │ │ │  
 │ │ │ │  
 │ │ │ │  
┌─────▼─────┐ ┌─────▼─────▼─┐ ┌─────▼─────┐  
│ │ │ │ │ │  
│ Supplier │ │ Store │ │ Customer │  
│ │ │ Operator │ │ │  
└───────────┘ └─────────────┘ └───────────┘

### 7.2 Level 1 DFD

┌───────────┐ Add/Update ┌───────────┐  
│ │◄───────────────►│ │  
│ Product │ │ User │  
│ Management│ Product Data │ Interface │  
│ │ │ │  
└─────┬─────┘ └───┬───────┘  
 │ │  
 │ │  
 │ │  
 │ ┌──────────────┐ │  
 │ │ │ │  
 └────────►│ MongoDB │◄──┘  
 │ Database │  
 └──────┬───────┘  
 │  
 │  
┌───────────┐ │ ┌───────────┐  
│ │◄─────────┘ │ │  
│ Inventory │ Inventory Data │ Billing │  
│ Management│◄──────────────────►│ Module │  
│ │ │ │  
└────┬──────┘ └─────┬─────┘  
 │ │  
 │ │  
 │ │  
 │ ┌────────────┐ │  
 │ │ │ │  
 └───────►│ Reporting │◄───────────┘  
 │ Module │  
 │ │  
 └────────────┘

## 8. Security Design

### 8.1 Authentication Mechanism

1. **Password Authentication**
   * Bcrypt/Argon2 for secure password hashing
   * JWT (JSON Web Tokens) for secure sessions
   * Minimum password strength requirements enforced
   * Account lockout after multiple failed attempts
2. **Session Management**
   * JWT tokens with appropriate expiration
   * Token refresh mechanism for extended sessions
   * Token invalidation on logout
   * Session timeout after 30 minutes of inactivity (configurable)

### 8.2 Data Protection

1. **Database Security**
   * Encrypted database files at rest
   * Secure connection strings
   * Input validation and sanitization
   * MongoDB access controls
2. **Application Data**
   * Configuration settings stored securely
   * Sensitive data encrypted in configuration files
   * Secure IPC (Inter-Process Communication) within Electron
   * CORS protection for local API server

### 8.3 Audit Trail

1. **User Activity Logging**
   * Login/logout events recorded
   * Critical operations logged (sales, inventory adjustments)
   * Audit trail for security-sensitive operations
2. **System Event Logging**
   * Application start/stop events
   * Error conditions and exceptions
   * Backup/restore operations

## 9. Exception Handling

### 9.1 Error Categories

1. **User Input Errors**
   * Invalid data entry
   * Business rule violations
   * Missing required fields
2. **System Errors**
   * Database connectivity issues
   * File I/O problems
   * Resource constraints
3. **Business Logic Errors**
   * Inventory discrepancies
   * Calculation errors
   * Process violations

### 9.2 Error Messages

Error messages will follow these guidelines:

* Clear, non-technical language
* Actionable information when possible
* Error codes for technical support reference
* Visual indication of error severity using Material-UI Alert components

Example error messages:

* "Product quantity must be a positive number. Please correct the value."
* "Unable to connect to the database. Please restart the application."
* "Insufficient stock available for [Product Name]. Only [X] items remain."

### 9.3 Recovery Procedures

1. **Data Validation Recovery**
   * Form validation with immediate feedback
   * Field-level error messages
   * Preserve valid user input
   * Provide clear correction instructions
2. **System Error Recovery**
   * Automatic retry for transient failures
   * Safe application state preservation via Redux persistence
   * Clear path to resume operation
   * Application restart capability for critical errors
3. **Database Error Recovery**
   * Transaction handling in MongoDB operations
   * Automatic reconnection attempts
   * Data consistency checks on reconnection
   * Recovery from corrupted data using backups

## 10. Testing Strategy

### 10.1 Unit Testing

1. **Scope**:
   * Backend services
   * React components
   * Redux actions and reducers
   * Utility functions
2. **Approach**:
   * Jest for JavaScript/React testing
   * React Testing Library for component tests
   * Enzyme for complex component testing
   * Mock services for isolated testing
3. **Coverage Goals**:
   * 80% code coverage for business logic
   * 100% coverage for critical calculations
   * All React components tested for rendering and basic interactions

### 10.2 Integration Testing

1. **Scope**:
   * API service integration
   * Database operations
   * React component integration
   * Redux state management
2. **Approach**:
   * Integration test suites with Jest
   * MongoDB memory server for database testing
   * React component integration tests
   * Electron-specific integration tests
3. **Focus Areas**:
   * Billing to inventory integration
   * Data flow through React component hierarchies
   * Redux state updates
   * Database transaction integrity

### 10.3 System Testing

1. **Scope**:
   * End-to-end workflows
   * Performance testing
   * Stability testing
   * Electron application packaging and installation
2. **Approach**:
   * Cypress for end-to-end testing
   * Spectron for Electron-specific testing
   * Manual test scripts
   * Performance profiling
3. **Scenarios**:
   * Complete sales process
   * Daily operations sequence
   * Monthly/yearly reporting cycles
   * Backup and restore operations
   * Application installation and updates

