RPOS Server Documentation - Version 2.1

Enterprise POS System - Complete Technical Documentation

Version: 2.1 **Last Updated**: January 2026 **Stack**: TypeScript + PostgreSQL + TypeORM + Redis + BullMQ/SQS + Socket.io

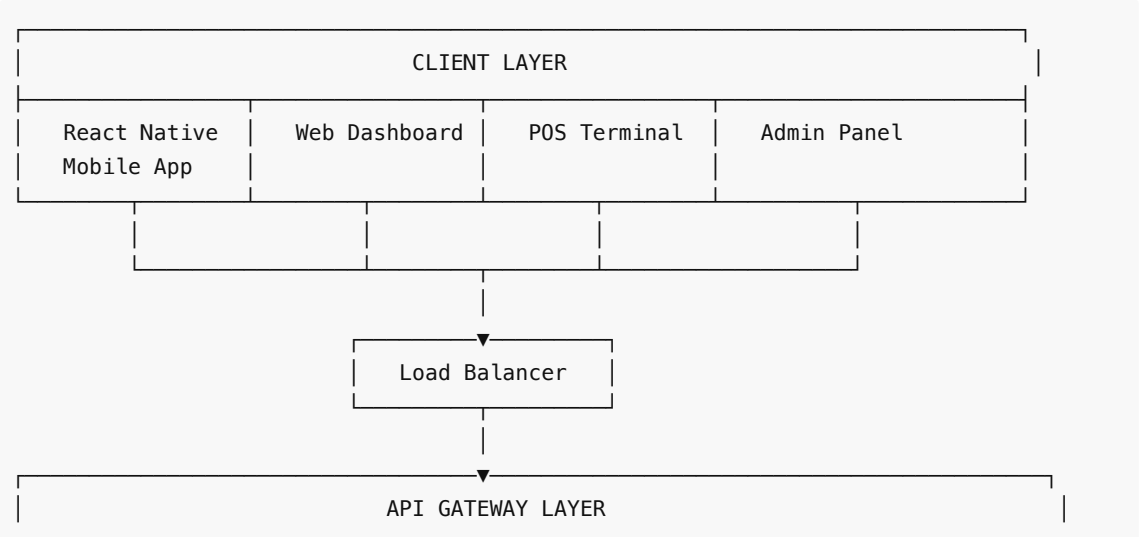
---

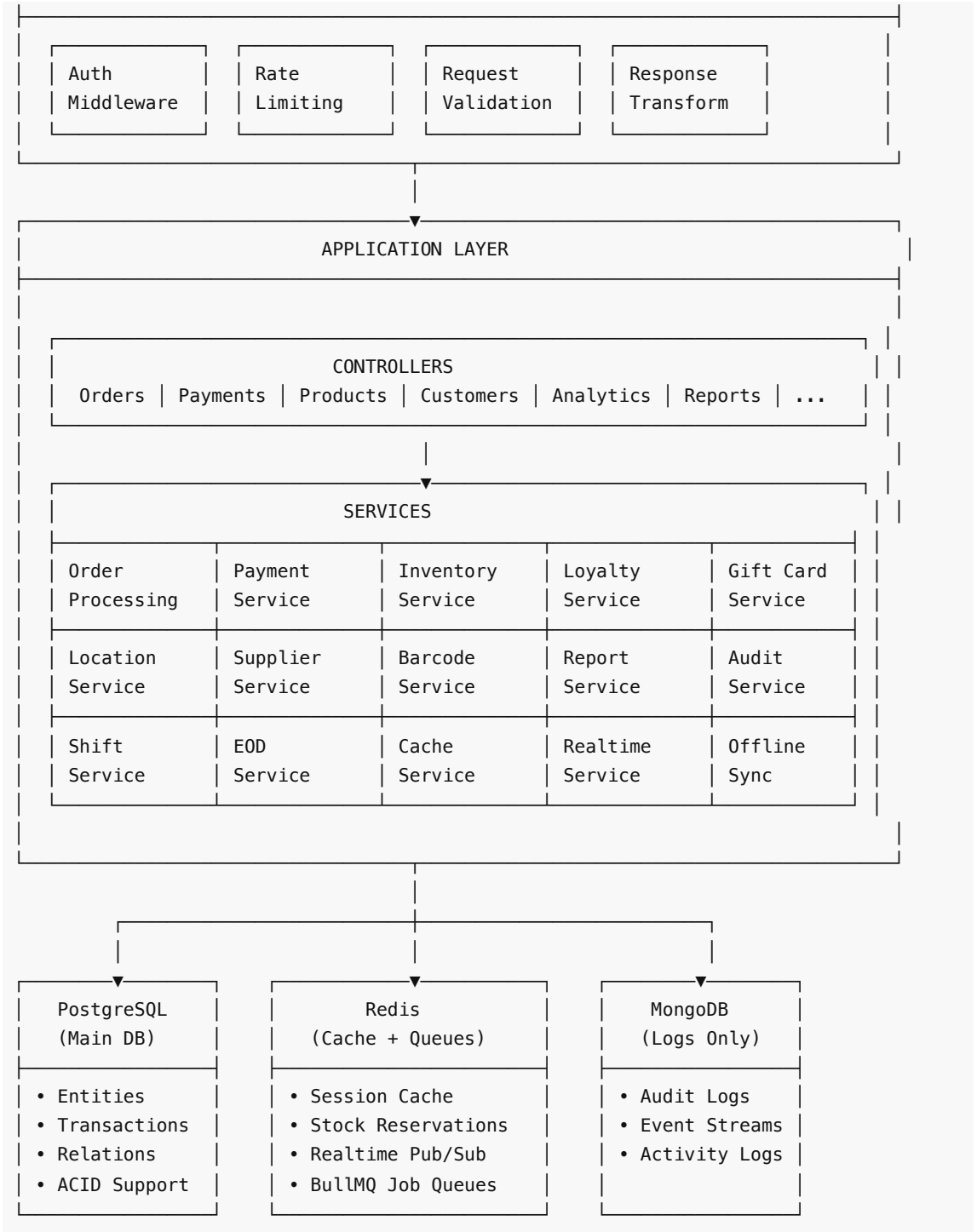
Table of Contents

---

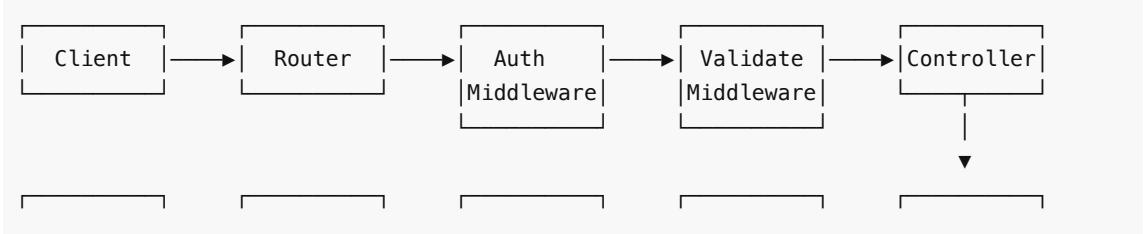
1. System Architecture

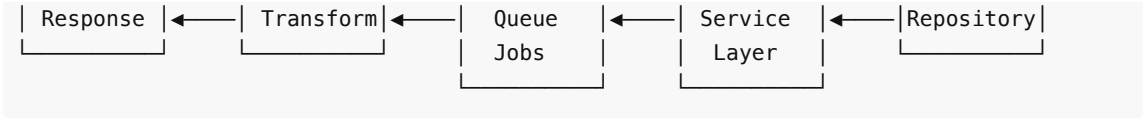
High-Level Architecture Diagram





Request Flow Diagram





---

2. Technology Stack

Core Technologies

Component	Technology	Purpose
Runtime	Node.js 18+	Server runtime
Language	TypeScript 5.x	Type-safe development
Framework	Express.js 4.x	HTTP server
Main Database	PostgreSQL 15+	Transactional data
ORM	TypeORM 0.3.x	Database abstraction
Cache	Redis 7+	Caching & pub/sub
Queue	BullMQ	Background job processing
WebSocket	Socket.io	Real-time communication
Log Database	MongoDB	Event logs & audit trail

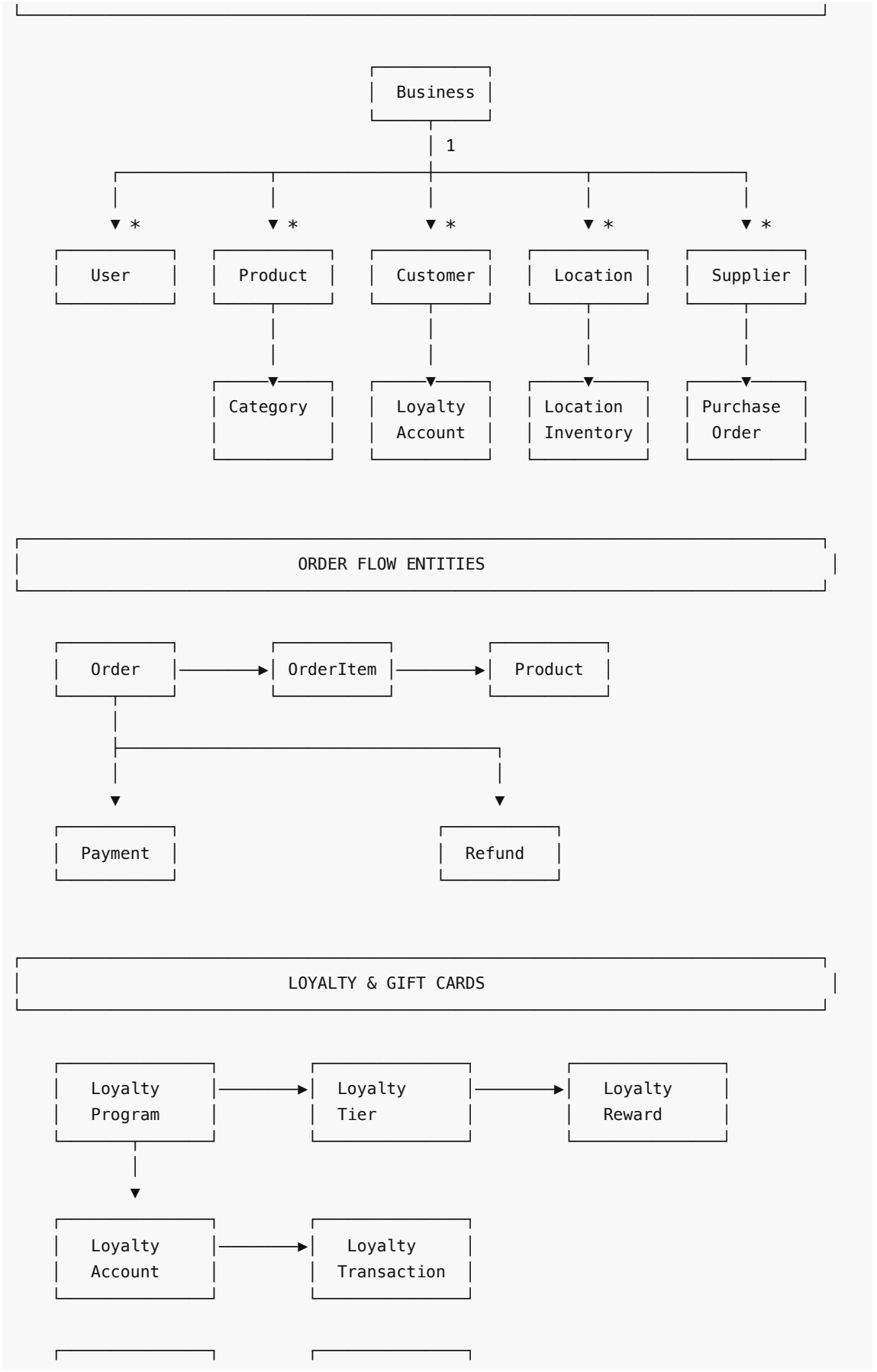
Key Dependencies

```
{
  "dependencies": {
    "express": "^4.18.x",
    "typeorm": "^0.3.x",
    "pg": "^8.x",
    "ioredis": "^5.x",
    "bullmq": "^4.x",
    "socket.io": "^4.x",
    "jsonwebtoken": "^9.x",
    "bcryptjs": "^2.x",
    "uuid": "^9.x",
    "class-validator": "^0.14.x"
  }
}
```

---

3. Database Schema

Entity Relationship Diagram


Business
1
User
Product
Customer
Location
Supplier
Category
Loyalty Account
Location Inventory
Purchase Order
ORDER FLOW ENTITIES
Order
OrderItem
Product
Payment
Refund
LOYALTY & GIFT CARDS
Loyalty Program
Loyalty Tier
Loyalty Reward
Loyalty Account
Loyalty Transaction

Complete Entity List

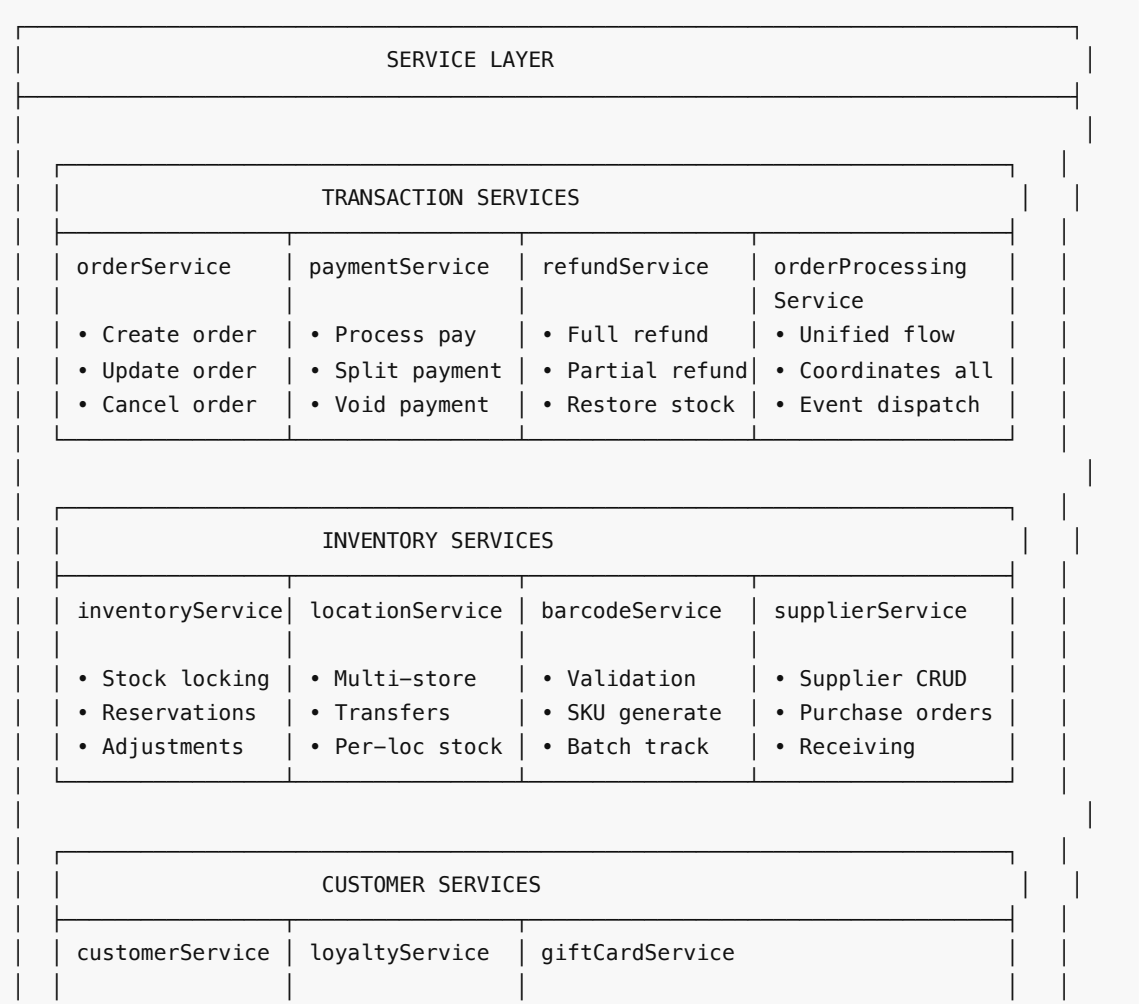
Entity	Table Name	Description
Business	`businesses`	Multi-tenant business accounts
User	`users`	Staff, managers, admins
Product	`products`	Inventory items
Category	`categories`	Product categories (hierarchical)
Customer	`customers`	Customer records
Order	`orders`	Sales transactions
OrderItem	`order_items`	Line items in orders
Payment	`payments`	Payment records
Refund	`refunds`	Refund records
Coupon	`coupons`	Discount codes
File	`files`	Media storage
Shift	`shifts`	Cash register shifts
CashMovement	`cash_movements`	Cash drawer transactions
EODReport	`eod_reports`	End-of-day reconciliation
GiftCard	`gift_cards`	Gift card records
GiftCardTransaction	`gift_card_transactions`	Gift card usage
LoyaltyProgram	`loyalty_programs`	Loyalty program config
LoyaltyTier	`loyalty_tiers`	Membership tiers
LoyaltyAccount	`loyalty_accounts`	Customer loyalty accounts
LoyaltyTransaction	`loyalty_transactions`	Points transactions
LoyaltyReward	`loyalty_rewards`	Redeemable rewards
Location	`locations`	Store locations
LocationInventory	`location_inventory`	Per-location stock
StockTransfer	`stock_transfers`	Inter-store transfers
StockTransferItem	`stock_transfer_items`	Transfer line items
ProductBarcode	`product_barcodes`	Barcode mappings

