Phase 5: Assets Management Implementation Guide

Overview

This phase completes the asset management functionality including CRUD operations, disposal tracking, and proper expense transaction creation. Assets represent business equipment and purchases that should create expense transactions.

Context

- Assets have unique IDs (auto-incremented)
- Required fields: name, category, type, purchase_date, purchase_price, status
- Description is optional
- Assets can be marked as disposed but maintain historical record
- Asset purchases should create expense transactions automatically

5.1 Backend Implementation

File: `blueprints/business/routes.py`

A. Implement Asset CRUD Operations

1. GET `/api/assets` endpoint:

```
Purpose: Retrieve all assets
Logic:
1. Query business_assets table
2. Include both active and disposed assets
3. Order by purchase_date DESC
4. Return JSON array with all fields
```

2. GET `/api/assets/<id>` endpoint:

```
Purpose: Get single asset details
Logic:
1. Query by asset ID
2. Return 404 if not found
3. Include all asset fields
4. Format dates for display
```

3. POST `/api/assets` endpoint:

```
Purpose: Create new asset
Required fields:
- name (string, 1-100 chars)
- asset_category (from allowed list or new)
- asset_type (string, 1-50 chars)
- purchase_date (valid date)
- purchase_price (decimal > 0)
- is_active (default true)
Optional fields:
- description (text)
- location (string)
- vendor (string)
- serial_number (string)
- warranty_expiry (date)

Logic:
1. Validate all required fields
2. Insert into business_assets table
3. Create expense transaction:
   - amount = purchase_price
   - category = 'Equipment & Supplies' or based on asset_category
   - description = "Asset Purchase: {name}"
   - transaction_type = 'Expense'
   - source_type = 'asset_purchase'
   - source_id = new asset ID
4. Return created asset with ID
```

4. PUT `/api/assets/<id>` endpoint:

```
Purpose: Update existing asset
Logic:
1. Fetch existing asset
2. Validate asset exists
3. Update only provided fields
4. Cannot change purchase_price (historical record)
5. Update updated_at timestamp
6. Return updated asset
```

5. POST `/api/assets/<id>/dispose` endpoint:

```
Purpose: Mark asset as disposed
Required in request body:
- disposal_date (valid date)
Optional:
- disposal_value (decimal >= 0)
- disposal_reason (text)

Logic:
1. Validate asset exists and is active
2. Update asset:
   - is_active = false
   - disposal_date = provided date
   - disposal_value = provided value or 0
3. Do NOT delete the asset (maintain history)
4. Optionally create transaction if disposal_value > 0
5. Return success response
```

6. DELETE `/api/assets/<id>` endpoint:

```
Purpose: Permanently delete asset
Logic:
1. Check if asset exists
2. Check if asset has related transactions
3. If has transactions, return error (maintain integrity)
4. If no transactions, perform hard delete
5. Return success response
```

B. Fix Asset Calculations

In `assets()` route function:

1. **Fix asset metrics calculation**:

   ```
   - total_assets: COUNT(*) WHERE is_active = 1
   - total_purchase_value: SUM(purchase_price) WHERE is_active = 1
   - total_current_value: Calculate depreciation if needed
   - disposed_count: COUNT(*) WHERE is_active = 0
   ```

2. **Fix assets by category**:

   ```
   - Group assets by asset_category
   - Calculate count and total value per category
   - Include only active assets
   ```

5.2 Frontend Implementation

File: `static/js/business-assets.js`

A. Fix Save Asset Function

In `saveAsset()` function:

1. Validate all required fields:
   - name: not empty
   - asset_category: selected
   - asset_type: not empty
   - purchase_date: valid date
   - purchase_price: numeric > 0
2. Show specific validation errors
3. Make proper API call:
   - POST for new assets
   - PUT for updates
   - Include all form fields
4. Handle response:
   - Show success message
   - Close modal
   - Refresh asset list

B. Fix Edit Asset Function

In `editAsset()` function:

1. Fetch asset details via API
2. Populate form with existing values
3. Disable purchase_price field (read-only)
4. Show modal with "Update" button
5. Handle update submission

C. Fix Dispose Asset Function

In `confirmDisposeAsset()` function:

1. Validate disposal_date is provided
2. Make API call to dispose endpoint
3. Update UI immediately:

    - Change row styling
    - Update status badge
    - Remove dispose button

4. Show success message

D. Implement Delete Asset Function

Create `deleteAsset()` function:

1. Show confirmation dialog
2. Make DELETE API call
3. Handle errors (e.g., has transactions)
4. Remove row from table on success
5. Update metrics

File: `templates/business/business_assets.html`

E. Add Delete Button

In actions column:

```html
Add delete button:
- Icon: 🗑️ or trash icon
- Style: btn-outline-danger btn-sm
- Onclick: deleteAsset(id)
- Tooltip: "Delete Asset"
- Show only for assets without transactions
```

F. Fix Modal Implementations

1. Edit Modal:

- Make purchase_price field readonly
- Show original purchase info
- Allow editing other fields

2. Dispose Modal:

- Default disposal_date to today
- Add optional disposal_value field
- Add optional reason field

3. Delete Confirmation:

- Create new modal for delete confirmation
- Show asset name and warning
- Explain about transaction integrity

5.3 CSS Improvements

File: `static/css/business-assets.css`

Apply Consistent Styling

1. **Detail Modal Styling**:

   ```css
   - Use card-based layout
   - Proper spacing between fields
   - Consistent label styling
   - Responsive design
   ```

2. **Status Indicators**:

   ```css
   - Active assets: normal styling
   - Disposed assets: grayed out/muted
   - Add visual indicators for status
   ```

3. **Category Badges**:

   ```css
   - Consistent colors per category
   - Proper padding and margins
   - Readable text contrast
   ```

5.4 Data Validation

Required Field Validation

1. **Name**:
   - Min 1 character
   - Max 100 characters
   - No special characters that break SQL
2. **Asset Category**:
   - Must be from list or validated new category

- Options: Marketing, Technology, Furniture, Other

3. **Asset Type**:
   - Min 1 character
   - Max 50 characters
   - Examples: "POS Equipment", "Display Rack", "Computer"
4. **Purchase Date**:
   - Valid date format
   - Cannot be future date
   - Format: YYYY-MM-DD
5. **Purchase Price**:
   - Numeric value > 0
   - Max 2 decimal places
   - Max value: 999999.99

Business Rules

1. Cannot edit purchase price after creation
2. Cannot dispose already disposed assets
3. Cannot delete assets with transactions
4. Disposal date must be >= purchase date
5. All monetary values use DECIMAL(10,2)

5.5 Integration Points

Asset Purchase → Expense Transaction

When creating new asset:

1. Begin database transaction
2. Insert asset record
3. Get new asset ID
4. Create business_transaction:
   - Link to asset via source_id
   - Set appropriate category
   - Use purchase price as amount
5. Commit transaction or rollback on error

Asset Disposal → Optional Transaction

If disposal generates income:

1. Update asset status
2. If disposal_value > 0:
    - Create income transaction
    - Category: "Asset Disposal"
    - Link to asset

5.6 Testing Checklist

CRUD Operations

- [ ] Create asset with all fields
- [ ] Create asset with only required fields
- [ ] Edit asset (except purchase price)
- [ ] Dispose asset with date
- [ ] Delete asset without transactions
- [ ] Cannot delete asset with transactions

Data Validation

- [ ] All required fields enforced
- [ ] Purchase price must be positive
- [ ] Dates validate correctly
- [ ] Category from list or new

Integration

- [ ] Asset purchase creates expense transaction
- [ ] Transaction links to asset
- [ ] Disposal updates asset status
- [ ] Metrics update immediately

UI/UX

- [ ] Modals open and close properly
- [ ] Forms validate before submission
- [ ] Success/error messages display
- [ ] Table updates without refresh
- [ ] Disposed assets show different styling

Error Handling

1. **Validation Errors**: Field-specific messages
2. **Duplicate Assets**: Allow (different serial numbers)

3. **Transaction Errors**: Rollback and inform user
4. **Network Errors**: Retry option
5. **Data Integrity**: Prevent orphaned records

Performance Optimization

1. Index asset_category and is_active fields
2. Limit initial load to last 100 assets
3. Implement pagination for large datasets
4. Cache category lists
5. Use database views for complex queries

Future Enhancements

1. Asset depreciation calculation
2. Maintenance schedule tracking
3. Document/receipt attachments
4. Barcode/QR code generation
5. Asset location tracking
6. Warranty expiration alerts
7. Asset transfer between locations
8. Bulk import from CSV

Notes for Implementation

- Use database transactions for multi-table operations
- Implement soft delete as alternative option
- Consider audit trail for asset changes
- Format all currency consistently
- Use ISO date format in database
- Add database constraints for data integrity