### 10.4 User Acceptance Testing

1. **Scope**:
   * Business requirements validation
   * Usability evaluation
   * Real-world scenario testing
2. **Approach**:
   * Guided testing with actual users
   * Predefined test scripts
   * Observation and feedback collection
3. **Success Criteria**:
   * All critical business functions work correctly
   * Users can complete tasks independently
   * Performance meets expectations
   * No blocking issues discovered

## 11. Implementation Plan

### 11.1 Development Environment

1. **Development Tools**:
   * Visual Studio Code with appropriate extensions
   * Node.js and npm/yarn package management
   * MongoDB Community Edition
   * Git for version control
   * Postman for API testing
2. **Development Practices**:
   * Feature branching workflow
   * Code review process
   * Continuous integration with GitHub Actions
   * Linting with ESLint and Prettier

### 11.2 Technology Stack

1. **Frontend**:
   * React.js for UI components
   * Redux for state management
   * Material-UI for component library
   * Electron.js for desktop application wrapper
2. **Backend**:
   * Node.js runtime
   * Express.js for API framework
   * MongoDB for database
   * Mongoose for object data modeling
3. **Development Tools**:
   * Webpack for bundling
   * Babel for JavaScript transpilation
   * Jest for testing
   * Electron Forge/Builder for packaging
4. **Reporting**:
   * React-ChartJS for charts and graphs
   * jsPDF for PDF generation
   * ExcelJS for Excel export
   * html-to-pdf for receipt printing

### 11.3 Development Phases

1. **Phase 1: Core System** (4 weeks)
   * Electron application scaffolding
   * MongoDB schema implementation
   * Basic Express API setup
   * Product management module
   * User authentication
2. **Phase 2: Sales Functions** (3 weeks)
   * Billing module
   * Sales reporting
   * Inventory integration
   * Receipt printing
3. **Phase 3: Complete Inventory** (2 weeks)
   * Full inventory management
   * Stock adjustments
   * Low stock alerts
   * Supplier management
4. **Phase 4: Reporting & Utilities** (3 weeks)
   * Complete reporting system
   * Data backup/restore
   * System configuration
   * Performance optimization
5. **Phase 5: Testing & Refinement** (2 weeks)
   * System testing
   * User acceptance testing
   * Bug fixing
   * Performance tuning

### 11.4 Deployment Strategy

1. **Application Packaging**:
   * Electron Builder for creating installers
   * Platform-specific packages (Windows, macOS, Linux)
   * Auto-update configuration
   * Installation scripts for dependencies
2. **Initial Deployment**:
   * On-site installation
   * Initial data setup
   * User training session
   * Configuration to local requirements
3. **Post-Deployment Support**:
   * 2 weeks of direct support
   * Issue resolution protocol
   * Remote assistance capability
   * Feedback collection for improvements

## 12. Appendices

### 12.1 Technical Libraries and Frameworks

1. **Core Technologies**:
   * Electron.js (v15.0.0 or later)
   * React.js (v18.0.0 or later)
   * Node.js (v16.0.0 or later)
   * MongoDB (v5.0.0 or later)
2. **Frontend Libraries**:
   * Material-UI (v5.0.0 or later)
   * Redux (v4.0.0 or later)
   * React Router (v6.0.0 or later)
   * Formik for form handling
   * Yup for validation
3. **Backend Libraries**:
   * Express.js (v4.0.0 or later)
   * Mongoose (v6.0.0 or later)
   * JWT for authentication
   * Bcrypt for password hashing
   * Multer for file uploads
4. **Development & Utility Libraries**:
   * Jest & React Testing Library
   * Webpack & Babel
   * ESLint & Prettier
   * Winston for logging
   * Nodemon for development

### 12.2 API Specifications

1. **Product API**
   * GET /api/products - List all products
   * GET /api/products/:id - Get product by ID
   * POST /api/products - Create new product
   * PUT /api/products/:id - Update product
   * DELETE /api/products/:id - Delete product
2. **Sales API**
   * GET /api/sales - List all sales
   * GET /api/sales/:id - Get sale by ID
   * POST /api/sales - Create new sale
   * PUT /api/sales/:id - Update sale
   * DELETE /api/sales/:id - Cancel sale
3. **Inventory API**
   * GET /api/inventory/log - Get inventory log
   * POST /api/inventory/adjust - Adjust inventory
   * GET /api/inventory/low-stock - Get low stock items
4. **Report API**
   * GET /api/reports/sales - Generate sales report
   * GET /api/reports/inventory - Generate inventory report
   * GET /api/reports/profit - Generate profit report
   * POST /api/reports/export - Export report
5. **Authentication API**
   * POST /api/auth/login - Login
   * POST /api/auth/logout - Logout
   * PUT /api/auth/change-password - Change password
6. **Backup API**
   * POST /api/backup/create - Create backup
   * POST /api/backup/restore - Restore from backup
   * GET /api/backup/history - Get backup history