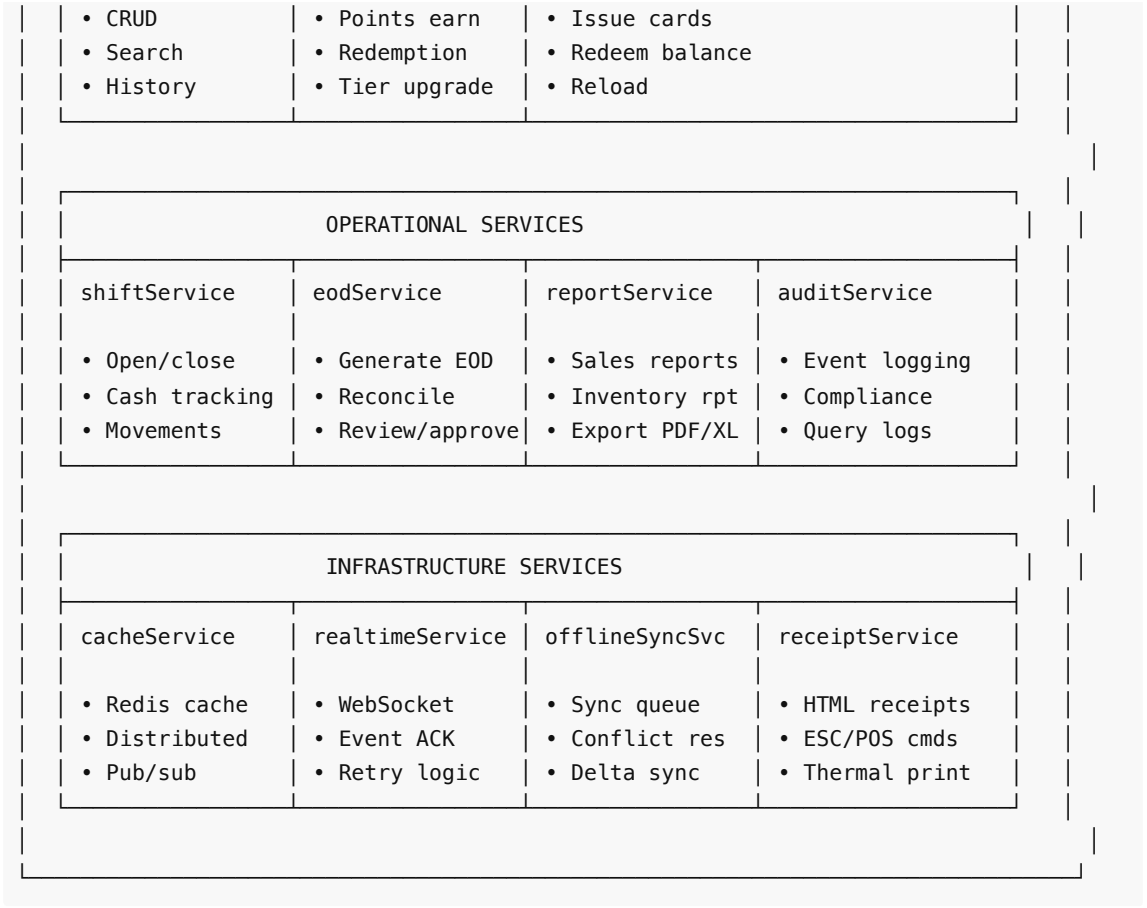
SKUConfiguration	`sku_configurations`	SKU generation rules
ProductBatch	`product_batches`	Batch/lot tracking
BarcodeScanLog	`barcode_scan_logs`	Scan audit trail
Supplier	`suppliers`	Vendor records
SupplierProduct	`supplier_products`	Supplier-product mapping
PurchaseOrder	`purchase_orders`	Purchase orders
PurchaseOrderItem	`purchase_order_items`	PO line items
PurchaseOrderReceiving	`purchase_order_receivings`	Goods receipt
PurchaseOrderReceivingItem	`purchase_order_receiving_items`	Receipt line items
SupplierPayment	`supplier_payments`	Vendor payments

---

4. Core Services

Service Architecture





---

5. Payment System

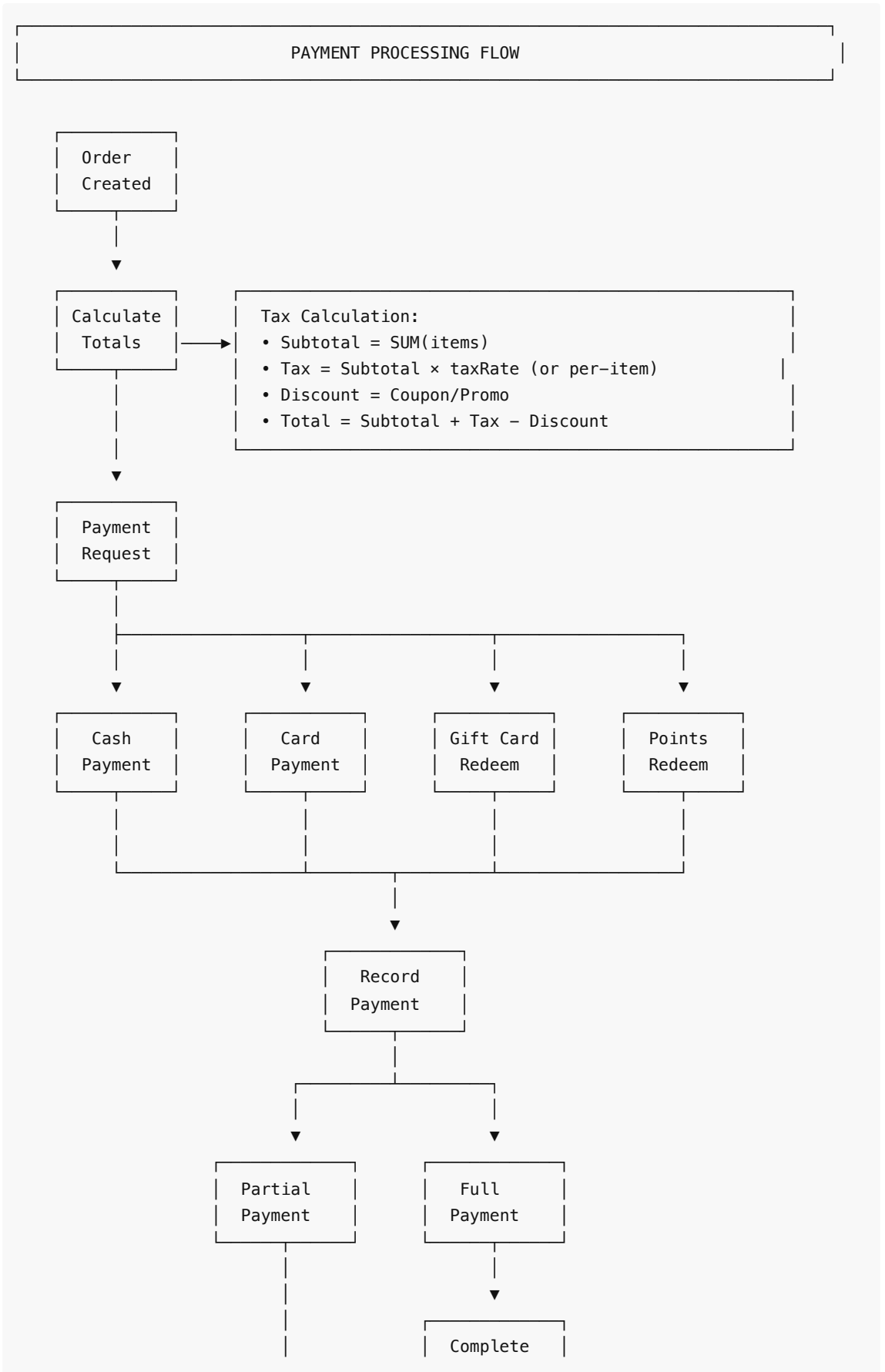
Payment Service (`payment.service.ts`)

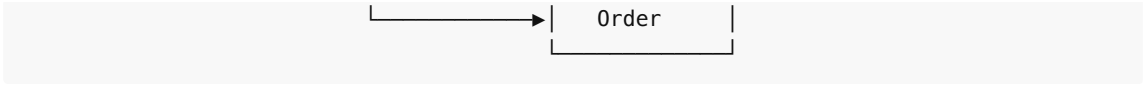
The payment system handles all monetary transactions in the POS, supporting multiple payment methods, split payments, and full integration with the order lifecycle.

Supported Payment Methods

```
enum PaymentMethod {
  CASH = 'cash',
  CREDIT_CARD = 'credit_card',
  DEBIT_CARD = 'debit_card',
  GIFT_CARD = 'gift_card',
  LOYALTY_POINTS = 'loyalty_points',
  STORE_CREDIT = 'store_credit',
  CHECK = 'check',
  MOBILE_PAYMENT = 'mobile_payment',
  BANK_TRANSFER = 'bank_transfer',
  OTHER = 'other',
}
```

Payment Flow Diagram


PAYMENT PROCESSING FLOW
Order Created
Calculate Totals
Tax Calculation:
• Subtotal = SUM(items)
• Tax = Subtotal × taxRate (or per-item)
• Discount = Coupon/Promo
• Total = Subtotal + Tax − Discount
Payment Request
Cash Payment
Card Payment
Gift Card Redeem
Points Redeem
Record Payment
Partial Payment
Full Payment
Complete



Key Methods

```
// Calculate order totals with tax
async calculateTotals(dto: CalculateTotalsDTO): Promise<{
  subtotal: number;
  taxAmount: number;
  discountAmount: number;
  total: number;
  items: Array<{...}>;
}>

// Process single payment
async processPayment(dto: ProcessPaymentDTO): Promise<{
  payment: Payment;
  order: Order;
  changeAmount: number;
  fullyPaid: boolean;
}>

// Process split payment (multiple payment methods)
async processSplitPayment(dto: ProcessSplitPaymentDTO): Promise<{
  payments: Payment[];
  order: Order;
  fullyPaid: boolean;
  totalApplied: number;
  remainingDue: number;
}>

// Process refund
async processRefund(dto: ProcessRefundDTO): Promise<Refund>
```

Tax Calculation Strategies

```
┌──────────────────────────────────────────────────────────────────────┐
│                        TAX CALCULATION TYPES                         │
├──────────────────────────────────────────────────────────────────────┤
│                                                                      │
│  1. INCLUSIVE (Tax included in price)                                │
│     ┌──────────────────────────────────────────────┐                 │
│     │ Price: $10.00 (includes tax)                 │                 │
│     │ Tax Rate: 10%                                │                 │
│     │ Subtotal: $10.00 / 1.10 = $9.09              │                 │
│     │ Tax: $9.09 × 0.10 = $0.91                    │                 │
│     │ Total: $10.00                                │                 │
│     └──────────────────────────────────────────────┘                 │
│                                                                      │
│  2. EXCLUSIVE (Tax added to price)                                   │
│     ┌──────────────────────────────────────────────┐                 │
```

```
│     │ Price: $10.00 (before tax)                                │                        │
│     │ Tax Rate: 10%                                             │                        │
│     │ Subtotal: $10.00                                          │                        │
│     │ Tax: $10.00 × 0.10 = $1.00                                │                        │
│     │ Total: $11.00                                             │                        │
│     └─────────────────────────────────────────────────────────┘                          │
│                                                                                          │
│  3. PER_ITEM (Different tax rates per product)                                           │
│     ┌─────────────────────────────────────────────────────────┐                          │
│     │ Item A: $10.00 @ 10% tax = $11.00                         │                        │
│     │ Item B: $20.00 @ 5% tax = $21.00                          │                        │
│     │ Total: $32.00                                             │                        │
│     └─────────────────────────────────────────────────────────┘                          │
│                                                                                          │
│  4. EXEMPT (No tax)                                                                      │
│     ┌─────────────────────────────────────────────────────────┐                          │
│     │ Price: $10.00                                             │                        │
│     │ Tax: $0.00                                                │                        │
│     │ Total: $10.00                                             │                        │
│     └─────────────────────────────────────────────────────────┘                          │
│                                                                                          │
└──────────────────────────────────────────────────────────────────────────────────────────┘
```

Split Payment Example

```
// Customer pays with multiple methods
const splitPayment = await paymentService.processSplitPayment({
  orderId: 'order-123',
  businessId: 'biz-456',
  payments: [
    { method: PaymentMethod.GIFT_CARD, amount: 25.00, giftCardCode: 'GIFT-1234' },
    { method: PaymentMethod.LOYALTY_POINTS, amount: 10.00, loyaltyAccountId: 'loy-
789' },
    { method: PaymentMethod.CREDIT_CARD, amount: 65.00, cardLastFour: '4242' },
  ],
  processedById: 'cashier-001',
});
```

---

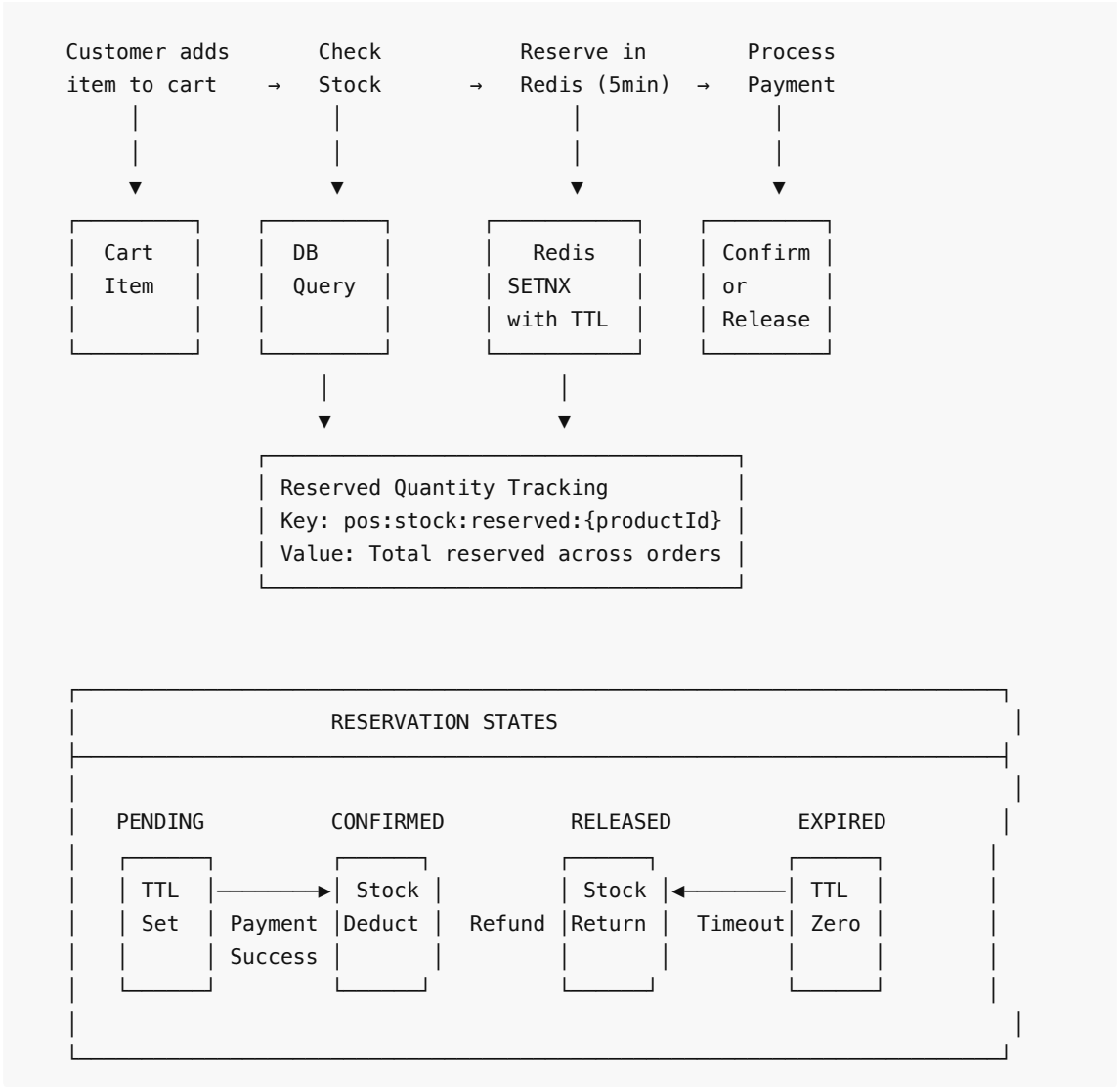
6. Inventory Management

Inventory Service (`inventory.service.ts`)

The inventory service handles stock management with support for distributed environments, preventing overselling through Redis-based reservations.

Stock Reservation Flow

```
┌──────────────────────────────────────────────────────────────────────────────────────────┐
│                          STOCK RESERVATION SYSTEM                                        │
└──────────────────────────────────────────────────────────────────────────────────────────┘
```



Key Methods

```
// Check if stock is available
async checkStockAvailability(
  businessId: string,
  items: Array<{ productId: string; quantity: number }>
): Promise<{
  available: boolean;
  unavailableItems: Array<{
    productId: string;
    requested: number;
    available: number;
  }>;
}>

// Reserve stock with TTL
async reserveStock(
  businessId: string,
```

```
  orderId: string,
  items: Array<{ productId: string; quantity: number }>,
  ttlSeconds?: number  // Default: 300 (5 minutes)
): Promise<{ reservationId: string; expiresAt: Date }>

// Confirm reservation (deduct from actual stock)
async confirmReservation(reservationId: string): Promise<void>

// Release reservation (return to available)
async releaseReservation(reservationId: string): Promise<void>

// Manual stock adjustment with automatic audit logging
async adjustInventory(
  productId: string,
  locationId: string,
  adjustment: {
    quantity: number;  // Positive or negative
    reason: string;
    referenceId?: string;
    adjustedBy: string;
    adjustedByName?: string;
  }
): Promise<void>
```

Audit Trail Integration

All inventory adjustments are automatically logged to the audit service for compliance and traceability.

```
// Automatic audit logging on inventory changes
await auditService.logInventory(
  AuditEventType.INVENTORY_ADJUSTED,
  businessId,
  adjustment.adjustedBy,          // User ID
  adjustment.adjustedByName,      // User display name
  productId,
  product.name,
  previousStock,                  // Stock before change
  newStock,                       // Stock after change
  reason,                         // Adjustment reason
  {
    adjustmentType: adjustment.type,  // SALE, RETURN, ADJUSTMENT, etc.
    referenceId: adjustment.referenceId,
    quantityChanged: adjustment.quantity,
  }
);
```

Audit Log Contents:

- User who made the change
- Timestamp of the adjustment
- Product affected
- Previous and new stock levels

- Reason for adjustment
- Reference to related transaction (order, transfer, etc.)
- Adjustment type classification

```
┌──────────────────────────────────────────────────────────────────────────────┐
│                      INVENTORY AUDIT TRAIL                                   │
├──────────────────────────────────────────────────────────────────────────────┤
│                                                                              │
│  Adjustment Types with Auto-Generated Reasons:                               │
│  ─────────────────────────────────────────────                               │
│                                                                              │
│  Type            │  Generated Reason                                         │
│  ────────────────┼─────────────────────────────────────────────────────────  │
│  SALE            │  "Stock reduced due to sale"                              │
│  RETURN          │  "Stock increased due to return"                          │
│  ADJUSTMENT      │  "Manual inventory adjustment"                            │
│  TRANSFER_OUT    │  "Stock transferred to another location"                  │
│  TRANSFER_IN     │  "Stock received from transfer"                           │
│  RECEIVED        │  "Stock received from purchase order"                     │
│  DAMAGED         │  "Stock written off as damaged"                           │
│  EXPIRED         │  "Stock removed due to expiration"                        │
│  COUNT           │  "Stock count adjustment"                                 │
│                                                                              │
│  Example Audit Entry:                                                        │
│  ────────────────────                                                        │
│  {                                                                           │
│    "eventType": "inventory.adjusted",                                        │
│    "businessId": "biz-123",                                                  │
│    "userId": "user-456",                                                     │
│    "userName": "John Smith",                                                 │
│    "productId": "prod-789",                                                  │
│    "productName": "Widget A",                                                │
│    "previousStock": 100,                                                     │
│    "newStock": 95,                                                           │
│    "reason": "Stock reduced due to sale",                                    │
│    "metadata": {                                                             │
│      "adjustmentType": "SALE",                                               │
│      "referenceId": "order-001",                                             │
│      "quantityChanged": -5                                                   │
│    },                                                                        │
│    "timestamp": "2026-01-07T12:30:00Z"                                       │
│  }                                                                           │
│                                                                              │
└──────────────────────────────────────────────────────────────────────────────┘
```

Race Condition Prevention

```
┌──────────────────────────────────────────────────────────────────────────────┐
│                   PESSIMISTIC LOCKING FOR STOCK UPDATES                      │
└──────────────────────────────────────────────────────────────────────────────┘
```

```
WITHOUT LOCKING (Race Condition):
─────────────────────────────────

Time    Thread A                Thread B                Database
──────────────────────────────────────────────────────────────────
T1      Read stock (10)                                 Stock: 10
T2                              Read stock (10)         Stock: 10
T3      Reserve 8               Reserve 5               Stock: 10
T4      Write stock (2)                                 Stock: 2
T5                              Write stock (5)         Stock: 5 ← WRONG!

Expected: 10 - 8 - 5 = -3 (should fail)
Actual: 5 (oversold!)


WITH PESSIMISTIC LOCKING:
─────────────────────────

Time    Thread A                Thread B                Database
──────────────────────────────────────────────────────────────────
T1      BEGIN TRANSACTION                               Stock: 10
T2      SELECT FOR UPDATE                               LOCKED
T3      Read stock (10)         BEGIN TRANSACTION       Stock: 10
T4                              SELECT FOR UPDATE       WAITING...
T5      Reserve 8                                       Stock: 10
T6      Write stock (2)                                 Stock: 2
T7      COMMIT                                          Stock: 2
T8                              Read stock (2)          UNLOCKED
T9                              Reserve 5 → FAIL        Stock: 2
T10                             ROLLBACK                Stock: 2 ✓
```

