Inventory Stock Entry and Purchase System Documentation

# Table of Contents

1. [System Overview](#system-overview)
2. [Inventory Stock Entry Process](#inventory-stock-entry-process)
3. [Purchase Bill Creation and Management](#purchase-bill-creation-and-management)
4. [Voucher, Invoice, and Bill Linking](#voucher-invoice-and-bill-linking)
5. [Data Flow Architecture](#data-flow-architecture)
6. [API Reference](#api-reference)
7. [User Interface Components](#user-interface-components)
8. [Business Logic and Validation](#business-logic-and-validation)
9. [Integration Points](#integration-points)
10. [Troubleshooting Guide](#troubleshooting-guide)

---

# System Overview

The Retail Management System provides a comprehensive solution for inventory management, purchase bill processing, and financial tracking. The system handles the complete lifecycle from stock entry to bill generation and payment processing.

## Key Features

* \*\*Stock Entry Management\*\*: Record incoming stock with challan details
* \*\*Purchase Bill Processing\*\*: Create and manage purchase bills from challans
* \*\*Payment Management\*\*: Handle multiple payment methods and tracking
* \*\*Invoice Generation\*\*: Generate invoices for sales transactions
* \*\*Financial Integration\*\*: Link vouchers, invoices, and bills for complete audit trail

---

# Inventory Stock Entry Process

## 1. Stock Entry Workflow

## Step 1: Vendor Selection

interface Vendor {  
 id: string  
 name: string  
 address?: string  
 contact?: string  
 gstNo?: string  
}

**Process:**

1. Select vendor from dropdown or search
2. System validates vendor details

## Step 2: Challan Information Entry

interface StockInForm {  
 challanDate: string  
 challanNo: string  
 transportName: string  
 transportNo: string  
 transportCharges: number  
 billNo?: string  
 billDate?: string  
 products: StockInProduct[]  
}

**Required Fields:**

* Challan Number
* Transport Details
* Product Information

## Step 3: Product Details Entry

interface StockInProduct {  
 slNo: number  
 productId: string  
 productName: string  
 inStock: number  
 qty: number  
 batchNo: string  
 mfDate: string  
 expDate: string  
 unitPrice: number  
 totalPrice: number  
}

**Product Entry Process:**

1. Select product from inventory
2. Enter quantity and pricing details
3. Add batch and expiry information
4. Calculate totals automatically

## Step 4: Stock Record Creation

interface StockInRecord {  
 id: string  
 vendorId: string  
 vendorName: string  
 challanDate: string  
 challanNo: string  
 transportName: string  
 transportNo: string  
 transportCharges: number  
 billNo?: string  
 billDate?: string  
 products: StockInProduct[]  
 totalAmount: number  
 createdAt: string  
}

**System Actions:**

1. Generate challan for purchase processing
2. Update product stock levels
3. Create audit trail entry

---

# Purchase Bill Creation and Management

## 1. Purchase Bill Workflow

## Step 1: Vendor and Challan Selection

interface PurchaseBillCreation {  
 billNo: string  
 billDate: string  
 challanIds: string[]  
}

**Process:**

1. View pending challans for selected vendor
2. Select multiple challans for bill creation
3. Enter bill number and date

## Step 1.5: Product Selection for Partial Payment

interface ProductSelectionState {  
 challanId: string  
 productId: string  
 isSelected: boolean  
 paymentStatus: 'unpaid' | 'partially\_paid' | 'fully\_paid' | 'pending'  
 partialPaymentAmount?: number  
 billId?: string  
 isEditable: boolean  
}

**Partial Payment Process:**

1. \*\*Payment Status Tracking\*\*: Each product has a payment status (Unpaid, Partially Paid, Fully Paid, Pending)
2. \*\*Partial Payment Processing\*\*: Users can make partial payments for selected products
3. \*\*Row Locking\*\*: Once payment is made, product rows become non-editable or limited editing
4. \*\*Visual Indicators\*\*: Color-coded status badges show payment state

