Phase 3: Inventory Management Implementation Guide

Overview

This phase addresses all inventory management issues including CRUD operations, metric calculations, filtering, and UI improvements. The inventory system is central to the business module as sales generate revenue transactions.

Context

- Inventory items have SKU as unique identifier
- When items are sold, they create revenue transactions automatically
- Required fields: All except description
- Categories can be predefined or custom-added

3.1 Backend Database Fixes

File: `blueprints/business/utils.py`

A. Remove Auto-Generated Sample Items

In `initialize_business_data()` function:

1. Remove the entire `sample_inventory` array and its insertion loop
2. Keep only category initialization
3. Ensure clean database start

B. Add Category Management

Create new function `add_inventory_category()`:

```
Purpose: Allow dynamic addition of inventory categories
Parameters: category_name, description (optional)
Logic:
1. Check if category exists
2. Insert into appropriate table
3. Return success/error response
```

File: `blueprints/business/routes.py`

C. Fix Inventory Metrics Calculation

In `inventory()` route function:

1. **Total Items Calculation**:

   ```
   - Count all items where is_active = 1
   - Include all statuses (inventory, listed, sold, kept)
   ```

2. **Available Items**:

   ```
   - Count where is_active = 1 AND listing_status = 'inventory'
   ```

3. **Sold Items Count**:

   ```
   - Count where listing_status = 'sold'
   ```

4. **Total Value Calculation**:

   ```
   - For unsold items: SUM(listing_price) WHERE listing_status != 'sold'
   - This represents potential revenue
   ```

5. **Total Cost Calculation**:

   ```
   - SUM(cost_with_tax) for all active items
   - Represents total investment
   ```

3.2 API Endpoint Implementation

File: `blueprints/business/routes.py`

A. Fix Form Validation

In POST `/api/inventory` endpoint:

1. Required fields validation:
   - `brand` - not empty
   - `item_type` - not empty
   - `category` - not empty or allow new
   - `cost` - numeric, >= 0
   - `listing_price` - numeric, > 0
   - `date_added` - valid date or default to today
   - `listing_status` - default to 'inventory'
   - `description` - optional
2. Auto-generate SKU if not provided:

   ```
   Format: ITEM-YYYY-XXXX (where XXXX is sequential)
   Example: ITEM-2025-0001
   ```

B. Implement Working Edit Functionality

PUT `/api/inventory/<sku>` endpoint:

1. Fetch existing item by SKU
2. Validate item exists and is not sold (sold items are read-only)
3. Update only provided fields
4. Maintain original SKU (cannot be changed)
5. Update `updated_at` timestamp
6. Return updated item data

C. Implement Delete Functionality

DELETE `/api/inventory/<sku>` endpoint:

1. Check item exists
2. Prevent deletion of sold items (historical record)
3. Allow deletion of unsold items only
4. Return appropriate error messages

D. Fix Mark as Sold Functionality

POST `/api/inventory/<sku>/sell` endpoint:

1. Show modal to request final selling price
2. Required validation:
   - `sold_price` must be numeric and > 0
   - `sold_date` defaults to today
3. Update inventory item:
   - `listing_status` = 'sold'
   - `sold_price` = user input
   - `sold_date` = current date
4. Create business transaction:
   - `transaction_type` = 'Income'
   - `amount` = sold_price
   - `category` = 'Sales Revenue'
   - `sub_category` = item category
   - `description` = "Sold: {brand} {item_type}"

- Link to inventory SKU for tracking

3.3 Frontend Implementation

File: `static/js/business.js`

A. Fix Filter Functionality

In `filterInventory()` function:

1. **Search Filter**:

```
- Search in: brand, item_type, description, SKU
- Case-insensitive matching
- Real-time filtering as user types
```

2. **Category Filter**:

```
- Match exact category
- Include "All Categories" option
- Populate from existing categories
```

3. **Status Filter**:

```
- Options: All, Available, Listed, Sold, Kept
- Map to listing_status field
```

4. **Size Filter**:

```
- Match exact size
- Include "All Sizes" option
```

5. **Combined Filtering**:

```
- Apply all active filters simultaneously
- Update results count
- Show "No results" message when empty
```

B. Implement Delete Button

Add to each inventory row:

1. Delete button with confirmation dialog
2. Prevent deletion of sold items (show error)
3. Update table after successful deletion
4. Show success/error messages

C. Fix Save Item Functionality

In `saveItem()` function:

1. Validate all required fields before submission
2. Show specific validation errors
3. Properly serialize form data
4. Handle both create and update modes
5. Refresh inventory list after save

D. Implement Sell Item Modal

Create new modal for selling:

1. Show current listing price for reference
2. Input for final selling price (required)
3. Optional notes field
4. Confirm button submits sale
5. Success message and table refresh

File: `templates/business/business_inventory.html`

E. Add Delete Button to Table

In actions column:

```html
- Add delete button with trash icon
- Include onclick handler: deleteItem(sku)
- Style: btn-outline-danger btn-sm
- Show only for unsold items
```

F. Style Details Modal

Copy from assets implementation:

1. Use same modal structure as assets detail view
2. Display all item information in organized layout
3. Include proper close button
4. Make modal responsive

G. Add Selling Price Modal

New modal structure:

```html
- Modal ID: sellItemModal
- Input field for price with currency formatting
- Show item details for context
- Validation messages container
- Submit and cancel buttons
```

3.4 Data Validation Rules

Required Field Validation

1. **Brand**: Min 1 character, max 100
2. **Item Type**: Min 1 character, max 100
3. **Category**: Must be from list or new (validated)
4. **Cost**: Numeric, >= 0, max 2 decimal places
5. **Listing Price**: Numeric, > 0, max 2 decimal places
6. **Status**: Must be valid enum value

Business Rules

1. Cannot edit sold items (read-only)
2. Cannot delete sold items (historical record)
3. SKU must be unique (auto-generated if not provided)
4. Selling price can differ from listing price
5. Cost with tax auto-calculates if not provided (cost * 1.08)

3.5 Testing Checklist

CRUD Operations

- [ ] Create item with all fields
- [ ] Create item with only required fields
- [ ] Edit unsold item - all fields update
- [ ] Cannot edit sold item
- [ ] Delete unsold item works
- [ ] Cannot delete sold item

Inventory Metrics

- [ ] Total items count is accurate
- [ ] Available items excludes sold
- [ ] Total value sums listing prices correctly

- [ ] Total cost sums cost_with_tax correctly

Filtering

- [ ] Search filters across all text fields
- [ ] Category filter works
- [ ] Status filter works
- [ ] Size filter works
- [ ] Combined filters work together
- [ ] Clear filters resets view

Selling Process

- [ ] Mark as sold opens price modal
- [ ] Validates selling price > 0
- [ ] Updates item status to sold
- [ ] Creates revenue transaction
- [ ] Updates metrics immediately

Error Handling

1. **Validation Errors**: Show field-specific messages
2. **Database Errors**: Generic message with console logging
3. **Network Errors**: Retry mechanism with user feedback
4. **Concurrent Updates**: Last-write-wins with warning

Performance Optimization

1. Implement pagination for large inventories
2. Cache category lists
3. Debounce search input (300ms delay)
4. Lazy load sold items tab
5. Index SKU and status fields in database

Notes for Implementation

- All prices should be stored as DECIMAL(10,2)
- Use database transactions for sell operation (inventory update + transaction create)
- Implement soft delete option for future (is_active flag)
- Consider barcode/QR code field for future scanning
- Add image upload capability in future phase