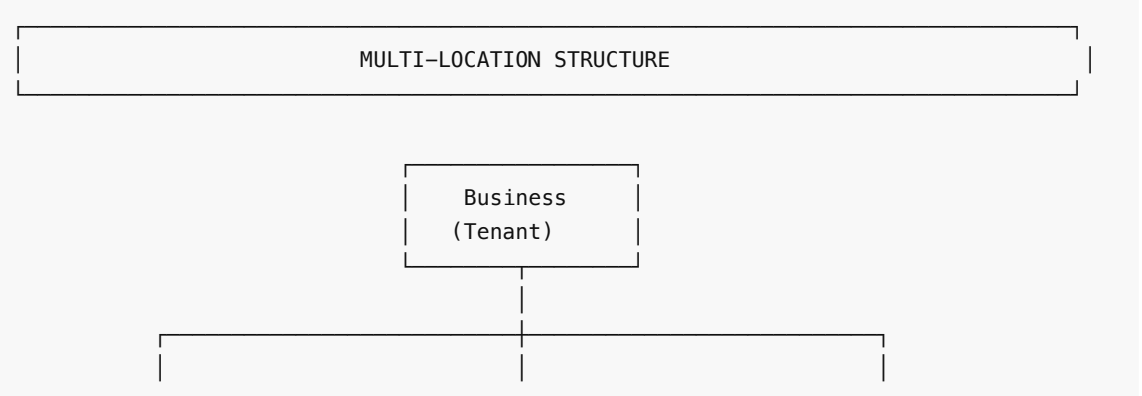
---

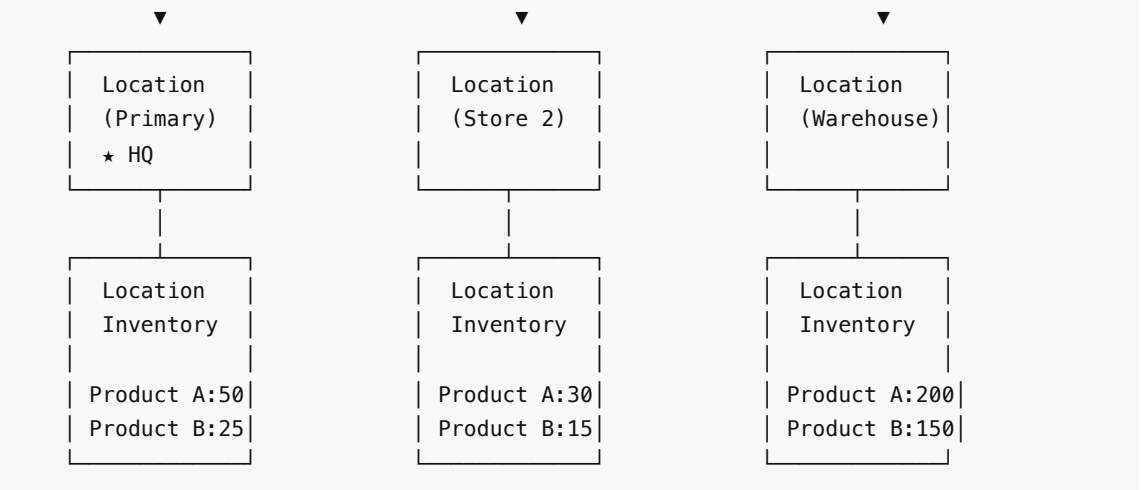
7. Multi-Location Support

Location Service (`location.service.ts`)

Enables businesses to manage multiple store locations with independent inventory, stock transfers between locations, and per-location configuration.

Location Hierarchy



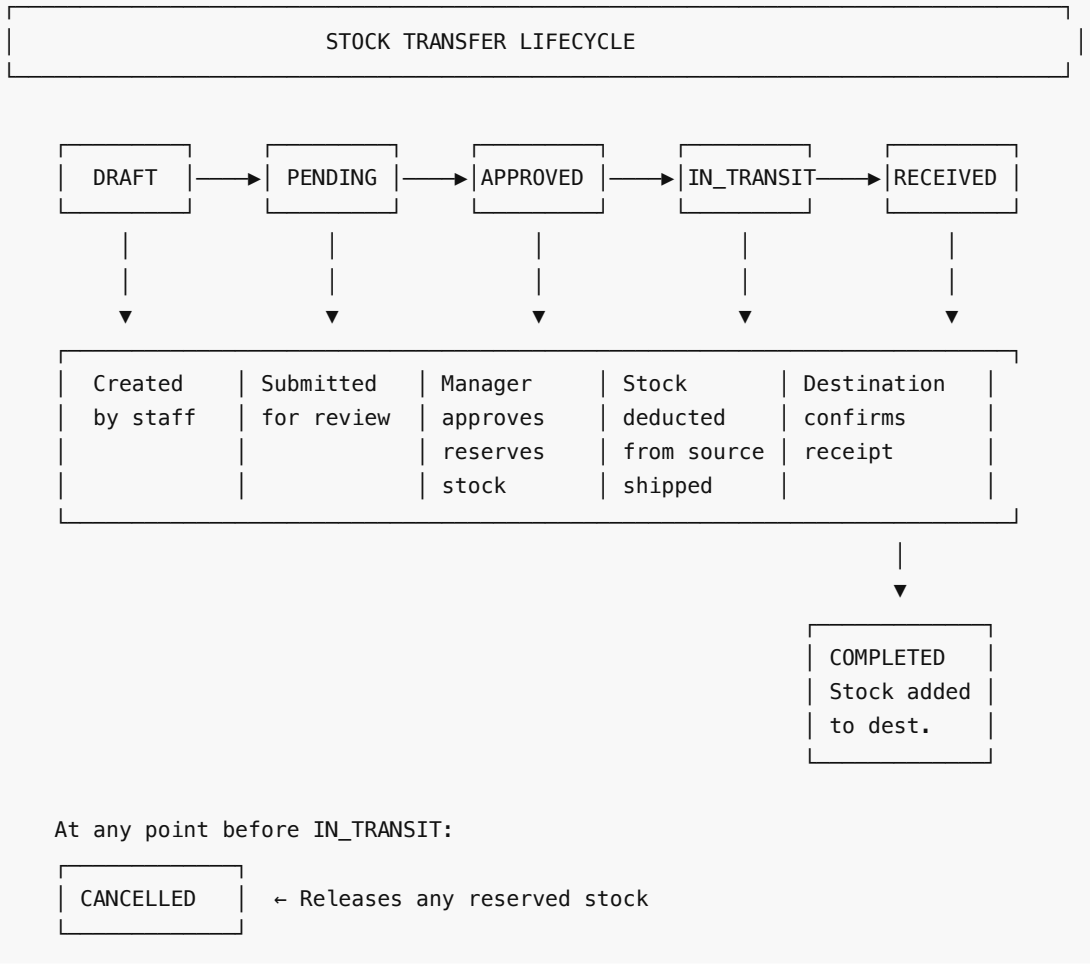


Location Entity Properties

```
interface Location {
  id: string;
  code: string;              // "STORE001"
  name: string;              // "Downtown Store"
  type: LocationType;        // RETAIL_STORE, WAREHOUSE, KIOSK, etc.
  status: LocationStatus;    // ACTIVE, INACTIVE, TEMPORARILY_CLOSED

  // Address
  addressLine1: string;
  city: string;
  state: string;
  country: string;

  // Contact
  phone: string;
  email: string;

  // Operating Hours (JSONB)
  operatingHours: {
    monday: { open: "09:00", close: "21:00" };
    tuesday: { open: "09:00", close: "21:00" };
    // ...
    sunday: { closed: true };
  };

  // Configuration
  timezone: string;          // "America/New_York"
  taxRate: number;           // Location-specific tax rate
  isPrimary: boolean;        // Main/HQ location
  trackInventory: boolean;   // Enable per-location stock
  allowNegativeStock: boolean;

  // Relations
```

```
  managerId: string;          // User responsible for location
}
```

Stock Transfer Workflow



Transfer Service Methods

```
// Create transfer request
async createTransfer(dto: CreateTransferDTO): Promise<StockTransfer>

// Submit for approval
async submitTransfer(transferId: string): Promise<StockTransfer>

// Approve (reserves stock at source)
async approveTransfer(transferId: string, approvedById: string):
Promise<StockTransfer>

// Ship (deducts from source)
async shipTransfer(transferId: string, shippingInfo: {
  shippingMethod?: string;
  trackingNumber?: string;
```

```
  expectedArrival?: Date;
}): Promise<StockTransfer>

// Receive at destination (adds to destination)
async receiveTransfer(transferId: string, receivedById: string, items: Array<{
  productId: string;
  quantityReceived: number;  // May differ from shipped
  notes?: string;
}>): Promise<StockTransfer>

// Complete transfer
async completeTransfer(transferId: string): Promise<StockTransfer>
```

Inventory Query Across Locations

```
// Get total inventory across all locations
const total = await locationService.getTotalInventory(businessId, productId);
// Returns: 280 (sum of all location quantities)

// Get breakdown by location
const breakdown = await locationService.getInventoryAcrossLocations(businessId,
productId);
// Returns:
// [
//   { location: "Downtown Store", quantity: 50, reserved: 5, available: 45 },
//   { location: "Mall Kiosk", quantity: 30, reserved: 2, available: 28 },
//   { location: "Main Warehouse", quantity: 200, reserved: 0, available: 200 },
// ]
```

---

8. Barcode & SKU System

Barcode Service (`barcode.service.ts`)

Comprehensive barcode management supporting multiple formats, automatic validation, SKU generation, and batch/lot tracking.

Supported Barcode Types

```
┌──────────────────────────────────────────────────────────────────────────┐
│                          BARCODE FORMAT SUPPORT                          │
└──────────────────────────────────────────────────────────────────────────┘


┌──────────────┬──────────────┬──────────────────────────────────────────────┐
│   Type       │  Digits      │  Description & Usage                         │
├──────────────┼──────────────┼──────────────────────────────────────────────┤
│  UPC-A       │    12        │  US/Canada retail. Format: XXXXXXXXXXX       │
│  UPC-E       │     8        │  Compressed UPC for small items              │
│  EAN-13      │    13        │  International standard. Format: XXXXXXXXXXXX│
│  EAN-8       │     8        │  Short form for small products               │
│  Code 39     │  1-43        │  Alphanumeric. Used in automotive, defense   │
```

```
|  Code 128    |  1-80     |  High density. Logistics, shipping                    |
|  ITF-14      |    14     |  Cartons and pallets                                  |
|  QR Code     |Variable   |  2D code. Mobile scanning, URLs                       |
|  Data Matrix|Variable   |  2D code. Small items, electronics                    |
|  PDF-417     |Variable   |  2D code. ID cards, tickets                           |
|  Internal    | Custom    |  Store-specific codes                                 |
└──────────────┴───────────┴──────────────────────────────────────────────────────┘
```

Barcode Validation

```typescript
// Validate barcode with automatic type detection
const result = barcodeService.validateBarcode('012345678905');
// Returns:
// {
//   valid: true,
//   type: BarcodeType.UPC_A,
//   error: undefined
// }

// Validate with specific type
const result = barcodeService.validateBarcode('5901234123457', BarcodeType.EAN_13);
// Performs check digit validation

// Check digit calculation (Modulo 10)
// For EAN-13: 5901234123457
// Odd positions (1,3,5...): 5+0+2+4+2+4+7 = 24
// Even positions (2,4,6...): 9+1+3+1+3+5 = 22 × 3 = 66
// Sum: 24 + 66 = 90
// Check digit: (10 - (90 mod 10)) mod 10 = 0 ✗ (actual is 7)
```

SKU Generation

```
┌──────────────────────────────────────────────────────────────────────────┐
│                          SKU GENERATION FORMATS                          │
└──────────────────────────────────────────────────────────────────────────┘

    FORMAT 1: SEQUENTIAL
    ────────────────────
    Pattern: {prefix}{sequence}{suffix}
    Example: SKU-00001, SKU-00002, SKU-00003


    FORMAT 2: CATEGORY PREFIX
    ─────────────────────────
    Pattern: {category_code}-{sequence}
    Example: ELEC-00001, CLTH-00042, FOOD-00103


    FORMAT 3: CUSTOM PATTERN
    ────────────────────────
    Tokens: {SEQ}, {SEQ:n}, {YEAR}, {MONTH}, {CAT}
```

```
Pattern: "PRD-{YEAR}{MONTH}-{CAT}-{SEQ:4}"
Example: PRD-202601-ELEC-0001


┌─────────────────────────────────────────────────────────┐
│                SKU Configuration Entity                 │
├─────────────────────────────────────────────────────────┤
│  format: SKUFormat.CATEGORY_PREFIX                      │
│  nextSequence: 1                                        │
│  sequencePadding: 5                                     │
│  prefix: "SKU-"                                         │
│  suffix: null                                           │
│  categoryCodeLength: 4                                  │
│  allowManual: true                                      │
│  requireUnique: true                                    │
└─────────────────────────────────────────────────────────┘
```

Batch/Lot Tracking

```
┌──────────────────────────────────────────────────────────────────────────┐
│                          BATCH TRACKING SYSTEM                           │
└──────────────────────────────────────────────────────────────────────────┘

Product: "Organic Milk 1L"

┌─────────────┬─────────────┬─────────────┬─────────────┬─────────────┐
│  Batch #    │  Lot #      │  Qty        │  Expiry     │  Status     │
├─────────────┼─────────────┼─────────────┼─────────────┼─────────────┤
│  BAT-001    │  LOT-A      │  50         │  2026-01-15 │  AVAILABLE  │
│  BAT-002    │  LOT-A      │  30         │  2026-01-20 │  AVAILABLE  │
│  BAT-003    │  LOT-B      │  100        │  2026-02-01 │  AVAILABLE  │
│  BAT-004    │  LOT-B      │  0          │  2025-12-01 │  EXPIRED    │
└─────────────┴─────────────┴─────────────┴─────────────┴─────────────┘


FEFO ALLOCATION (First Expire, First Out):
──────────────────────────────────────────

Customer orders 60 units:

Step 1: Sort by expiration date (ASC)
        BAT-001 (Jan 15) → BAT-002 (Jan 20) → BAT-003 (Feb 01)

Step 2: Allocate from earliest expiring
        BAT-001: 50 units (depleted)
        BAT-002: 10 units (20 remaining)

Result: [
  { batch: "BAT-001", quantity: 50 },
  { batch: "BAT-002", quantity: 10 }
]
```

SVG Barcode Generation

The barcode service provides native SVG generation for multiple barcode formats, enabling high-quality barcode rendering without external dependencies.

```
// Generate EAN-13 barcode
const ean13Svg = await barcodeService.generateEAN13('5901234123457');
// Returns: Complete SVG with proper L/R encoding patterns and check digit
validation

// Generate Code 128 barcode (alphanumeric support)
const code128Svg = await barcodeService.generateCode128('PROD-12345');
// Returns: SVG using Code 128B encoding with optimized bar patterns

// Generate Code 39 barcode (alphanumeric with special chars)
const code39Svg = await barcodeService.generateCode39('ABC-123');
// Returns: SVG with start/stop characters and proper spacing

// Generate QR Code (data matrix)
const qrSvg = await
barcodeService.generateQRCodeSVG('https://store.com/product/123');
// Returns: SVG with QR code matrix pattern
```

SVG Output Features:

- Scalable vector graphics for crisp printing at any resolution
- Configurable dimensions (width, height, barWidth)
- Human-readable text below barcode (optional)
- White background with proper quiet zones
- Monospace font for optimal readability

```
┌──────────────────────────────────────────────────────────────────────────┐
│                        BARCODE SVG GENERATION                            │
├──────────────────────────────────────────────────────────────────────────┤
│                                                                          │
│  EAN-13 Structure:                                                       │
│  ┌─────┬──────────────────────────────────┬──────────────────────────┐    │
│  │Start│    Left Group (L/G patterns)     │    Right Group (R patterns)    │   │
│  │Guard│    First 6 digits                │    Last 6 digits + check       │   │
│  │ ||| │    Based on first digit parity   │    Always R encoding           │   │
│  └─────┴──────────────────────────────────┴──────────────────────────┘    │
│                                                                          │
│  Code 128 Features:                                                      │
│  • Full ASCII support (Code A/B/C switching)                             │
│  • Optimal density for alphanumeric data                                 │
│  • Modulo 103 check digit calculation                                    │
│                                                                          │
│  QR Code Support:                                                        │
│  • Version 1-4 auto-sizing based on data length                          │
│  • Error correction level M (15% recovery)                               │
│  • Finder patterns and timing patterns included                          │
```

```
│                                                                              │
└──────────────────────────────────────────────────────────────────────────────┘
```

Barcode Scan Logging

```
// Log scan for analytics and audit
const scanResult = await barcodeService.lookupByBarcode(
  businessId,
  '012345678905',
  {
    scanType: ScanType.SALE,
    scannedBy: 'user-123',
    locationId: 'loc-456',
    ipAddress: '192.168.1.100',
  }
);

// Returns:
// {
//   matched: true,
//   barcode: '012345678905',
//   type: BarcodeType.UPC_A,
//   product: { id: '...', name: 'Product Name', ... },
//   productBarcode: { id: '...', isPrimary: true, ... },
//   batches: [{ batchNumber: 'BAT-001', ... }],
// }

// Scan statistics
const stats = await barcodeService.getScanStatistics(businessId, {
  startDate: new Date('2026-01-01'),
  endDate: new Date('2026-01-31'),
});
// Returns: { totalScans, matchedScans, unmatchedBarcodes, topProducts, ... }
```

---

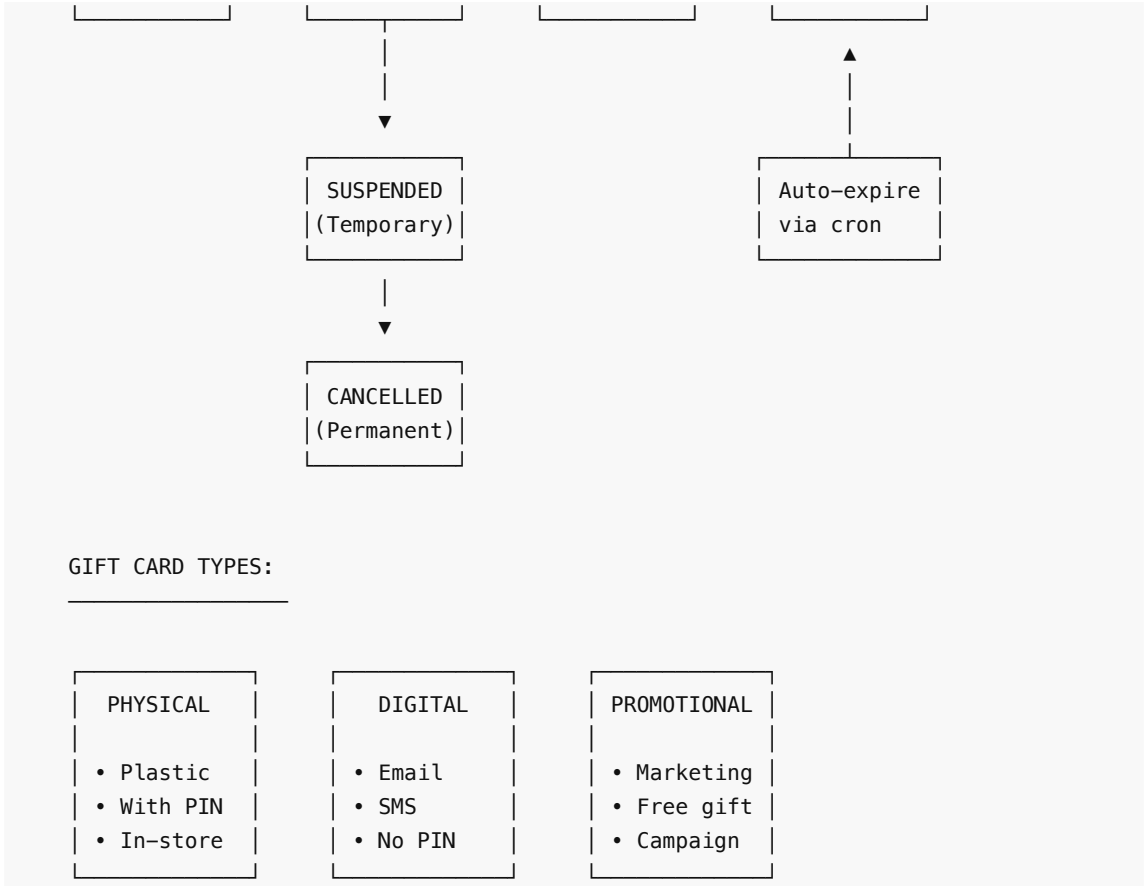
9. Gift Card System

Gift Card Service (`giftcard.service.ts`)

Complete gift card management including issuance, redemption, reload, and balance tracking with full transaction history.