## Step 2: Product Details Processing

interface PurchaseBillProduct {  
 slNo: number  
 productId: string  
 productName: string  
 sku: string  
 quantity: number  
 rate: number  
 discount: number  
 taxableValue: number  
 sgst: number  
 cgst: number  
 total: number  
 billNo: string  
 billDate: string  
 challanId: string  
 challanNo: string  
 isSelected?: boolean  
 paymentStatus?: 'unpaid' | 'partially\_paid' | 'fully\_paid' | 'pending'  
 partialPaymentAmount?: number  
 originalChallanProductId?: string  
 isEditable?: boolean  
}

**Product Processing:**

1. Allow quantity and rate adjustments
2. Calculate taxes and totals
3. Validate product information
4. \*\*Partial Payment Support\*\*: Enable product selection for partial payments
5. \*\*Payment Status Management\*\*: Track and display payment status for each product
6. \*\*Row State Management\*\*: Control editability based on payment status

## Step 3: Payment Entry

interface PaymentEntry {  
 transactionTypes: ('cash' | 'cheque' | 'credit' | 'discount' | 'upi')[]  
 amounts: {  
 cash?: number  
 cheque?: number  
 credit?: number  
 discount?: number  
 upi?: number  
 }  
 paymentDate: string  
 reference?: string  
 chequeNo?: string  
 bankName?: string  
 discountReason?: string  
 upiId?: string  
 upiTransactionId?: string  
}

**Payment Methods:**

* \*\*Cheque\*\*: Bank cheque with details
* \*\*Credit\*\*: Credit payment terms
* \*\*Discount\*\*: Discount adjustments
* \*\*UPI\*\*: Digital payment with transaction ID

## Step 3.5: Partial Payment Management

interface PartialPaymentEntry {  
 productId: string  
 challanId: string  
 challanNo: string  
 productName: string  
 totalAmount: number  
 paidAmount: number  
 remainingAmount: number  
 paymentStatus: 'unpaid' | 'partially\_paid' | 'fully\_paid' | 'pending'  
 paymentMethods: {  
 cash?: number  
 cheque?: number  
 credit?: number  
 discount?: number  
 upi?: number  
 }  
 paymentDate: string  
 reference?: string  
 isEditable: boolean  
}

**Partial Payment Features:**

1. \*\*Payment Status Tracking\*\*: Real-time status updates (Unpaid, Partially Paid, Fully Paid, Pending)
2. \*\*Payment Amount Management\*\*: Track partial payments and remaining amounts
3. \*\*Row State Control\*\*: Lock/unlock product rows based on payment status
4. \*\*Visual Status Indicators\*\*: Color-coded badges for easy status identification
5. \*\*Payment History\*\*: Maintain audit trail of all payment transactions

## Step 4: Bill Summary and Submission

interface PurchaseBill {  
 id: string  
 billNo: string  
 billDate: string  
 vendorId: string  
 vendorName: string  
 challanIds: string[]  
 challanNumbers: string[]  
 products: PurchaseBillProduct[]  
 paymentEntry: PaymentEntry  
 totals: {  
 discount: number  
 taxableValue: number  
 sgst: number  
 cgst: number  
 total: number  
 }  
 status: 'draft' | 'pending' | 'paid' | 'cancelled'  
 remainingAmount: number  
 advanceAmount: number  
 createdAt: string  
 updatedAt: string  
}

---

# Voucher, Invoice, and Bill Linking

## 1. Document Linking Architecture

## Challan to Purchase Bill Linking

// Challan creation from stock entry  
dispatch(addChallanFromStockIn({  
 id: stockInRecord.id,  
 vendorId: stockInRecord.vendorId,  
 vendorName: stockInRecord.vendorName,  
 challanDate: stockInRecord.challanDate,  
 challanNo: stockInRecord.challanNo,  
 transportName: stockInRecord.transportName,  
 transportNo: stockInRecord.transportNo,  
 transportCharges: stockInRecord.transportCharges,  
 products: stockInRecord.products,  
 totalAmount: stockInRecord.totalAmount  
}))