Gift Card Lifecycle

```
┌──────────────────────────────────────────────────────────────────────────────┐
│                          GIFT CARD LIFECYCLE                                 │
└──────────────────────────────────────────────────────────────────────────────┘


    ┌──────────┐     ┌──────────┐     ┌──────────┐     ┌──────────┐
    │  ISSUED  │────►│  ACTIVE  │────►│ REDEEMED │     │ EXPIRED  │
    │          │     │          │     │ (Balance 0)    │          │
```



Gift Card Code Format

```
Code Generation: XXXX-XXXX-XXXX-XXXX
──────────────────────────────────────

Characters: A-Z, 0-9 (36 possibilities per position)
Total combinations: 36^16 = 7.96 × 10^24

Example: ABCD-1234-EFGH-5678


PIN Generation (Physical cards only):
──────────────────────────────────────

Format: 4-digit numeric
Range: 1000-9999

Example: 4521
```

Transaction Flow

```
// Issue new gift card
const giftCard = await giftCardService.issueGiftCard({
  businessId: 'biz-123',
```

```
  issuedById: 'staff-456',
  type: GiftCardType.DIGITAL,
  initialValue: 50.00,
  expiresAt: new Date('2027-01-01'),
  recipientEmail: 'customer@email.com',
  recipientName: 'John Doe',
  message: 'Happy Birthday!',
});

// Check balance
const balance = await giftCardService.checkBalance(
  'ABCD-1234-EFGH-5678',
  'biz-123',
  '4521'  // PIN for physical cards
);
// Returns: { valid: true, balance: 50.00, status: 'active', expiresAt: Date }

// Redeem at checkout
const redemption = await giftCardService.redeem({
  code: 'ABCD-1234-EFGH-5678',
  businessId: 'biz-123',
  amount: 25.00,
  orderId: 'order-789',
  processedById: 'cashier-001',
});
// Returns: { success: true, amountApplied: 25.00, remainingBalance: 25.00 }

// Reload gift card
await giftCardService.reload({
  giftCardId: 'gc-123',
  businessId: 'biz-123',
  amount: 30.00,
  processedById: 'cashier-001',
});
// Balance now: 55.00
```

Transaction Types

```
┌──────────────────────────────────────────────────────────────────────────┐
│                        GIFT CARD TRANSACTION TYPES                       │
├──────────────┬───────────────────────────────────────────────────────────┤
│    Type      │  Description                                              │
├──────────────┼───────────────────────────────────────────────────────────┤
│   issue      │  Initial card issuance with value                         │
│   redeem     │  Used at checkout (decreases balance)                     │
│   reload     │  Add more value (increases balance)                       │
│   refund     │  Return to gift card instead of original payment          │
│   adjust     │  Manual balance adjustment by admin                       │
│   expire     │  System marks expired cards                               │
└──────────────┴───────────────────────────────────────────────────────────┘
```

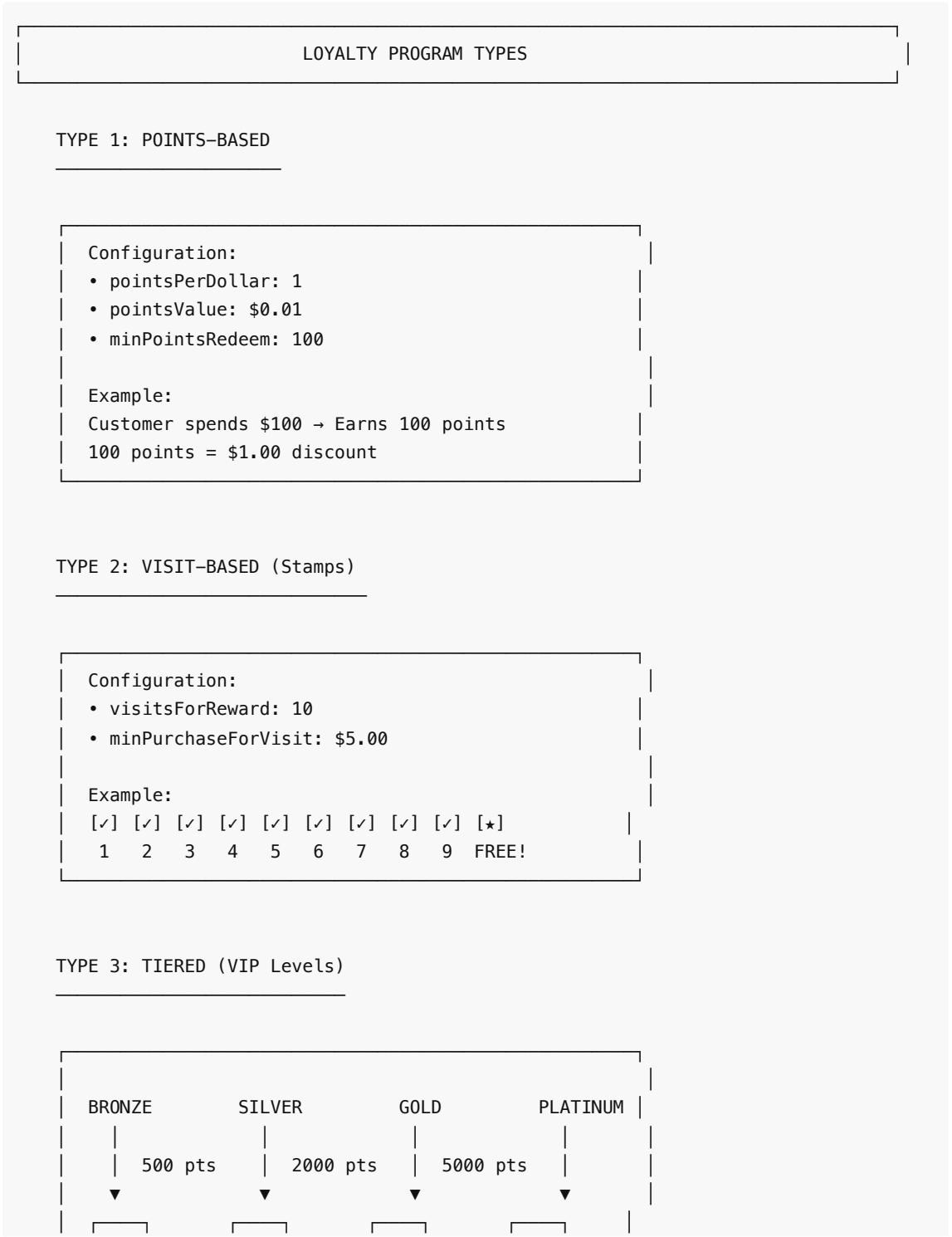
---

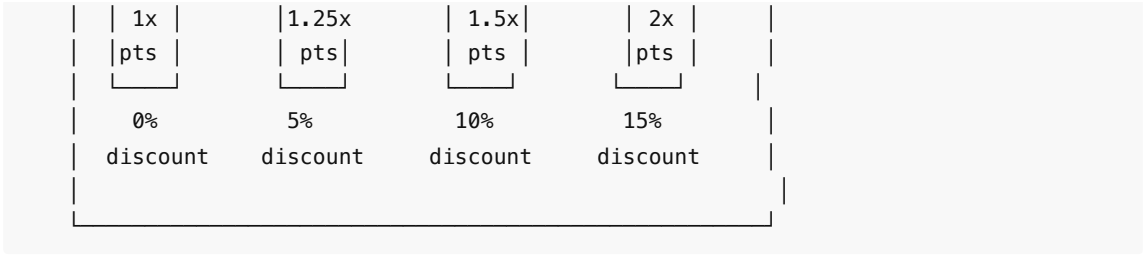
10. Loyalty Program

Loyalty Service (`loyalty.service.ts`)

Full-featured loyalty program supporting points-based rewards, visit stamps, tiered memberships, and automated rewards.

Program Types





Tier Configuration

```typescript
interface LoyaltyTier {
  name: string;           // "Gold"
  color: string;          // "#FFD700"
  icon: string;           // "crown"
  minPoints: number;      // 2000 (lifetime points to reach tier)
  maxPoints: number;      // 4999

  // Benefits
  pointsMultiplier: number;    // 1.5 = 50% bonus points
  discountPercentage: number;  // 10 = 10% off all purchases
  freeShipping: boolean;
  earlyAccess: boolean;
  exclusiveOffers: boolean;

  customBenefits: [
    { name: "Birthday Gift", description: "Free item on birthday" },
    { name: "Priority Support", description: "Dedicated support line" },
  ];
}
```

Points Transaction Types

LOYALTY TRANSACTION TYPES	
Type	Description
earn	Points earned from purchase
redeem	Points spent on reward
bonus	Manual bonus points added
signup	Enrollment bonus
birthday	Birthday reward points
referral	Points for referring new customer
tier_bonus	Bonus for reaching new tier
adjustment	Manual correction
expire	Points expired (if expiry enabled)
refund	Points returned due to order refund

Reward Types

```
enum RewardType {
  DISCOUNT_PERCENTAGE = 'discount_percentage',  // 10% off
  DISCOUNT_FIXED = 'discount_fixed',            // $5 off
  FREE_PRODUCT = 'free_product',                // Free item
  FREE_SHIPPING = 'free_shipping',              // Free delivery
  BONUS_POINTS = 'bonus_points',                // 2x points day
  CUSTOM = 'custom',                            // Store-specific
}

// Example rewards catalog
const rewards = [
  { name: "$5 Off", type: RewardType.DISCOUNT_FIXED, pointsCost: 500, value: 5 },
  { name: "10% Off", type: RewardType.DISCOUNT_PERCENTAGE, pointsCost: 1000, value:
10 },
  { name: "Free Coffee", type: RewardType.FREE_PRODUCT, pointsCost: 200, productId:
"prod-coffee" },
];
```

Points Calculation with Tier Multiplier

```
┌─────────────────────────────────────────────────────────────────────────┐
│                    POINTS EARNING CALCULATION                           │
└─────────────────────────────────────────────────────────────────────────┘

   Customer: Gold Tier (1.5x multiplier)
   Purchase: $100.00
   Base Rate: 1 point per $1

   Calculation:
   ──────────────
   Base Points = $100 × 1 = 100 points
   Tier Bonus  = 100 × (1.5 - 1) = 50 points
   Total       = 100 + 50 = 150 points


   With Promotional Multiplier:
   ──────────────────────────
   Base Rate: 1 point per $1
   Tier Multiplier: 1.5x
   Promo Multiplier: 2x (Double Points Day)

   Base Points = $100 × 1 = 100 points
   After Tier  = 100 × 1.5 = 150 points
   After Promo = 150 × 2 = 300 points
```

---

11. Supplier & Purchase Orders

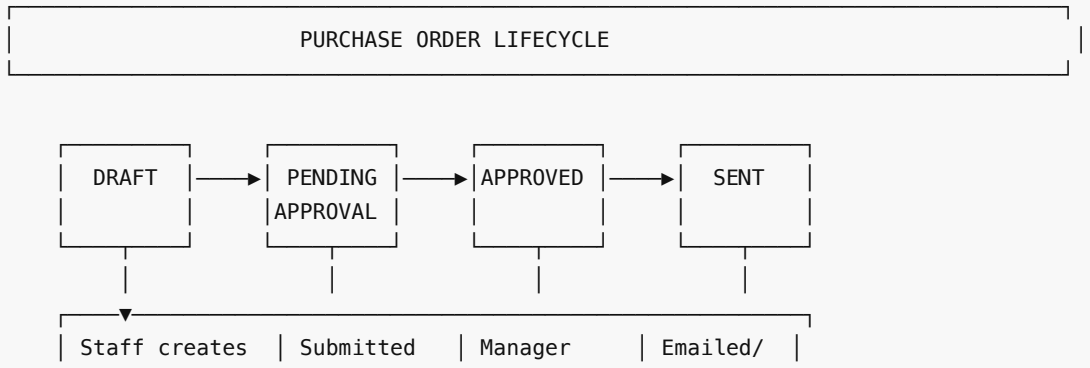
Supplier Service (`supplier.service.ts`)

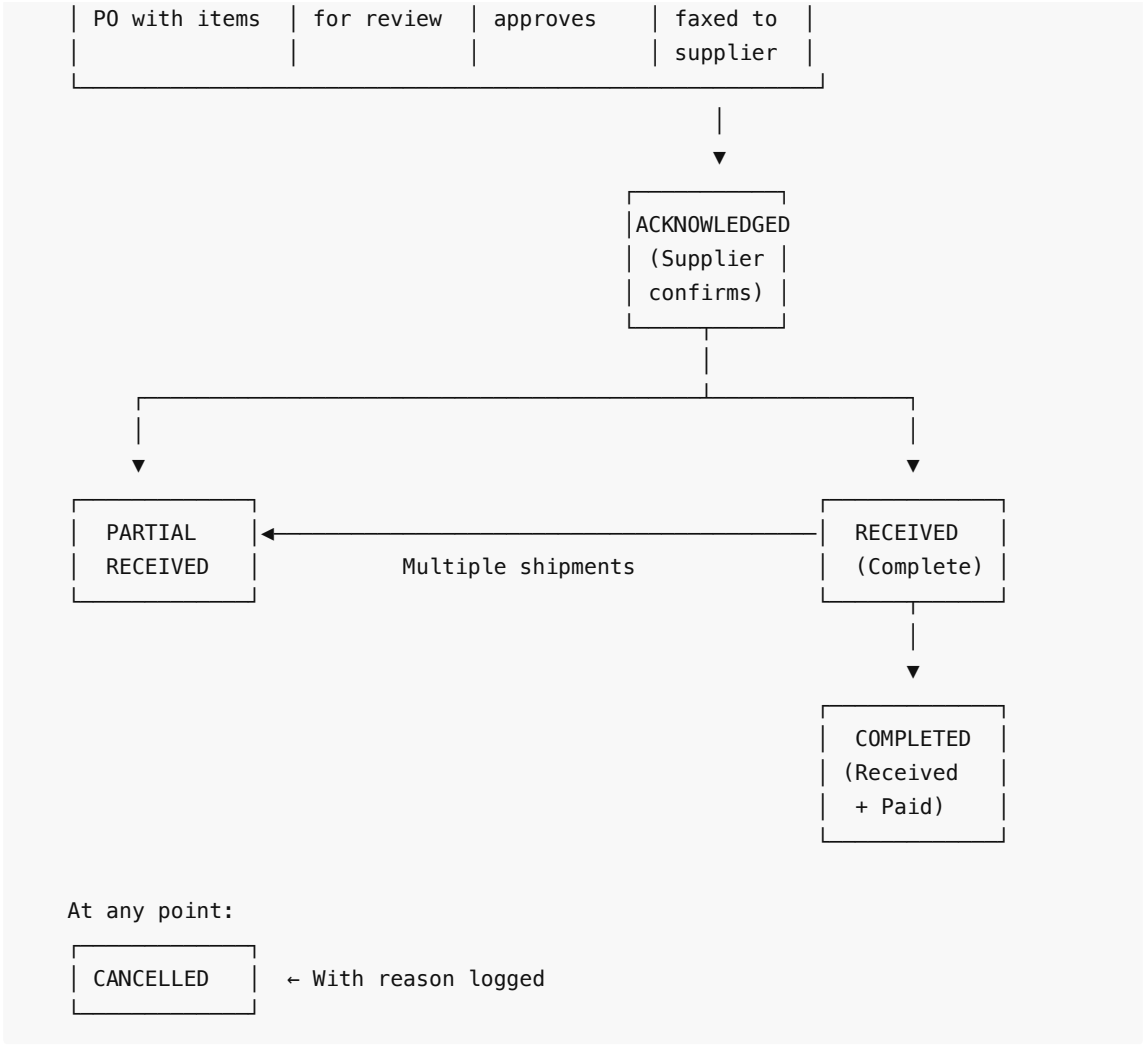
Complete supplier relationship management and purchase order lifecycle from creation through receiving and payment.

Supplier Entity

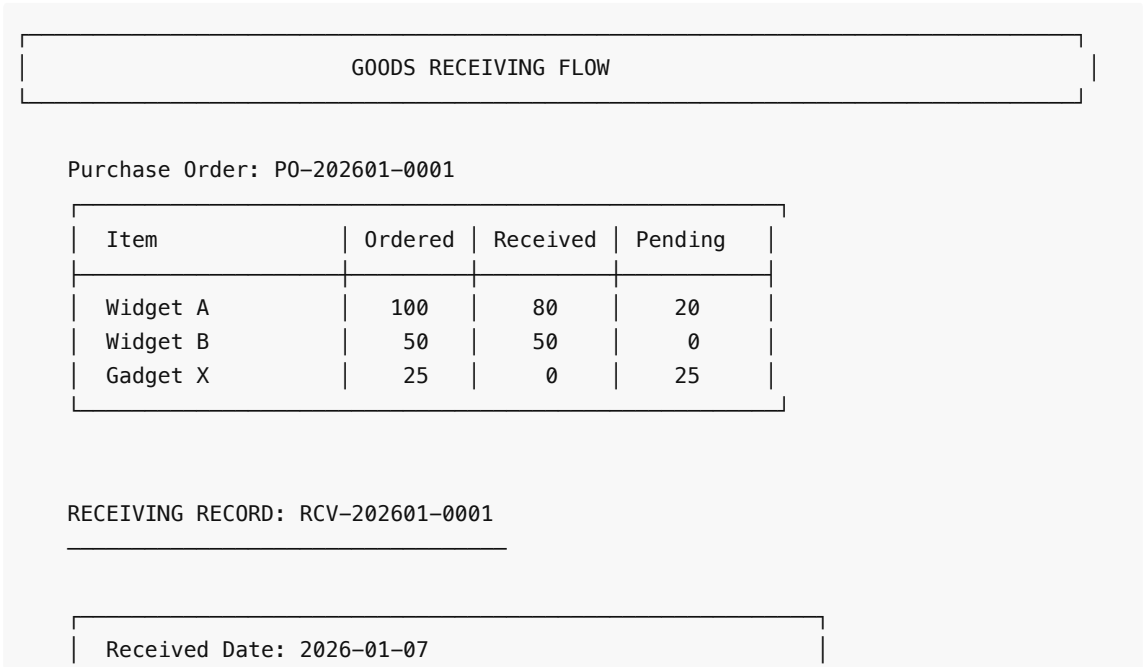
```
interface Supplier {
  id: string;
  code: string;              // "SUP001"
  name: string;              // "ABC Distributors"
  companyName: string;
  status: SupplierStatus;    // ACTIVE, INACTIVE, BLOCKED

  // Contact
  contactName: string;
  email: string;
  phone: string;

  // Address
  addressLine1: string;
  city: string;
  country: string;

  // Financial
  paymentTerms: PaymentTerms;  // NET_30, NET_60, COD, etc.
  currency: string;            // "USD"
  creditLimit: number;
  currentBalance: number;      // Outstanding amount

  // Performance
  rating: number;              // 0-5 stars
  defaultLeadTimeDays: number; // Expected delivery time
  minOrderAmount: number;

  // Banking
  bankName: string;
  bankAccount: string;
}
```

Purchase Order Lifecycle





Receiving Process



```
│  Delivery Note: DN-12345                                  │
│  Carrier: FedEx                                           │
│  Quality Check: PASSED                                    │
│                                                           │
│  Items Received:                                          │
│  ┌─────────────────────────────────────────────────────┐  │
│  │ Widget A: 80 units                                  │  │
│  │   - Batch: BAT-2026-001                             │  │
│  │   - Lot: LOT-A                                      │  │
│  │   - Expiry: 2027-01-01                              │  │
│  │   - Bin Location: A-01-03                           │  │
│  │                                                     │  │
│  │ Widget B: 50 units                                  │  │
│  │   - Batch: BAT-2026-002                             │  │
│  │   - Quality Issue: 2 units rejected (damaged)       │  │
│  └─────────────────────────────────────────────────────┘  │
└───────────────────────────────────────────────────────────┘


INVENTORY UPDATE (Automatic):
─────────────────────────────


Location: Main Warehouse
┌──────────────┬──────────────┬──────────────┐
│   Product    │   Before     │   After      │
├──────────────┼──────────────┼──────────────┤
│  Widget A    │     50       │    130       │
│  Widget B    │     20       │     68       │
└──────────────┴──────────────┴──────────────┘
```

Supplier Payment Terms

```
┌──────────────────────────────────────────────────────────────────────────────┐
│                          PAYMENT TERMS                                       │
├──────────────────┬───────────────────────────────────────────────────────────┤
│     Term         │  Description                                              │
├──────────────────┼───────────────────────────────────────────────────────────┤
│   IMMEDIATE      │  Payment upon receipt of goods                            │
│   NET_15         │  Payment due within 15 days                               │
│   NET_30         │  Payment due within 30 days                               │
│   NET_45         │  Payment due within 45 days                               │
│   NET_60         │  Payment due within 60 days                               │
│   NET_90         │  Payment due within 90 days                               │
│   COD            │  Cash on delivery                                         │
│   PREPAID        │  Payment before shipment                                  │
│   CUSTOM         │  Custom number of days                                    │
└──────────────────┴───────────────────────────────────────────────────────────┘
```

Reorder Suggestions

```
// Automatic reorder suggestions based on stock levels
const suggestions = await supplierService.getReorderSuggestions(businessId);

// Returns:
[
  {
    productId: "prod-123",
    currentStock: 15,
    reorderPoint: 20,
    suggestedQuantity: 100,  // Based on reorderQuantity or 2x reorderPoint
    preferredSupplier: { id: "sup-456", name: "ABC Distributors" },
    lastCost: 12.50,
  },
  // ...
]
```

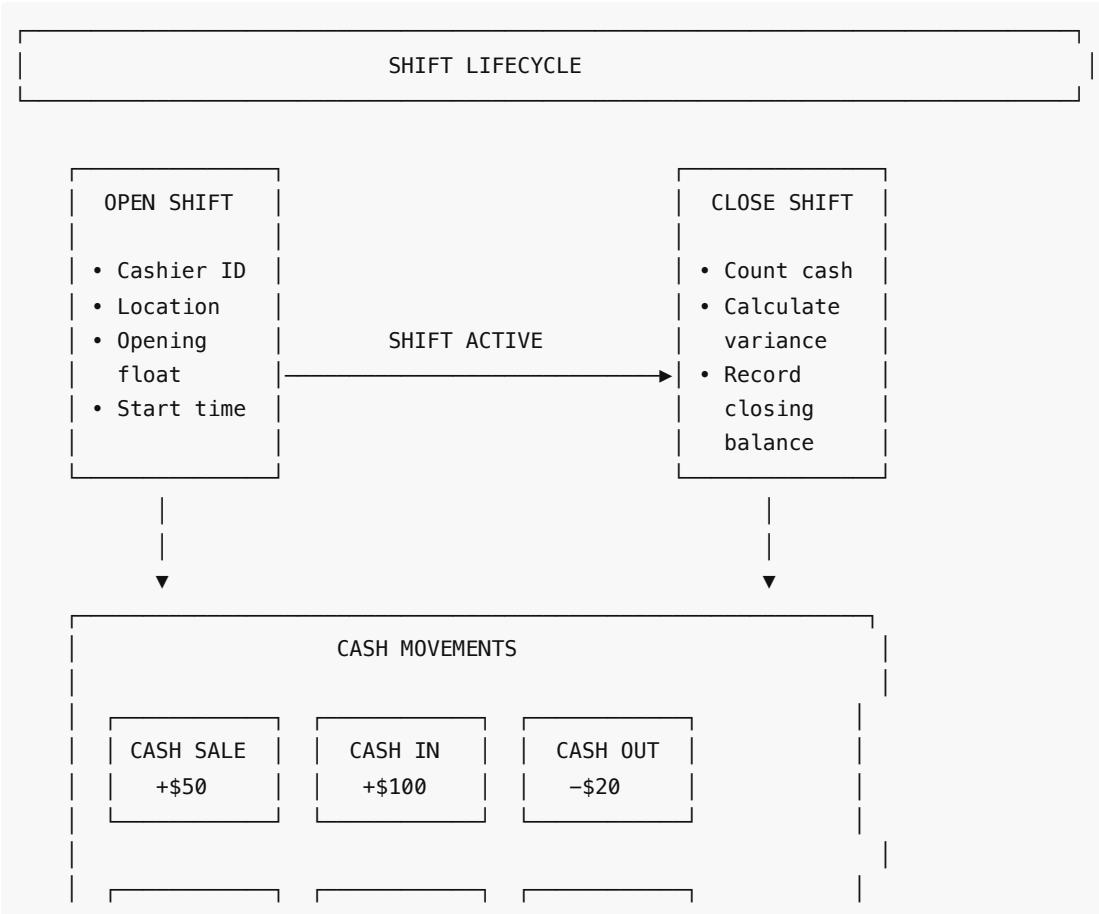
---

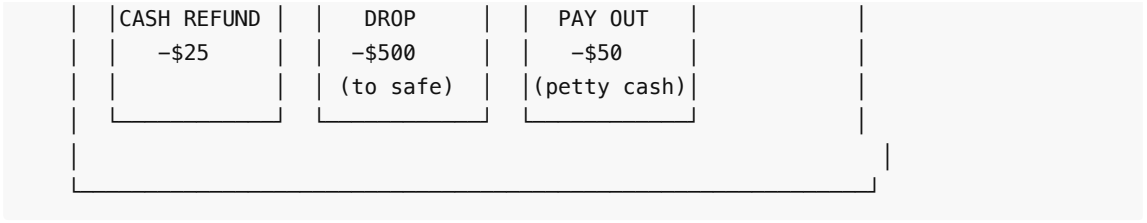
12. Shift & Cash Management

Shift Service (`shift.service.ts`)

Track cash register shifts, cash movements, and drawer balances for accountability and reconciliation.

Shift Lifecycle





Cash Movement Types

```
enum CashMovementType {
  OPENING_FLOAT = 'opening_float',  // Starting cash in drawer
  CASH_SALE = 'cash_sale',          // Cash received from sale
  CASH_REFUND = 'cash_refund',      // Cash given for refund
  CASH_IN = 'cash_in',              // Additional cash added (change)
  CASH_OUT = 'cash_out',            // Cash removed (correction)
  PAY_OUT = 'pay_out',              // Petty cash expenses
  DROP = 'drop',                    // Cash dropped to safe
  CLOSING_COUNT = 'closing_count',  // Final count at close
}
```

Shift Summary

```
┌──────────────────────────────────────────────────────────────────────────────┐
│                           SHIFT SUMMARY                                      │
├──────────────────────────────────────────────────────────────────────────────┤
│                                                                              │
│  Shift: SHF-2026-01-07-001                                                   │
│  Cashier: John Smith                                                         │
│  Location: Downtown Store - Register 1                                       │
│  Duration: 09:00 - 17:00 (8 hours)                                           │
│                                                                              │
│  ┌──────────────────────────────────────────────────────────────────────┐    │
│  │                          CASH SUMMARY                                │    │
│  ├──────────────────────────────────────────────────────────────────────┤    │
│  │  Opening Float:                                    $200.00           │    │
│  │                                                                      │    │
│  │  + Cash Sales:                                   +$1,250.00          │    │
│  │  + Cash In:                                         +$50.00          │    │
│  │  - Cash Refunds:                                    -$75.00          │    │
│  │  - Cash Out:                                        -$20.00          │    │
│  │  - Pay Outs:                                        -$35.00          │    │
│  │  - Drops:                                          -$800.00          │    │
│  │  ──────────────────────────────────────────────────────              │    │
│  │  Expected in Drawer:                               $570.00           │    │
│  │  Actual Count:                                     $568.50           │    │
│  │  ──────────────────────────────────────────────────────              │    │
│  │  Variance:                                          -$1.50           │    │
│  └──────────────────────────────────────────────────────────────────────┘    │
│                                                                              │
│  ┌──────────────────────────────────────────────────────────────────────┐    │
│  │                          SALES SUMMARY                               │    │
```

```
│  ├──────────────────────────────────────────────────────────────────────┤│
│  │  Total Sales:          $3,450.00                                     ││
│  │  Total Orders:         47                                            ││
│  │  Average Order:        $73.40                                        ││
│  │                                                                      ││
│  │  Payment Breakdown:                                                  ││
│  │    Cash:               $1,250.00  (36%)                              ││
│  │    Credit Card:        $1,850.00  (54%)                              ││
│  │    Debit Card:           $275.00  (8%)                               ││
│  │    Gift Card:             $75.00  (2%)                               ││
│  └──────────────────────────────────────────────────────────────────────┘│
│                                                                          │
└──────────────────────────────────────────────────────────────────────────┘
```

---

13. End-of-Day Reconciliation

EOD Service (`eod.service.ts`)

Comprehensive end-of-day reporting that aggregates all business activities for daily close procedures.

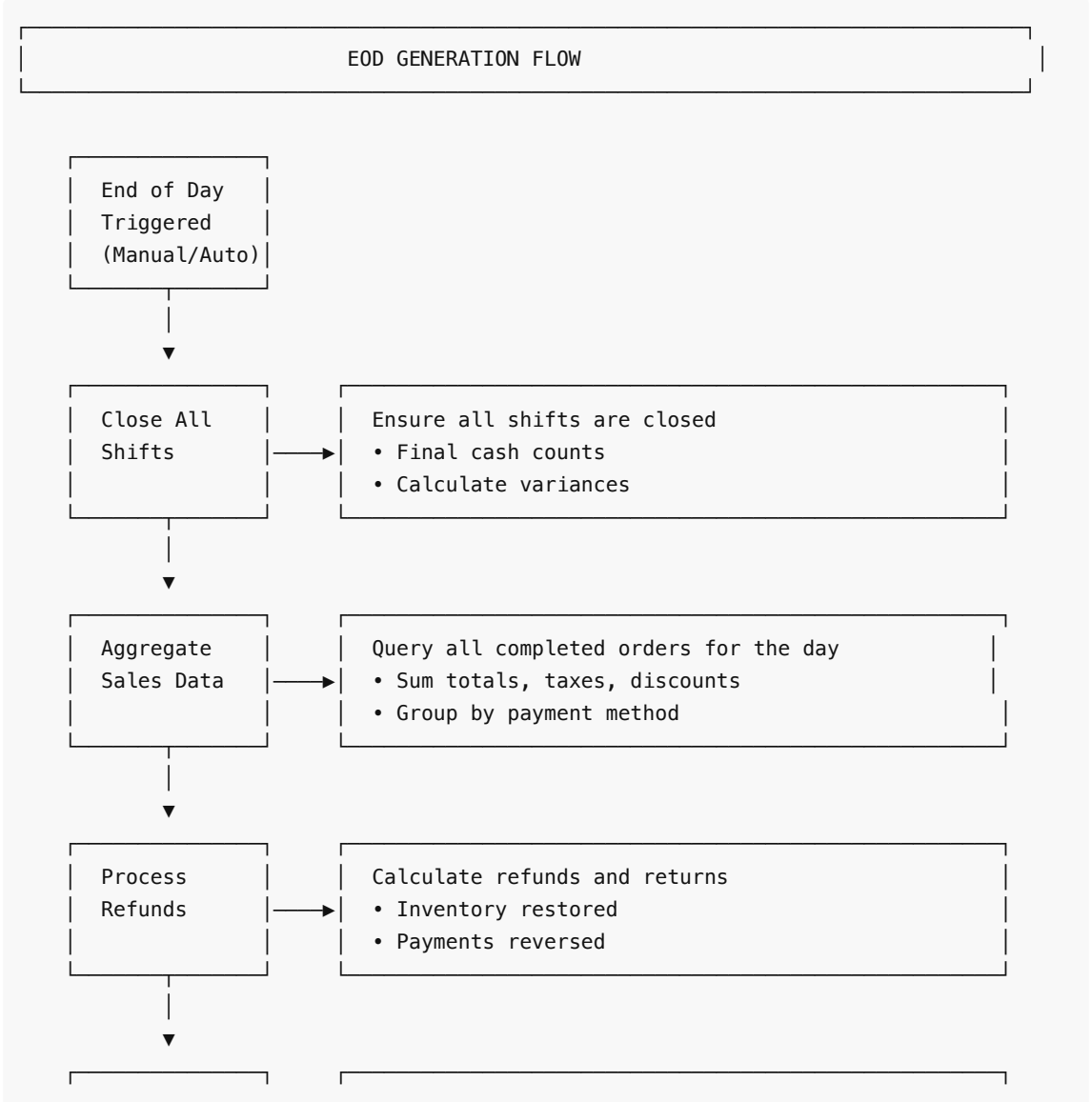
EOD Report Structure

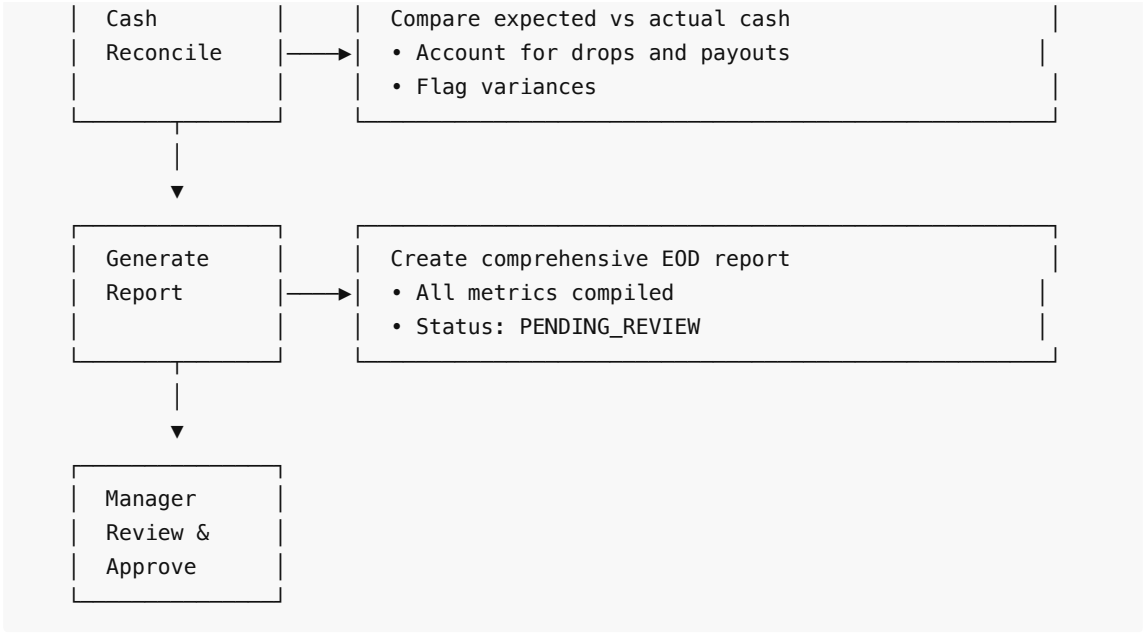
```typescript
interface EODReport {
  id: string;
  businessId: string;
  locationId: string;
  reportDate: Date;
  status: EODReportStatus;  // DRAFT, PENDING_REVIEW, REVIEWED, APPROVED

  // Sales Metrics
  salesSummary: {
    totalSales: number;
    totalOrders: number;
    averageOrderValue: number;
    totalTax: number;
    totalDiscount: number;
    netSales: number;
  };

  // Payment Breakdown
  paymentBreakdown: {
    cash: { count: number; amount: number };
    creditCard: { count: number; amount: number };
    debitCard: { count: number; amount: number };
    giftCard: { count: number; amount: number };
    loyalty: { count: number; amount: number };
    other: { count: number; amount: number };
  };

  // Refunds
  refundsSummary: {
```

```
  totalRefunds: number;
  refundCount: number;
  refundsByMethod: Record<string, number>;
};

// Cash Reconciliation
cashReconciliation: {
  expectedCash: number;
  actualCash: number;
  variance: number;
  drops: number;
  payOuts: number;
};

// Shifts Summary
shiftsSummary: Array<{
  shiftId: string;
  cashierName: string;
  startTime: Date;
  endTime: Date;
  salesTotal: number;
  cashVariance: number;
}>;

// Category Performance
categoryBreakdown: Array<{
  categoryId: string;
  categoryName: string;
  quantity: number;
  revenue: number;
}>;

// Top Products
topProducts: Array<{
  productId: string;
  productName: string;
  quantity: number;
  revenue: number;
}>;

// Hourly Sales
hourlySales: Array<{
  hour: number;
  orders: number;
  sales: number;
}>;

// Inventory Alerts
inventorySnapshot: {
  lowStockCount: number;
  outOfStockCount: number;
  adjustmentsCount: number;
```

```
  };

  // Customer Metrics
  customerMetrics: {
    newCustomers: number;
    returningCustomers: number;
    loyaltyPointsIssued: number;
    loyaltyPointsRedeemed: number;
  };

  // Approval
  reviewedById: string;
  reviewedAt: Date;
  reviewNotes: string;
}
```

EOD Generation Flow





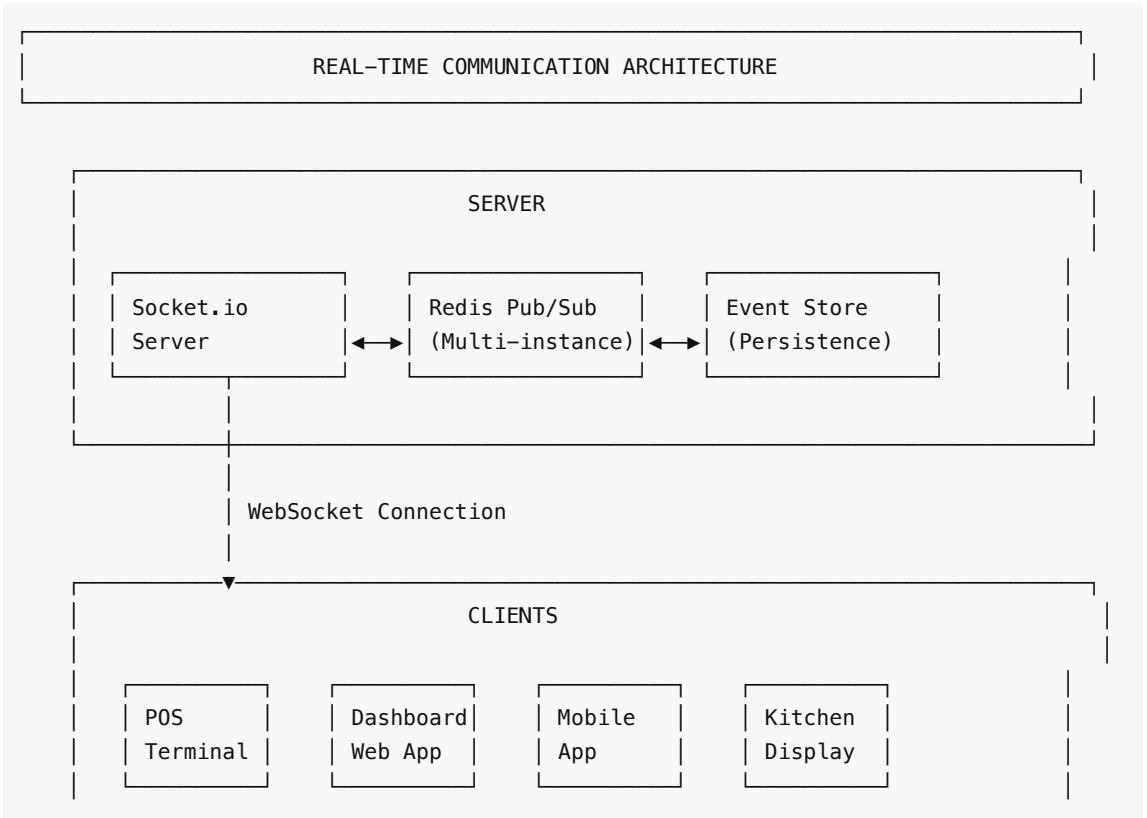
---

14. Real-time Communication

Reliable Real-time Service (`realtime-reliable.service.ts`)

Socket.io-based real-time communication with guaranteed delivery, acknowledgment, retry logic, and event persistence.