## Purchase Bill to Payment Linking

// Payment processing  
dispatch(processPurchaseBill({  
 products: currentPurchaseBill.products.map(product => ({  
 productId: product.productId,  
 quantity: product.quantity,  
 unitPrice: product.unitPrice  
 }))  
}))

## 2. Invoice Generation Process

## Sales Invoice Creation

interface SalesInvoice {  
 number: string  
 date: string  
 customer: string  
 items: CartItem[]  
 subtotal: number  
 discount: number  
 gst: number  
 roundOff: number  
 total: number  
 paymentMethod: string  
 paidAmount: number  
}

**Invoice Generation Steps:**

1. Generate unique invoice number
2. Calculate taxes and totals
3. Update inventory stock
4. Create payment record

## 3. Financial Document Flow

Stock Entry → Challan → Purchase Bill → Payment → Invoice  
 ↓ ↓ ↓ ↓ ↓  
 Inventory Pending Bill Status Payment Sales  
 Update Status Tracking History Record

---

# Data Flow Architecture

## 1. State Management Structure

## Redux Store Structure

interface RootState {  
 products: ProductState  
 challan: ChallanState  
 sales: SalesState  
 auth: AuthState  
}  
  
interface ProductState {  
 products: Product[]  
 vendors: Vendor[]  
 customers: Customer[]  
 stockInRecords: StockInRecord[]  
 loading: boolean  
 error: string | null  
}  
  
interface ChallanState {  
 challans: Challan[]  
 selectedVendor: string | null  
 pendingChallans: Challan[]  
 selectedChallans: string[]  
 currentPurchaseBill: PurchaseBill | null  
 paymentHistory: PaymentHistory[]  
 loading: boolean  
}

## 2. Data Persistence

## Local Storage Integration

* User preferences and settings
* Temporary form data
* Session management
* Offline data caching

## Database Operations

* Stock record creation
* Purchase bill processing
* Payment tracking
* Audit trail maintenance

---

# API Reference

## 1. Stock Entry APIs

## Create Stock Record

dispatch(addStockInRecord(stockInRecord: StockInRecord))

## Update Product Stock

dispatch(updateStock({  
 sku: string,  
 quantity: number,  
 operation: 'add' | 'subtract'  
}))

## 2. Purchase Bill APIs

## Create Purchase Bill

dispatch(createPurchaseBill({  
 billNo: string,  
 billDate: string,  
 challanIds: string[]  
}))

## Update Purchase Bill Product

dispatch(updatePurchaseBillProduct({  
 productId: string,  
 field: string,  
 value: any  
}))

## Process Purchase Bill

dispatch(processPurchaseBill({  
 products: Array<{  
 productId: string,  
 quantity: number,  
 unitPrice: number  
 }>  
}))

## 3. Payment Management APIs

## Update Payment Entry

dispatch(updatePaymentEntry(paymentData: Partial<PaymentEntry>))

## Update Advance Amount

dispatch(updateAdvanceAmount(amount: number))

## Submit Purchase Bill

dispatch(submitPurchaseBill())

## 4. Partial Payment Management APIs

## Select Product for Payment

dispatch(selectProductForPayment({  
 challanId: string,  
 productId: string,  
 isSelected: boolean  
}))

## Update Product Payment Status

dispatch(updateProductPaymentStatus({  
 challanId: string,  
 productId: string,  
 paymentStatus: 'unpaid' | 'partially\_paid' | 'fully\_paid' | 'pending',  
 partialPaymentAmount?: number,  
 billId?: string,  
 billNo?: string,  
 paymentDate?: string  
}))

## Process Partial Payment

dispatch(processPartialPayment({  
 challanId: string,  
 productId: string,  
 paymentAmount: number,  
 paymentMethods: {  
 cash?: number,  
 cheque?: number,  
 credit?: number,  
 discount?: number,  
 upi?: number  
 },  
 paymentDate: string,  
 reference?: string  
}))