Architecture



Event Types

```
enum RealtimeEvent {
  // Order events
  ORDER_CREATED = 'order:created',
  ORDER_UPDATED = 'order:updated',
  ORDER_COMPLETED = 'order:completed',
  ORDER_CANCELLED = 'order:cancelled',

  // Payment events
  PAYMENT_RECEIVED = 'payment:received',
  PAYMENT_FAILED = 'payment:failed',
  REFUND_PROCESSED = 'refund:processed',

  // Inventory events
  STOCK_UPDATED = 'stock:updated',
  LOW_STOCK_ALERT = 'stock:low',
  OUT_OF_STOCK_ALERT = 'stock:out',

  // Shift events
  SHIFT_OPENED = 'shift:opened',
  SHIFT_CLOSED = 'shift:closed',
  CASH_MOVEMENT = 'cash:movement',

  // Product events
  PRODUCT_UPDATED = 'product:updated',
  PRICE_CHANGED = 'product:price_changed',

  // Customer events
  LOYALTY_EARNED = 'loyalty:earned',
  LOYALTY_REDEEMED = 'loyalty:redeemed',

  // System events
  SYNC_REQUIRED = 'system:sync_required',
  SYSTEM_NOTIFICATION = 'system:notification',
}
```

Guaranteed Delivery Flow

```
|  NORMAL (3)    | Best effort       | Status updates, notifications          |
|  LOW (4)       | Can be dropped    | Heartbeats, analytics pings            |
└──────────────────────────────────────────────────────────────────────────────┘


EVENT FLOW WITH ACK:
────────────────────

    Server                                      Client
       │                                          │
       │  1. Emit event with eventId              │
       │  ──────────────────────────────────────► │
       │                                          │
       │                                          │ 2. Process event
       │                                          │
       │  3. ACK with eventId                     │
       │  ◄────────────────────────────────────── │
       │                                          │
       │  4. Remove from pending                  │
       │                                          │


RETRY LOGIC:
────────────

┌──────────┐     ┌──────────┐     ┌──────────┐     ┌──────────┐
│  Send    │────►│ Wait for │────►│  Retry   │────►│  Mark    │
│  Event   │     │ ACK (5s) │     │ (3 max)  │     │  Failed  │
└──────────┘     └──────────┘     └──────────┘     └──────────┘
                      │                │
                      │                │
                      │ ACK received   │ ACK received
                      ▼                ▼
                ┌──────────────────────────┐
                │      Event Delivered     │
                │    (Remove from queue)   │
                └──────────────────────────┘


RECONNECTION HANDLING:
──────────────────────

1. Client disconnects
2. Server stores pending events in Redis
3. Client reconnects with lastEventId
4. Server replays all events since lastEventId
5. Normal operation resumes
```

Event Persistence

```
┌──────────────────────────────────────────────────────────────────────────────┐
│                          REDIS EVENT STORAGE                                 │
└──────────────────────────────────────────────────────────────────────────────┘
```

```
Key Structure:
──────────────

Pending Events:    pos:realtime:pending:{clientId}:{eventId}
Event Stream:      pos:realtime:stream:{businessId}
Client State:      pos:realtime:client:{clientId}


Event Storage Format:
─────────────────────

{
  "eventId": "evt-abc123",
  "type": "order:created",
  "priority": 1,
  "data": { ... },
  "timestamp": 1704585600000,
  "retryCount": 0,
  "acked": false
}


TTL Configuration:
──────────────────

• Critical events: 24 hours
• High priority: 4 hours
• Normal: 1 hour
• Low: No persistence
```

Cache Service Extended Operations

The `cacheService` provides comprehensive Redis operations for real-time features, including sorted sets for event queues and priority handling.

```
// Sorted Set Operations (for event queues and priority handling)

// Add item with score (timestamp/priority)
await cacheService.zadd('pos:realtime:events:biz123', Date.now(), eventJson);

// Get items by score range (events since timestamp)
const events = await cacheService.zrangebyscore(
  'pos:realtime:events:biz123',
  lastTimestamp,
  '+inf',
  100  // limit
);

// Get items in reverse order (newest first)
const recentEvents = await cacheService.zrevrangebyscore(
  'pos:realtime:events:biz123',
```

```
  '+inf',
  0,
  10  // limit
);

// Clean up old events by score range
await cacheService.zremrangebyscore(
  'pos:realtime:events:biz123',
  0,
  oldTimestamp
);

// Remove specific event
await cacheService.zrem('pos:realtime:events:biz123', eventJson);

// Get count of pending events
const count = await cacheService.zcard('pos:realtime:events:biz123');

// Set TTL on key
await cacheService.expire('pos:realtime:events:biz123', 86400);

// Batch get multiple keys
const values = await cacheService.mget<EventData>([
  'pos:event:evt1',
  'pos:event:evt2',
  'pos:event:evt3',
]);

// Access raw Redis client for advanced operations
const client = cacheService.getClient();
```

```
┌──────────────────────────────────────────────────────────────────────────────┐
│                      CACHE SERVICE SORTED SET OPERATIONS                     │
├──────────────────────────────────────────────────────────────────────────────┤
│                                                                              │
│  USE CASES:                                                                  │
│  ──────────                                                                  │
│  • Event Queues: Store events with timestamp scores for replay               │
│  • Priority Queues: Process items by priority score                          │
│  • Rate Limiting: Track request timestamps in sliding window                 │
│  • Leaderboards: Track top performers with scores                            │
│                                                                              │
│  EXAMPLE: Event Queue with Priority                                          │
│  ──────────────────────────────────                                          │
│                                                                              │
│  Key: pos:realtime:pending:client123                                         │
│  ┌──────────────┬──────────────┬──────────────────────────────┐              │
│  │    Score     │   Priority   │         Member (Event)       │              │
│  ├──────────────┼──────────────┼──────────────────────────────┤              │
│  │ 1704585600001│   CRITICAL   │ {"type":"order:created",...} │              │
│  │ 1704585600050│   HIGH       │ {"type":"stock:updated",...} │              │
```

```
│  │ 1704585600100│   NORMAL    │ {"type":"shift:opened",...}    │              │
│  └──────────────┴─────────────┴────────────────────────────────┘              │
│                                                                              │
└──────────────────────────────────────────────────────────────────────────────┘
```

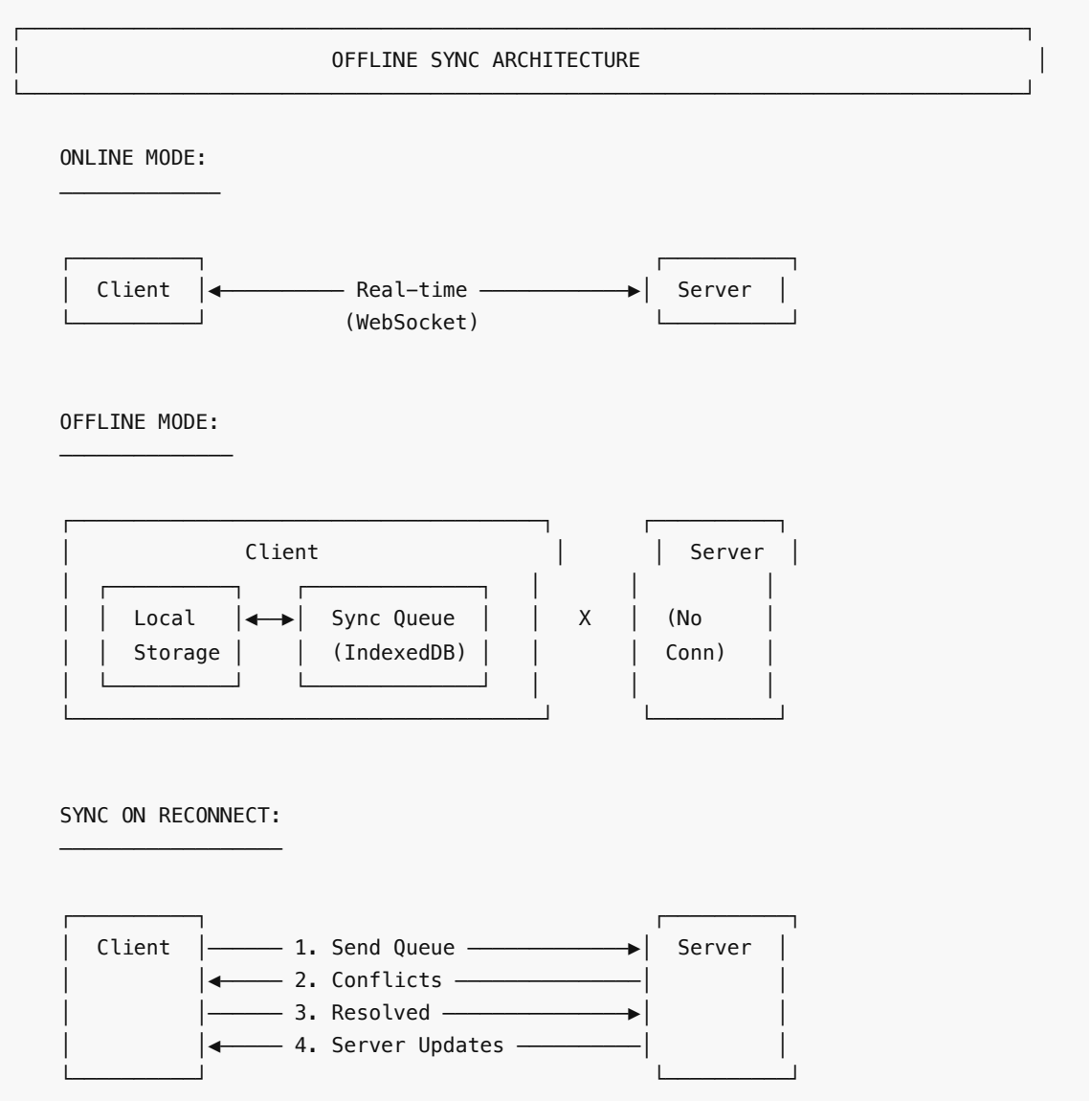
---

15. Offline Mode & Sync

Offline Sync Service (`offline-sync.service.ts`)

Enables POS terminals to operate offline and synchronize changes when connectivity is restored.

Sync Architecture



Sync Item Structure

```typescript
interface SyncItem {
  id: string;
```

```
  clientId: string;           // Device identifier
  businessId: string;
  entityType: string;         // 'order', 'customer', etc.
  entityId: string;
  operation: SyncOperation;   // CREATE, UPDATE, DELETE
  data: Record<string, any>;
  clientTimestamp: Date;      // When changed offline
  serverTimestamp?: Date;     // When synced
  status: SyncStatus;         // PENDING, PROCESSING, COMPLETED, CONFLICT
  retryCount: number;
  version: number;            // For conflict detection
  priority: number;           // Processing order
}
```

Conflict Resolution Strategies

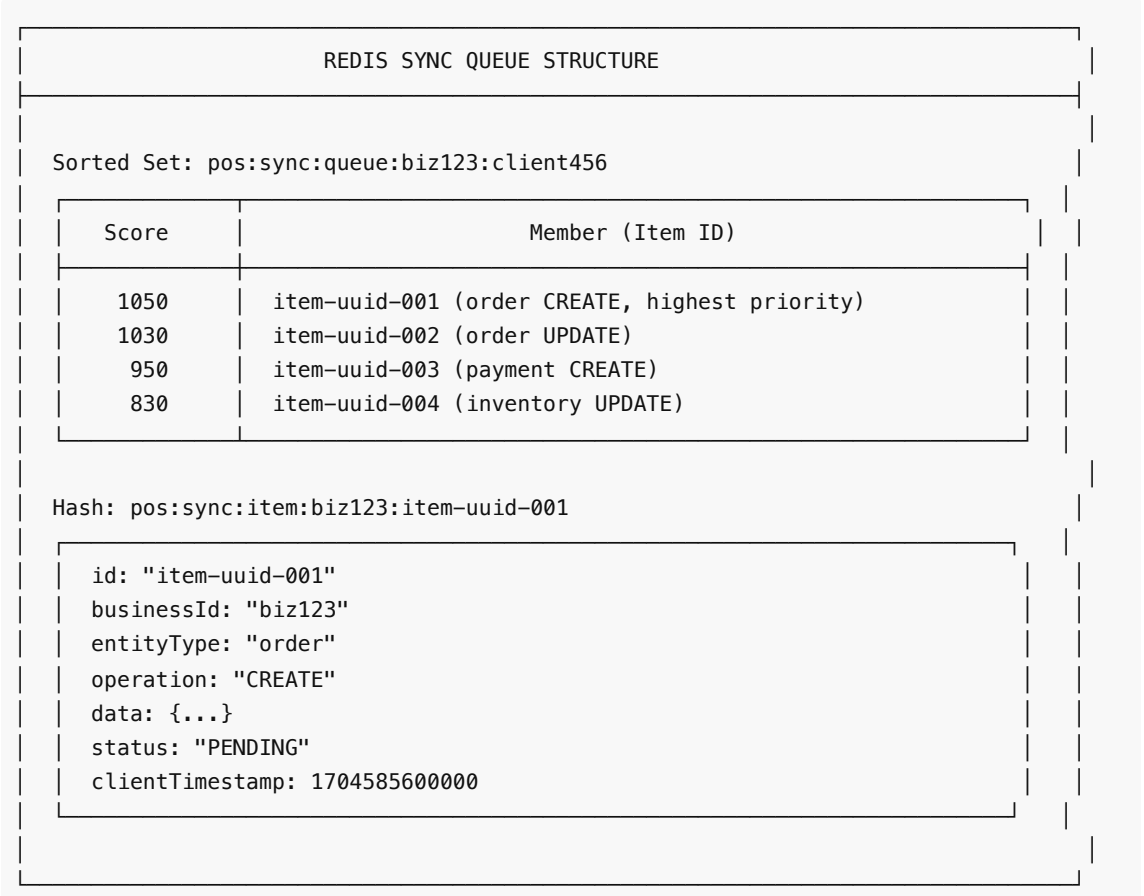
```
┌─────────────────────────────────────────────────────────────────────────┐
│                      CONFLICT RESOLUTION STRATEGIES                     │
└─────────────────────────────────────────────────────────────────────────┘

   1. CLIENT_WINS
   ──────────────
   Offline changes override server. Use when client is authoritative.

   Server: { name: "Product A", price: 10 }
   Client: { name: "Product A", price: 12 }  (changed offline)
   Result: { name: "Product A", price: 12 }  ← Client wins


   2. SERVER_WINS
   ──────────────
   Server data overrides offline changes. Use for critical data.

   Server: { name: "Product A", price: 10 }  (updated)
   Client: { name: "Product A", price: 12 }  (changed offline)
   Result: { name: "Product A", price: 10 }  ← Server wins


   3. MERGE
   ────────
   Attempt to merge non-conflicting fields.

   Server: { name: "Product A", price: 10, stock: 50 }  (stock updated)
   Client: { name: "Product A", price: 12, stock: 100 } (price changed offline)
   Result: { name: "Product A", price: 12, stock: 50 }  ← Merged


   4. MANUAL
   ─────────
   Require user intervention for resolution.
```

```
    Server: { name: "Product A", price: 10 }
    Client: { name: "Product A", price: 12 }
    Result: CONFLICT - User must choose
```

Redis-Based Persistence

The offline sync service uses Redis for durable storage of sync queues, ensuring data survives server restarts and enabling distributed processing.

```
// Redis Key Structure
const REDIS_KEYS = {
  SYNC_QUEUE: 'pos:sync:queue:{businessId}:{clientId}',      // Pending items
  SYNC_ITEM: 'pos:sync:item:{businessId}:{itemId}',          // Individual items
  SYNC_PROCESSING: 'pos:sync:processing:{businessId}:{clientId}', // Lock flag
};

// TTL Configuration
const TTL = {
  SYNC_ITEM: 7 * 24 * 60 * 60,        // 7 days for sync items
  PROCESSING_LOCK: 5 * 60,             // 5 minutes for processing lock
};
```



Multi-Tenant Security

The offline sync service enforces strict tenant isolation to prevent cross-business data access:

```
// Tenant validation during queue processing
async processQueue(businessId: string, clientId: string): Promise<SyncResult[]> {
  // 1. Validate all items belong to the requesting business
  pendingItems = pendingItems.filter(item => {
    if (item.businessId !== businessId) {
      console.warn(`Sync item ${item.id} rejected: businessId mismatch`);
      return false;  // Reject cross-tenant items
    }
    return true;
  });

  // 2. Validate entity ownership before UPDATE/DELETE
  if (operation === SyncOperation.UPDATE || operation === SyncOperation.DELETE) {
    const existing = await repository.findOne({
      where: {
        id: item.entityId,
        businessId: item.businessId  // Must match business
      }
    });

    if (!existing) {
      throw new Error(`Entity ${item.entityId} not found or access denied`);
    }
  }
}
```

```
┌─────────────────────────────────────────────────────────────────────────────┐
│                      TENANT ISOLATION ENFORCEMENT                           │
├─────────────────────────────────────────────────────────────────────────────┤
│                                                                             │
│  Security Layers:                                                           │
│  ─────────────────                                                          │
│                                                                             │
│  1. QUEUE LEVEL: Items stored in business-specific Redis keys               │
│     pos:sync:queue:{businessId}:*                                           │
│                                                                             │
│  2. ITEM VALIDATION: businessId verified before processing                  │
│     item.businessId === request.businessId                                  │
│                                                                             │
│  3. DATABASE LEVEL: Queries always include businessId filter                │
│     WHERE id = :entityId AND businessId = :businessId                       │
│                                                                             │
│  4. AUDIT LOGGING: All sync operations logged with tenant context           │
│                                                                             │
│  Attack Prevention:                                                         │
│  ──────────────────                                                         │
│  ✗ Cannot inject items for other businesses into queue                      │
│  ✗ Cannot modify entities owned by other businesses                         │
│  ✗ Cannot read sync status of other businesses                              │
```

```
│                                                                              │
└──────────────────────────────────────────────────────────────────────────────┘
```

Delta Sync

```
┌──────────────────────────────────────────────────────────────────────────────┐
│                            DELTA SYNC PROCESS                                │
└──────────────────────────────────────────────────────────────────────────────┘

    FULL SYNC (Initial or recovery):
    ────────────────────────────────

    Client ──── "Give me everything" ────► Server
    Client ◄──── All data (products,        Server
                 customers, etc.)


    DELTA SYNC (Incremental):
    ─────────────────────────

    Client ──── "Changes since 2026-01-07T10:00:00" ────► Server
    Client ◄──── {                                         Server
                   created: [new items],
                   updated: [modified items],
                   deleted: [removed ids]
                 }


    CHECKSUM VERIFICATION:
    ──────────────────────

    Before: Client checksum = "a1b2c3"
            Server checksum = "a1b2c3"
            ✓ In sync

    After changes:
            Client checksum = "d4e5f6"
            Server checksum = "d4e5f6"
            ✓ Sync successful
```

Priority Processing

```
┌──────────────────────────────────────────────────────────────────────────────┐
│                          SYNC PRIORITY ORDER                                 │
└──────────────────────────────────────────────────────────────────────────────┘

    PRIORITY CALCULATION:
    ─────────────────────

    Base priority + Entity priority + Operation priority
```

```
Entity Priorities:
• order:     1000  (highest - financial data)
• payment:    900
• inventory:  800
• customer:   500
• product:    400
• category:   300

Operation Priorities:
• CREATE:     +50  (new data important)
• UPDATE:     +30
• DELETE:     +10  (least urgent)


EXAMPLE QUEUE:
──────────────

┌──────┬─────────────┬─────────────┬─────────────┐
│ Rank │   Entity    │ Operation │ Priority │
├──────┼─────────────┼─────────────┼─────────────┤
│  1   │   Order     │  CREATE   │   1050   │
│  2   │   Order     │  UPDATE   │   1030   │
│  3   │  Payment    │  CREATE   │    950   │
│  4   │ Inventory   │  UPDATE   │    830   │
│  5   │  Customer   │  CREATE   │    550   │
│  6   │  Product    │  UPDATE   │    430   │
└──────┴─────────────┴─────────────┴─────────────┘
```

---

16. Reporting & Analytics

Report Service (`report.service.ts`)

Comprehensive reporting system with sales, inventory, and customer analytics, supporting multiple export formats.

Report Types

```
┌──────────────────────────────────────────────────────────────────────────────┐
│                          AVAILABLE REPORTS                                   │
└──────────────────────────────────────────────────────────────────────────────┘

    1. SALES REPORT
    ──────────────────

    ┌──────────────────────────────────────────────────────────┐
    │  Metrics:                                                │
    │  • Total Sales / Net Sales                               │
    │  • Order Count / Average Order Value                     │
    │  • Tax / Discounts / Refunds                             │
    │  • Gross Profit / Profit Margin                          │
    │                                                          │
    │  Breakdowns:                                             │
```

```
│  • By Date (daily/weekly/monthly)                          │
│  • By Hour (peak hours analysis)                           │
│  • By Category                                             │
│  • By Payment Method                                       │
│  • Top Products                                            │
│  • Top Customers                                           │
└────────────────────────────────────────────────────────────┘


2. INVENTORY REPORT
────────────────────

┌────────────────────────────────────────────────────────────┐
│  Metrics:                                                  │
│  • Total Products / Total Value                            │
│  • Low Stock Count / Out of Stock Count                    │
│  • Expiring Soon Count                                     │
│                                                            │
│  Breakdowns:                                               │
│  • By Category                                             │
│  • Low Stock Items List                                    │
│  • Expiring Items List                                     │
│  • Inventory Movement                                      │
└────────────────────────────────────────────────────────────┘


3. CUSTOMER REPORT
────────────────────

┌────────────────────────────────────────────────────────────┐
│  Metrics:                                                  │
│  • Total / New / Returning Customers                       │
│  • Average Customer Value                                  │
│  • Repeat Purchase Rate                                    │
│                                                            │
│  Breakdowns:                                               │
│  • By Segment                                              │
│  • Top Customers                                           │
│  • Acquisition Trend                                       │
└────────────────────────────────────────────────────────────┘
```

Export Formats

```
┌──────────────────────────────────────────────────────────────────────────┐
│                          EXPORT FORMATS                                  │
└──────────────────────────────────────────────────────────────────────────┘


JSON (API Response)
────────────────────

{
  "summary": { "totalSales": 15000, ... },
  "salesByDate": [...],
  "topProducts": [...]
}
```

```
CSV (Spreadsheet)
─────────────────

Date,Sales,Orders,Average
2026-01-01,1500.00,25,60.00
2026-01-02,1800.00,30,60.00
...


EXCEL (.xlsx)
─────────────

┌─────────────────────────────────────────────┐
│  Sheet 1: Summary                           │
│  Sheet 2: Daily Sales                       │
│  Sheet 3: Top Products                      │
│  Sheet 4: Payment Methods                   │
└─────────────────────────────────────────────┘


PDF (Printable)
───────────────

┌─────────────────────────────────────────────┐
│  ╔═══════════════════════════════════════╗  │
│  ║         SALES REPORT                  ║  │
│  ║         Jan 1 - Jan 31, 2026          ║  │
│  ╚═══════════════════════════════════════╝  │
│                                             │
│  SUMMARY                                    │
│  ────────                                   │
│  Total Sales: $15,000.00                    │
│  Total Orders: 250                          │
│  ...                                        │
└─────────────────────────────────────────────┘
```

Native PDF Generation

The report service includes a native PDF generator implementing the PDF 1.4 specification, enabling high-quality printable reports without external dependencies.

```
// PDF Export - Native generation with proper structure
const pdf = await reportService.exportToPDF(
  'Sales Report',
  reportData,
  'Acme Store Inc.',
  { start: new Date('2026-01-01'), end: new Date('2026-01-31') }
);

// Returns: Buffer containing valid PDF 1.4 document
```

PDF Features:

- PDF 1.4 specification compliance
- Embedded Helvetica font (PDF standard font)
- Multi-page support with automatic pagination
- Header with business name and date range
- Formatted data tables with proper alignment
- Summary section with key metrics
- Professional layout with proper margins

```
┌──────────────────────────────────────────────────────────────────────────┐
│                        PDF DOCUMENT STRUCTURE                            │
├──────────────────────────────────────────────────────────────────────────┤
│                                                                          │
│  %PDF-1.4                          ← Version header                      │
│  %âãÏÓ                              ← Binary indicator                    │
│                                                                          │
│  1 0 obj << /Type /Catalog /Pages 2 0 R >> endobj                        │
│  2 0 obj << /Type /Pages /Kids [...] /Count n >> endobj                  │
│  3 0 obj << /Type /Page ... /Contents 5 0 R >> endobj                    │
│  4 0 obj << /Type /Font /Subtype /Type1 /BaseFont /Helvetica >> endobj   │
│  5 0 obj << /Length ... >> stream ... endstream endobj                   │
│                                                                          │
│  xref                              ← Cross-reference table               │
│  0 6                                                                     │
│  0000000000 65535 f                                                      │
│  0000000015 00000 n                                                      │
│  ...                                                                     │
│  trailer << /Size 6 /Root 1 0 R >>                                       │
│  startxref                                                               │
│  [byte offset]                                                           │
│  %%EOF                                                                   │
│                                                                          │
└──────────────────────────────────────────────────────────────────────────┘
```

Native Excel Generation

The report service generates valid XLSX files using the Office Open XML (OOXML) specification with proper ZIP structure.

```
// Excel Export - Native XLSX generation
const excel = await reportService.exportToExcel(reportData, 'Sales Report');

// Returns: Buffer containing valid XLSX file (ZIP with XML contents)
```

Excel Features:

- OOXML specification compliance
- Multiple worksheets (Summary, Details, Charts data)
- Proper column headers with formatting
- Numeric formatting for currency/percentages
- Compatible with Excel, Google Sheets, LibreOffice

```
┌──────────────────────────────────────────────────────────────────────────┐
│                        XLSX FILE STRUCTURE                               │
├──────────────────────────────────────────────────────────────────────────┤
│                                                                          │
│  report.xlsx (ZIP Archive)                                               │
│  │                                                                       │
│  ├── [Content_Types].xml          ← Content type definitions             │
│  │   <?xml ...?>                                                         │
│  │   <Types xmlns="...">                                                 │
│  │     <Default Extension="xml" ContentType="application/xml"/>          │
│  │     <Default Extension="rels" ContentType="...relationships+xml"/>    │
│  │     <Override PartName="/xl/workbook.xml" ContentType="...spreadsheet"/>
│  │   </Types>                                                            │
│  │                                                                       │
│  ├── _rels/                                                              │
│  │   └── .rels                    ← Root relationships                   │
│  │                                                                       │
│  └── xl/                                                                 │
│      ├── workbook.xml             ← Workbook definition                  │
│      ├── styles.xml               ← Cell styles & formatting             │
│      ├── sharedStrings.xml        ← Shared string table                  │
│      ├── worksheets/                                                     │
│      │   └── sheet1.xml           ← Worksheet data                       │
│      └── _rels/                                                          │
│          └── workbook.xml.rels    ← Workbook relationships               │
│                                                                          │
│  CRC-32 Checksums: Calculated per file for ZIP integrity                 │
│                                                                          │
└──────────────────────────────────────────────────────────────────────────┘
```

Period Comparison

```
// Compare two periods
const comparison = await reportService.generateComparisonReport(
  businessId,
  { start: new Date('2025-12-01'), end: new Date('2025-12-31') }, // Last month
  { start: new Date('2026-01-01'), end: new Date('2026-01-31') }, // This month
);

// Returns:
{
  period1: { /* December sales data */ },
  period2: { /* January sales data */ },
  changes: {
    salesChange: 15.5,      // +15.5% increase
    ordersChange: 12.3,     // +12.3% increase
    avgOrderChange: 2.8,    // +2.8% increase
    profitChange: 18.2,     // +18.2% increase
  }
}
```

---

17. Audit Trail

Audit Service (`audit.service.ts`)

Comprehensive audit logging for compliance, security, and operational insights.

Event Types

```
enum AuditEventType {
  // Authentication
  AUTH_LOGIN = 'auth.login',
  AUTH_LOGOUT = 'auth.logout',
  AUTH_FAILED = 'auth.failed',
  AUTH_PASSWORD_CHANGE = 'auth.password_change',
  AUTH_PASSWORD_RESET = 'auth.password_reset',

  // Orders
  ORDER_CREATED = 'order.created',
  ORDER_UPDATED = 'order.updated',
  ORDER_COMPLETED = 'order.completed',
  ORDER_CANCELLED = 'order.cancelled',
  ORDER_VOIDED = 'order.voided',

  // Payments
  PAYMENT_PROCESSED = 'payment.processed',
  PAYMENT_FAILED = 'payment.failed',
  PAYMENT_VOIDED = 'payment.voided',
  REFUND_PROCESSED = 'refund.processed',

  // Inventory
  INVENTORY_ADJUSTED = 'inventory.adjusted',
  INVENTORY_TRANSFERRED = 'inventory.transferred',
  INVENTORY_COUNT = 'inventory.count',

  // Products
  PRODUCT_CREATED = 'product.created',
  PRODUCT_UPDATED = 'product.updated',
  PRODUCT_DELETED = 'product.deleted',
  PRICE_CHANGED = 'product.price_changed',

  // Cash Drawer
  DRAWER_OPENED = 'drawer.opened',
  DRAWER_CLOSED = 'drawer.closed',
  CASH_ADDED = 'cash.added',
  CASH_REMOVED = 'cash.removed',
  CASH_DROP = 'cash.drop',

  // Admin Actions
  USER_CREATED = 'user.created',
  USER_UPDATED = 'user.updated',
  USER_DELETED = 'user.deleted',
```

```
  PERMISSIONS_CHANGED = 'permissions.changed',
  SETTINGS_CHANGED = 'settings.changed',
}
```

Audit Log Structure

```
interface AuditLog {
  id: string;
  eventType: AuditEventType;
  businessId: string;
  userId: string;
  userName: string;

  // What was affected
  entityType: string;       // 'order', 'product', etc.
  entityId: string;

  // Change details
  action: string;           // Human-readable description
  oldValue: any;            // Previous state
  newValue: any;            // New state

  // Context
  metadata: {
    ipAddress: string;
    userAgent: string;
    locationId: string;
    shiftId: string;
    orderId: string;
  };

  // Timestamps
  timestamp: Date;

  // Compliance
  isHighRisk: boolean;      // Flag sensitive operations
  requiresReview: boolean;
}
```

Compliance Report

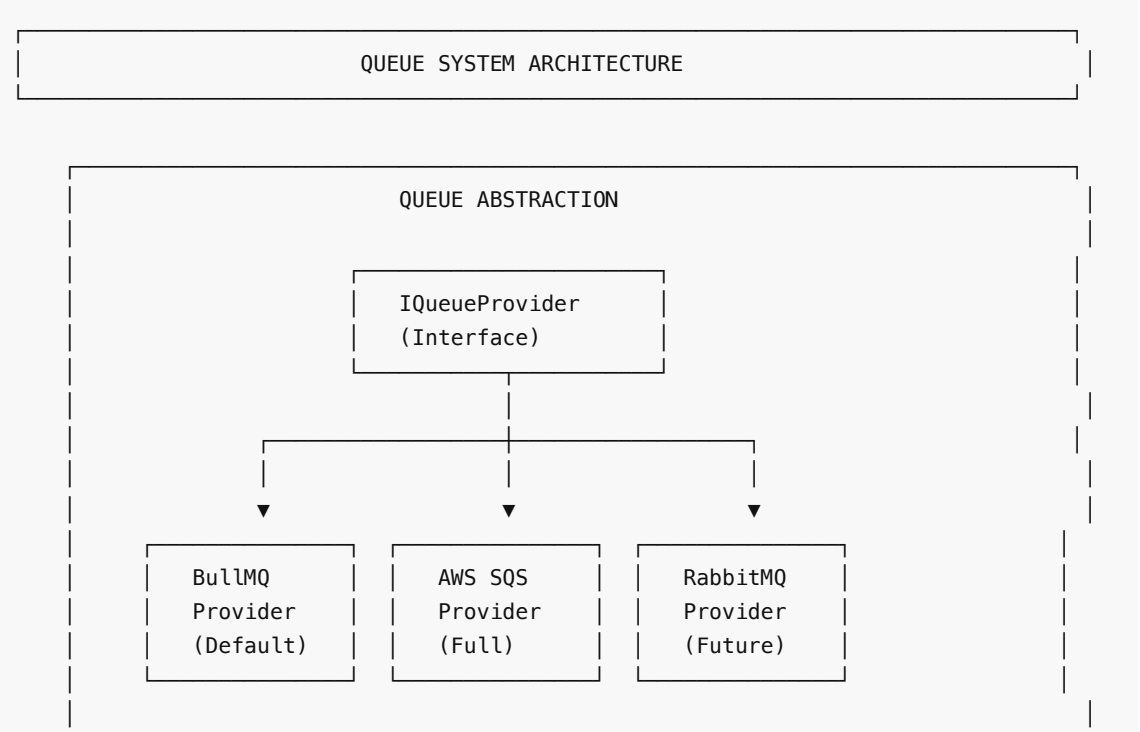
```
┌─────────────────────────────────────────────────────────────────────────┐
│                        COMPLIANCE AUDIT REPORT                          │
├─────────────────────────────────────────────────────────────────────────┤
│  Period: 2026-01-01 to 2026-01-31                                       │
│  Generated: 2026-02-01 09:00:00                                         │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                         │
│  SUMMARY                                                                │
│  ────────                                                               │
```

```
│  Total Events: 15,423                                                    │
│  High Risk Events: 47                                                    │
│  Failed Authentications: 12                                              │
│  Price Changes: 89                                                       │
│  Void Transactions: 23                                                   │
│  Cash Discrepancies: 8                                                   │
│                                                                          │
│  HIGH RISK EVENTS                                                        │
│  ────────────────                                                        │
│  ┌────────────┬──────────────────────┬─────────────┬──────────────────────┐ │
│  │    Date    │       Event          │    User     │        Details       │ │
│  ├────────────┼──────────────────────┼─────────────┼──────────────────────┤ │
│  │ 2026-01-05 │ ORDER_VOIDED         │ John Smith  │ Order #1234 - $250.00│ │
│  │ 2026-01-08 │ PRICE_CHANGED        │ Jane Doe    │ Product A: $10→$5 (50%↓)│ │
│  │ 2026-01-12 │ CASH_DISCREPANCY     │ Bob Wilson  │ Shift close: -$15.50 │ │
│  │ ...        │ ...                  │ ...         │ ...                  │ │
│  └────────────┴──────────────────────┴─────────────┴──────────────────────┘ │
│                                                                          │
│  AUTHENTICATION FAILURES                                                 │
│  ──────────────────────                                                  │
│  User: admin@store.com - 5 failed attempts (2026-01-15)                  │
│  User: manager@store.com - 3 failed attempts (2026-01-20)                │
│  IP: 192.168.1.100 - 4 failed attempts (multiple users)                  │
│                                                                          │
└──────────────────────────────────────────────────────────────────────────┘
```

---

18. Queue System

Queue Architecture



```
└──────────────────────────────────────────────────────────────────────────────┘


QUEUE TYPES:
────────────

┌──────────────────┬──────────────────────────────────────────────────────────┐
│      Queue       │                        Jobs                              │
├──────────────────┼──────────────────────────────────────────────────────────┤
│  orders          │  process-order, calculate-totals, send-confirmation      │
│  inventory       │  update-stock, low-stock-alert, reorder-check            │
│  reports         │  daily-report, weekly-report, custom-report              │
│  notifications   │  send-email, send-sms, push-notification                 │
│  sync            │  process-sync-batch, resolve-conflicts                   │
└──────────────────┴──────────────────────────────────────────────────────────┘
```

Job Processing Flow

```
┌──────────────────────────────────────────────────────────────────────────────┐
│                        JOB PROCESSING FLOW                                   │
└──────────────────────────────────────────────────────────────────────────────┘

    1. JOB DISPATCH
    ───────────────

    ┌─────────┐     ┌─────────┐     ┌─────────┐
    │ Service │────▶│  Queue  │────▶│  Redis  │
    │ Layer   │     │ Dispatch│     │ (BullMQ)│
    └─────────┘     └─────────┘     └─────────┘


    2. JOB PROCESSING
    ─────────────────

    ┌─────────┐     ┌─────────┐     ┌─────────┐     ┌─────────┐
    │  Redis  │────▶│  Worker │────▶│ Process │────▶│ Complete│
    │ (Queue) │     │ (Poll)  │     │  Job    │     │ or Fail │
    └─────────┘     └─────────┘     └─────────┘     └─────────┘


    3. RETRY LOGIC
    ──────────────

         Job Fails
             │
             ▼
    ┌─────────────────┐      YES      ┌─────────────────┐
    │ Retry < Max?    │──────────────▶│  Re-queue       │
    │ (default: 3)    │               │  with backoff   │
    └─────────────────┘               └─────────────────┘
             │
             │ NO
             ▼
```

```
    ┌──────────────────┐
    │  Move to DLQ     │
    │ (Dead Letter Q)  │
    └──────────────────┘
```