---

# User Interface Components

## 1. Stock Entry Components

## StockInPage

* \*\*Location\*\*: `apps/web/app/retail/inventory/stock-in/page.tsx`
* \*\*Purpose\*\*: Main stock entry interface
* \*\*Features\*\*:
* Vendor selection
* Challan information entry
* Product details management
* Form validation

## Key Features:

* Dynamic product row addition/removal
* Real-time total calculation
* Batch and expiry date tracking
* Transport details management

## 2. Purchase Bill Components

## PurchasePage

* \*\*Location\*\*: `apps/web/app/retail/purchase/page.tsx`
* \*\*Purpose\*\*: Purchase bill creation and management
* \*\*Features\*\*:
* Vendor and challan selection
* Product table management
* Payment entry forms
* Bill summary display

## ChallanProductTable

* \*\*Location\*\*: `apps/web/app/retail/purchase/components/ChallanProductTable.tsx`
* \*\*Purpose\*\*: Product details table for purchase bills with partial payment support
* \*\*Features\*\*:
* Editable product information
* Tax calculations
* Total computations
* Compact layout optimization
* \*\*Product Selection\*\*: Checkboxes for individual product selection
* \*\*Payment Status Display\*\*: Visual indicators for payment status
* \*\*Row State Management\*\*: Conditional editing based on payment status
* \*\*Partial Payment Tracking\*\*: Display partial payment amounts

## PaymentEntryForm

* \*\*Location\*\*: `apps/web/app/retail/purchase/components/PaymentEntryForm.tsx`
* \*\*Purpose\*\*: Payment method selection and entry
* \*\*Features\*\*:
* Multiple payment methods
* Payment amount tracking
* Reference information
* Payment summary

## 3. Sales and POS Components

## POSPage

* \*\*Location\*\*: `apps/web/app/retail/pos/page.tsx`
* \*\*Purpose\*\*: Point of sale transactions
* \*\*Features\*\*:
* Product selection
* Cart management
* Payment processing
* Invoice generation

## SalesPage

* \*\*Location\*\*: `apps/web/app/retail/sales/page.tsx`
* \*\*Purpose\*\*: Sales order management
* \*\*Features\*\*:
* Customer selection
* Product configuration
* Tax calculations
* Order processing

---

# Business Logic and Validation

## 1. Stock Entry Validation

## Required Field Validation

// Vendor selection validation  
if (!selectedVendor) {  
 showError('Vendor Required', 'Please select a vendor first')  
 return  
}  
  
// Challan number validation  
if (form.challanNo === '') {  
 showError('Challan Number Required', 'Please enter challan number')  
 return  
}  
  
// Product details validation  
if (form.products.some(p => p.productId === '' || p.qty === 0)) {  
 showError('Product Details Required', 'Please fill in all product details')  
 return  
}

## Stock Update Logic

// Process purchase bill and update stock  
dispatch(processPurchaseBill({  
 products: currentPurchaseBill.products.map(product => ({  
 productId: product.productId,  
 quantity: product.quantity,  
 unitPrice: product.unitPrice  
 }))  
}))

## 2. Purchase Bill Validation

## Bill Creation Validation

// Bill information validation  
if (!billNo || !billDate) {  
 showError('Missing Information', 'Please fill in bill number and date')  
 return  
}

## Payment Validation

// Payment amount validation  
const getTotalAmount = () => {  
 return Object.values(amounts).reduce((sum, amount) => sum + (amount || 0), 0)  
}

## 3. Financial Calculations

## Tax Calculations

// GST calculation  
const calculateGST = (taxableValue: number, gstRate: number) => {  
 const gstAmount = (taxableValue \* gstRate) / 100  
 const sgst = gstAmount / 2  
 const cgst = gstAmount / 2  
 return { sgst, cgst, totalGST: gstAmount }  
}