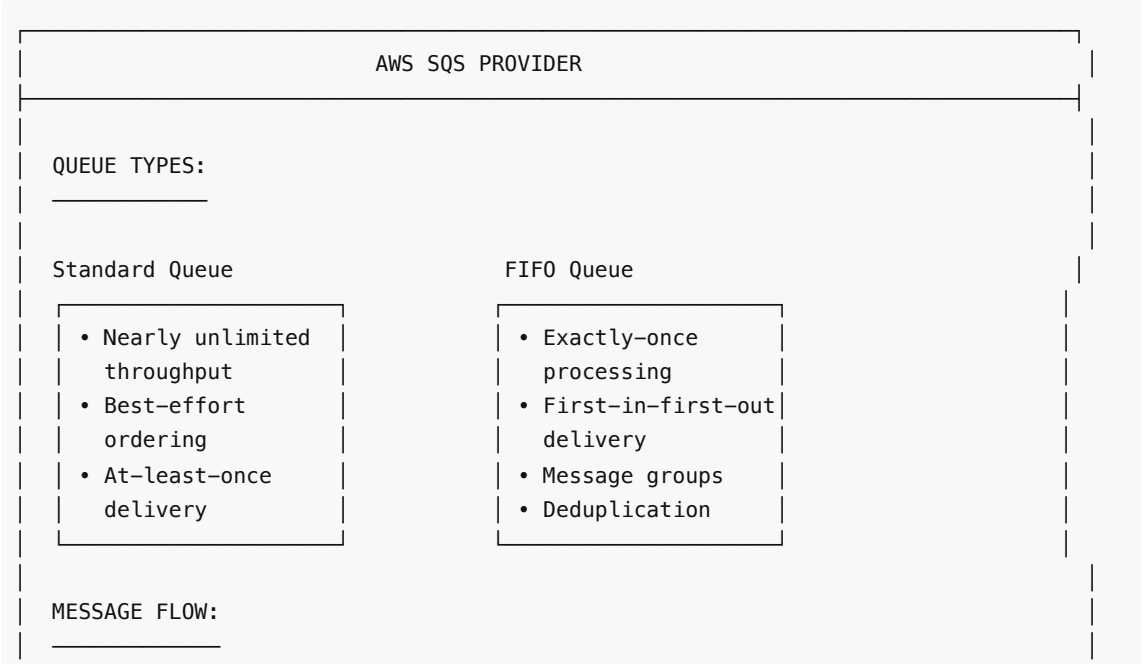
AWS SQS Provider

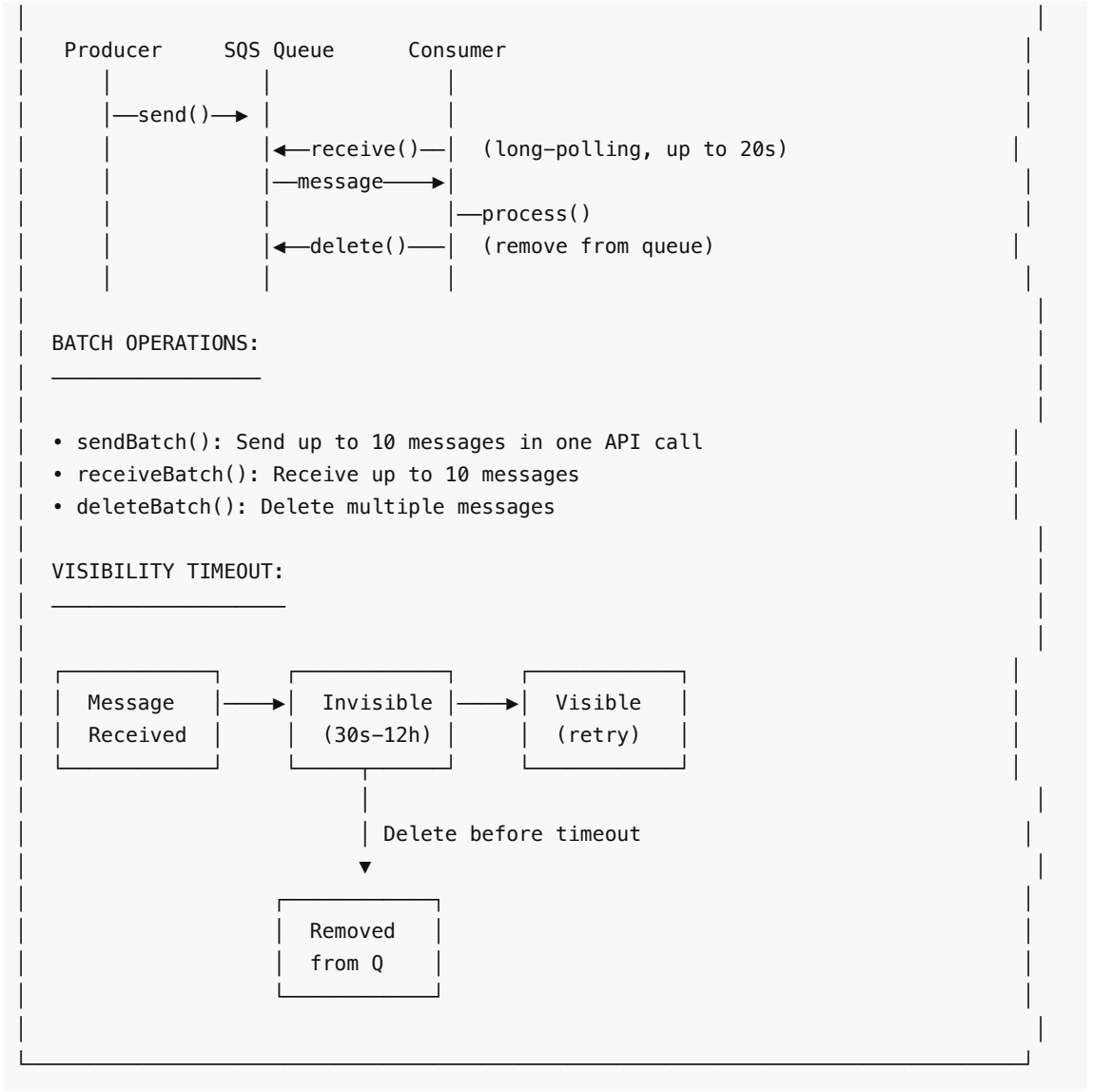
Full AWS Simple Queue Service integration for production deployments with FIFO queue support, long-polling, and batch operations.

```
// Configuration
const sqsConfig = {
  region: 'us-east-1',
  accessKeyId: process.env.AWS_ACCESS_KEY_ID,
  secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
  queueUrlPrefix: 'https://sqs.us-east-1.amazonaws.com/123456789',
};

// Initialize provider
const sqsProvider = new SQSProvider(sqsConfig);
await sqsProvider.initialize();
```

SQS Features:

- Standard and FIFO queue support
- Long-polling for efficient message retrieval
- Delayed message delivery (up to 15 minutes)
- Batch send/receive/delete operations
- Automatic visibility timeout management
- Dead letter queue integration
- Message deduplication (FIFO)
- Message grouping (FIFO)






```typescript
// SQS Provider API

interface IQueueProvider {
  // Lifecycle
  initialize(): Promise<void>;
  shutdown(): Promise<void>;

  // Queue management
  createQueue(name: string, options?: QueueOptions): Promise<string>;
  deleteQueue(name: string): Promise<void>;
  getQueueUrl(name: string): Promise<string>;

  // Message operations
  send(queueName: string, message: QueueMessage): Promise<string>;
  sendBatch(queueName: string, messages: QueueMessage[]): Promise<string[]>;
  receive(queueName: string, options?: ReceiveOptions): Promise<ReceivedMessage[]>;
  delete(queueName: string, receiptHandle: string): Promise<void>;
```

```
  deleteBatch(queueName: string, receiptHandles: string[]): Promise<void>;

  // Queue info
  getQueueAttributes(name: string): Promise<QueueAttributes>;
  purgeQueue(name: string): Promise<void>;
}

// Example usage
await sqsProvider.send('orders', {
  body: JSON.stringify({ orderId: 'ord-123', action: 'process' }),
  delaySeconds: 0,
  messageGroupId: 'order-processing',     // FIFO only
  messageDeduplicationId: 'unique-id-123', // FIFO only
});

const messages = await sqsProvider.receive('orders', {
  maxMessages: 10,
  waitTimeSeconds: 20,  // Long polling
  visibilityTimeout: 300,
});

for (const msg of messages) {
  await processOrder(JSON.parse(msg.body));
  await sqsProvider.delete('orders', msg.receiptHandle);
}
```

---

19. API Reference

Endpoint Summary

```
┌─────────────────────────────────────────────────────────────────────────┐
│                           API ENDPOINTS                                 │
└─────────────────────────────────────────────────────────────────────────┘

   BASE URL: /api/v1

   AUTHENTICATION
   ──────────────
   POST   /auth/login              Login user
   POST   /auth/register           Register business
   POST   /auth/refresh            Refresh token
   POST   /auth/logout             Logout
   POST   /auth/forgot-password    Request password reset
   POST   /auth/reset-password     Reset password

   ORDERS
   ──────
   GET    /orders                  List orders
   POST   /orders                  Create order
   GET    /orders/:id              Get order
```

```
PUT    /orders/:id              Update order
DELETE /orders/:id              Delete order

PAYMENTS
────────
POST   /orders/:id/pay          Process payment
POST   /orders/:id/pay/split    Split payment
POST   /orders/:id/refund       Process refund
POST   /orders/:id/complete     Complete order
POST   /orders/:id/cancel       Cancel order
POST   /calculate-totals        Calculate order totals

PRODUCTS
────────
GET    /products                List products
POST   /products                Create product
GET    /products/:id            Get product
PUT    /products/:id            Update product
DELETE /products/:id            Delete product

CATEGORIES
──────────
GET    /categories              List categories
POST   /categories              Create category
GET    /categories/:id          Get category
PUT    /categories/:id          Update category
DELETE /categories/:id          Delete category

CUSTOMERS
─────────
GET    /customers               List customers
POST   /customers               Create customer
GET    /customers/:id           Get customer
PUT    /customers/:id           Update customer
DELETE /customers/:id           Delete customer

INVENTORY
─────────
GET    /inventory               List inventory
POST   /inventory/adjust        Adjust stock
POST   /inventory/count         Stock count
GET    /inventory/low-stock     Low stock alerts

LOCATIONS
─────────
GET    /locations               List locations
POST   /locations               Create location
GET    /locations/:id           Get location
PUT    /locations/:id           Update location
DELETE /locations/:id           Delete location

TRANSFERS
```

```
─────────

GET    /transfers               List transfers
POST   /transfers               Create transfer
GET    /transfers/:id           Get transfer
POST   /transfers/:id/submit    Submit for approval
POST   /transfers/:id/approve   Approve transfer
POST   /transfers/:id/ship      Ship transfer
POST   /transfers/:id/receive   Receive transfer
POST   /transfers/:id/complete  Complete transfer
POST   /transfers/:id/cancel    Cancel transfer

SUPPLIERS
─────────

GET    /suppliers               List suppliers
POST   /suppliers               Create supplier
GET    /suppliers/:id           Get supplier
PUT    /suppliers/:id           Update supplier
DELETE /suppliers/:id           Delete supplier

PURCHASE ORDERS
──────────────

GET    /purchase-orders         List POs
POST   /purchase-orders         Create PO
GET    /purchase-orders/:id     Get PO
POST   /purchase-orders/:id/submit    Submit PO
POST   /purchase-orders/:id/approve   Approve PO
POST   /purchase-orders/:id/send      Send to supplier
POST   /purchase-orders/:id/receive   Receive goods
POST   /purchase-orders/:id/cancel    Cancel PO

GIFT CARDS
──────────

GET    /gift-cards              List gift cards
POST   /gift-cards              Issue gift card
GET    /gift-cards/:code/balance Check balance
POST   /gift-cards/:code/redeem Redeem
POST   /gift-cards/:id/reload   Reload

LOYALTY
───────

GET    /loyalty/program         Get program config
PUT    /loyalty/program         Update program
GET    /loyalty/accounts/:id    Get customer account
POST   /loyalty/earn            Earn points
POST   /loyalty/redeem          Redeem points
GET    /loyalty/rewards         List rewards

SHIFTS
──────

GET    /shifts                  List shifts
POST   /shifts/open             Open shift
POST   /shifts/:id/close        Close shift
```

```
POST   /shifts/:id/movement     Cash movement
GET    /shifts/:id/summary      Shift summary

REPORTS
────────
GET    /reports/sales           Sales report
GET    /reports/inventory       Inventory report
GET    /reports/customers       Customer report
GET    /reports/eod             EOD report
POST   /reports/export          Export report

ANALYTICS
──────────
GET    /analytics/dashboard     Dashboard summary
GET    /analytics/sales         Sales analytics
GET    /analytics/products      Product analytics
GET    /analytics/customers     Customer analytics

BARCODES
─────────
POST   /barcodes/lookup         Lookup barcode
POST   /barcodes/validate       Validate barcode
POST   /barcodes/generate-sku   Generate SKU

SYNC (Offline)
───────────────
POST   /sync/queue              Queue sync items
POST   /sync/process            Process queue
GET    /sync/status             Sync status
GET    /sync/download           Download offline data
GET    /sync/delta              Get delta updates
```

---

20. Deployment

Docker Configuration

```yaml
# docker-compose.yml
version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: docker/Dockerfile
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
      - DATABASE_URL=postgresql://user:pass@postgres:5432/pos
      - REDIS_URL=redis://redis:6379
```

```
      - MONGODB_URL=mongodb://mongo:27017/pos_logs
    depends_on:
      - postgres
      - redis
      - mongo

  postgres:
    image: postgres:15-alpine
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB=pos
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=pass

  redis:
    image: redis:7-alpine
    volumes:
      - redis_data:/data

  mongo:
    image: mongo:6
    volumes:
      - mongo_data:/data/db

volumes:
  postgres_data:
  redis_data:
  mongo_data:
```

Environment Variables

```
# Application
NODE_ENV=production
PORT=3000
API_VERSION=v1

# PostgreSQL
DATABASE_URL=postgresql://user:password@localhost:5432/pos_db

# MongoDB (Logs)
MONGODB_URL=mongodb://localhost:27017/pos_logs

# Redis
REDIS_HOST=localhost
REDIS_PORT=6379
REDIS_PASSWORD=

# Queue
QUEUE_PROVIDER=bullmq  # or 'sqs' for AWS
```

```
# JWT
JWT_SECRET=your-secret-key
JWT_EXPIRES_IN=1d
JWT_REFRESH_EXPIRES_IN=7d

# File Storage
STORAGE_PROVIDER=local  # or 's3'
S3_BUCKET=
S3_REGION=

# Email
SMTP_HOST=
SMTP_PORT=
SMTP_USER=
SMTP_PASS=
```

Health Check Endpoint

```
// GET /health
{
  "status": "healthy",
  "timestamp": "2026-01-07T12:00:00Z",
  "version": "2.0.0",
  "services": {
    "database": "connected",
    "redis": "connected",
    "mongodb": "connected",
    "queue": "running"
  },
  "uptime": 86400
}
```

Appendix A: Migration from v1

Breaking Changes

1. **Database**: MongoDB → PostgreSQL
2. **IDs**: ObjectId (24 chars) → UUID (36 chars)
3. **Order.items**: Embedded array → Separate `order_items` table
4. **Product.sellingInfo**: Embedded → Flattened columns

Migration Steps

1. Export data from MongoDB
2. Transform IDs to UUIDs
3. Normalize embedded documents
4. Import to PostgreSQL
5. Verify data integrity

Appendix B: Performance Considerations

Database Indexes

All entities have appropriate indexes defined via TypeORM decorators:

```
@Index(['businessId', 'code'], { unique: true })
@Index(['businessId', 'status'])
@Index(['createdAt'])
```

Caching Strategy

- **Redis**: Session data, stock reservations, real-time events
- **TTLs**: Configurable per data type
- **Invalidation**: Event-driven cache invalidation

Connection Pooling

```
// PostgreSQL pool configuration
{
  type: 'postgres',
  poolSize: 20,
  connectionTimeoutMillis: 5000,
}
```

Appendix C: Version 2.0 Changelog

Completed Implementations

The following features have been fully implemented in this version:

Feature	Description	Status
Barcode SVG Generation	Native SVG generation for EAN-13, Code 128, Code 39, QR codes	✅ Complete
Cache Service Extensions	Redis sorted set operations (zadd, zrangebyscore, zrem, etc.)	✅ Complete
PDF Export	Native PDF 1.4 generation with proper structure	✅ Complete
Excel Export	Native XLSX generation using OOXML specification	✅ Complete
Offline Sync (Redis)	Redis-based persistence replacing in-memory storage	✅ Complete
Tenant Isolation	Multi-tenant security validation in sync service	✅ Complete

SQS Provider	Full AWS SQS integration with FIFO support	✅ Complete
Inventory Audit Trail	Automatic audit logging for all inventory changes	✅ Complete
Realtime Events (Redis)	Redis pub/sub and sorted sets for event persistence	✅ Complete

Technology Stack Summary

```
┌──────────────────────────────────────────────────────────────────────────┐
│                        PRODUCTION-READY STACK                            │
├──────────────────────────────────────────────────────────────────────────┤
│                                                                          │
│  Backend                                                                 │
│  • Node.js 18+ with TypeScript 5.x                                       │
│  • Express.js for HTTP routing                                           │
│  • TypeORM for database abstraction                                      │
│                                                                          │
│  Databases                                                               │
│  • PostgreSQL 15+ (Primary data store, ACID transactions)                │
│  • Redis 7+ (Cache, pub/sub, sorted sets, queues)                        │
│  • MongoDB 6+ (Audit logs, event streams)                                │
│                                                                          │
│  Queue Systems                                                           │
│  • BullMQ (Default, Redis-based)                                         │
│  • AWS SQS (Production, Standard/FIFO queues)                            │
│                                                                          │
│  Real-time                                                               │
│  • Socket.io with Redis adapter                                          │
│  • Guaranteed delivery with acknowledgments                              │
│  • Event persistence for reconnection replay                             │
│                                                                          │
│  Security                                                                │
│  • JWT authentication with refresh tokens                                │
│  • Multi-tenant isolation at all layers                                  │
│  • Comprehensive audit trail                                             │
│                                                                          │
│  Export Formats                                                          │
│  • PDF (Native PDF 1.4)                                                  │
│  • Excel XLSX (Native OOXML)                                             │
│  • CSV                                                                   │
│  • JSON                                                                  │
│                                                                          │
└──────────────────────────────────────────────────────────────────────────┘
```

End of Documentation

Version: 2.1 **Last Updated**: January 2026 **Previous Version**: 2.0 **Maintainer**: Development Team

Change History

Version	Date	Changes
2.1	Jan 2026	Added SVG barcode generation, native PDF/Excel export, Redis sync persistence, SQS provider, inventory audit trail
2.0	Jan 2026	Initial v2 documentation with TypeScript + PostgreSQL stack