## Partial Payment Calculations

// Partial payment status calculation  
const calculatePaymentStatus = (paidAmount: number, totalAmount: number) => {  
 if (paidAmount >= totalAmount) {  
 return 'fully\_paid'  
 } else if (paidAmount > 0) {  
 return 'partially\_paid'  
 } else {  
 return 'unpaid'  
 }  
}  
  
// Remaining amount calculation  
const calculateRemainingAmount = (totalAmount: number, paidAmount: number) => {  
 return Math.max(0, totalAmount - paidAmount)  
}  
  
// Payment validation  
const validatePartialPayment = (paymentAmount: number, remainingAmount: number) => {  
 return paymentAmount > 0 && paymentAmount <= remainingAmount  
}

## Total Calculations

// Purchase bill totals  
const calculateTotals = (products: PurchaseBillProduct[]) => {  
 const discount = products.reduce((sum, p) => sum + p.discount, 0)  
 const taxableValue = products.reduce((sum, p) => sum + p.taxableValue, 0)  
 const sgst = products.reduce((sum, p) => sum + p.sgst, 0)  
 const cgst = products.reduce((sum, p) => sum + p.cgst, 0)  
 const total = products.reduce((sum, p) => sum + p.total, 0)  
   
 return { discount, taxableValue, sgst, cgst, total }  
}

---

# Integration Points

## 1. Inventory Management Integration

## Stock Level Updates

* Automatic stock updates on purchase bill processing
* Real-time inventory tracking
* Low stock alerts and notifications
* Stock adjustment capabilities

## Product Management

* Product catalog integration
* SKU-based product identification
* Category and brand management
* Pricing and cost tracking

## 2. Financial System Integration

## Payment Processing

* Multiple payment method support
* Payment tracking and history
* Advance payment management
* Credit and discount handling

## Accounting Integration

* Purchase bill generation
* Payment voucher creation
* Tax calculation and reporting
* Financial audit trail

## 3. Reporting and Analytics

## Purchase Reports

* Vendor-wise purchase analysis
* Product-wise purchase tracking
* Payment status reports
* Tax calculation summaries

## Inventory Reports

* Stock level reports
* Product movement tracking
* Expiry date monitoring
* Cost analysis reports

---

# Troubleshooting Guide

## 1. Common Issues and Solutions

## Stock Entry Issues

\*\*Issue\*\*: Products not appearing in dropdown

* \*\*Check\*\*: Product active status and inventory setup

\*\*Issue\*\*: Challan creation failing

* \*\*Check\*\*: Form validation and data integrity

## Purchase Bill Issues

\*\*Issue\*\*: Challans not showing for vendor

* \*\*Check\*\*: Pending challans filter and data loading

\*\*Issue\*\*: Payment calculation errors

* \*\*Check\*\*: Payment entry validation and totals calculation

## Partial Payment Issues

\*\*Issue\*\*: Product selection not working

* \*\*Check\*\*: Product selection handlers and state updates

\*\*Issue\*\*: Payment status not updating

* \*\*Check\*\*: Payment amount validation and status transitions

\*\*Issue\*\*: Product rows not locking after payment

* \*\*Check\*\*: Row state management and conditional rendering

## 2. Data Validation Issues

## Form Validation

* Ensure all required fields are filled
* Check data format and type validation
* Verify business rule compliance
* Test calculation accuracy

## System Integration

* Check Redux state management
* Verify API endpoint connectivity
* Test data persistence
* Validate user permissions

## 3. Performance Optimization

## UI Performance

* Implement component memoization
* Optimize re-rendering cycles
* Use efficient data structures
* Minimize API calls

## Data Management

* Implement pagination for large datasets
* Use efficient search and filtering
* Optimize database queries
* Cache frequently accessed data

---

# Conclusion

This documentation provides a comprehensive guide to the inventory stock entry and purchase system. The system offers a complete solution for managing stock, processing purchases, and maintaining financial records with proper audit trails and integration points.

For additional support or feature requests, please refer to the development team or system administrator.

---

\*\*Document Version\*\*: 1.0

\*\*Maintained By\*\*: Development Team