Inventory Management System - Project Report

1. Project Overview

The Inventory Management System is a web-based application designed to help businesses maintain accurate records of products and stock levels. The system automates inventory tracking, eliminates manual errors, and provides real-time monitoring of stock levels with automated alerts.

1.1 Objectives

- Automate product and stock management
- Provide real-time inventory monitoring
- Generate low stock alerts
- Enable efficient product search and filtering
- Generate comprehensive inventory reports
- Support multiple product categorization (brand, category, subcategory)

2. Technology Stack

2.1 Backend

- **Framework**: Flask 3.0.0
- **Database**: SQLite3
- **Language**: Python 3.x

2.2 Frontend

- **HTML5**: Structure and layout
- **CSS3**: Styling and responsive design
- **JavaScript**: Dynamic functionality and AJAX requests
- **Font Awesome 6.4.0**: Icons and visual elements

2.3 Architecture

- **Pattern**: RESTful API architecture
- **Communication**: JSON-based API endpoints
- **Data Flow**: Client-side JavaScript communicates with Flask backend via AJAX

3. Database Design

3.1 Schema

Products Table

```
CREATE TABLE products (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    brand TEXT,
    category TEXT NOT NULL,
    subcategory TEXT,
    price REAL NOT NULL,
    quantity INTEGER NOT NULL DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)
```

Fields: - `id`: Unique identifier for each product - `name`: Product name (required) - `brand`: Product brand (optional) - `category`: Product category (required) - `subcategory`: Product subcategory (optional) - `price`: Product price in decimal format - `quantity`: Current stock quantity - `created_at`: Timestamp of product creation

Stock Transactions Table

```
CREATE TABLE stock_transactions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    product_id INTEGER NOT NULL,
    transaction_type TEXT NOT NULL,
    quantity INTEGER NOT NULL,
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (product_id) REFERENCES products (id)
)
```

Fields: - `id`: Unique transaction identifier - `product_id`: Reference to products table - `transaction_type`: 'IN' for stock addition, 'OUT' for stock removal - `quantity`: Quantity involved in transaction - `notes`: Optional transaction notes - `created_at`: Transaction timestamp

3.2 Database Relationships

- One-to-Many: One product can have multiple stock transactions
- Foreign Key Constraint: Ensures referential integrity between products and transactions

4. System Modules

4.1 Product Management Module

Functionality: - Create, Read, Update, Delete (CRUD) operations for products - Product details include: name, brand, category, subcategory, price, and quantity - Search functionality across product names and brands - Multi-level filtering by brand, category, and subcategory

API Endpoints: - `GET /api/products` - Retrieve all products with optional filters - `POST /api/products` - Add new product - `PUT /api/products/<id>` - Update existing product - `DELETE /api/products/<id>` - Delete product

Features: - Real-time product listing with status indicators - Low stock badge display (threshold: 10 units) - Duplicate product detection and cleanup - Sample product data import functionality

4.2 Stock In/Out Module

Functionality: - Record stock purchases (Stock In) - Record stock sales/usage (Stock Out) - Automatic quantity updates - Transaction history tracking - Stock validation (prevents negative quantities)

API Endpoints: - `POST /api/stock/in` - Add stock to product - `POST /api/stock/out` - Remove stock from product

Business Logic: - Stock Out operations validate available quantity before processing - Automatic inventory quantity updates on each transaction - Transaction notes for audit trail - Real-time product quantity updates

4.3 Inventory Monitoring Module

Functionality: - Low stock alert system - Real-time stock status monitoring - Configurable threshold (default: 10 units) - Alert refresh functionality

API Endpoints: - `GET /api/alerts` - Retrieve low stock products

Features: - Automatic detection of products below threshold - Visual alert indicators - Product details with current stock levels - Threshold-based filtering

4.4 Reporting Module

Functionality: - Summary statistics dashboard - Category-wise breakdown - Inventory valuation - Recent transaction history - CSV export functionality

API Endpoints: - `GET /api/reports/summary` - Get comprehensive report data - `GET /api/reports/export-csv` - Export inventory as CSV

Report Components: - Total products count - Total inventory value (sum of price × quantity) - Low stock items count - Category statistics (product count, total quantity, total value) - Recent transactions (last 10)

CSV Export Includes: - All product details (ID, Name, Brand, Category, Subcategory, Price, Quantity, Status, Total Value) - Summary statistics - Category breakdown

5. User Interface

5.1 Design Features

- **Color Scheme**: Blue/teal gradient theme (avoiding purple as per requirements)
- **Icons**: Font Awesome icons throughout (no AI-generated symbols)
- **Layout**: Tab-based navigation for different modules
- **Responsive Design**: Mobile-friendly layout with grid system

5.2 Interface Components

Header: - Application title with icon - Branded header with gradient background

Navigation Tabs: - Products Tab: Product management interface - Stock Management Tab: Stock in/out operations - Monitoring Tab: Low stock alerts - Reports Tab: Statistics and analytics

Filter System: - Sticky filter container (pinned to top while scrolling) - Search input for product names and brands - Brand dropdown filter - Category dropdown filter - Subcategory dropdown filter (dynamic based on category) - Clear all filters button

Product Table: - Columns: ID, Name, Brand, Category, Subcategory, Price, Quantity, Status, Actions - Status badges (Low Stock / In Stock) - Action buttons (Edit, Delete) - Responsive table design

Modals: - Add Product Modal: Form for new product entry - Edit Product Modal: Form for product updates - Both modals include all product fields including brand and subcategory

6. API Documentation

6.1 Product Endpoints

Get Products

```
GET /api/products?search=<term>&category=<cat>&subcategory=<sub>&brand=
<brand>
```

Returns filtered list of products.

Add Product

```
POST /api/products
Content-Type: application/json

{
    "name": "Product Name",
    "brand": "Brand Name",
    "category": "Category",
    "subcategory": "Subcategory",
    "price": 99.99,
    "quantity": 10
}
```

Update Product

```
PUT /api/products/<id>
Content-Type: application/json

{
    "name": "Updated Name",
    "brand": "Brand",
    "category": "Category",
    "subcategory": "Subcategory",
    "price": 89.99
}
```

Delete Product

```
DELETE /api/products/<id>
```

6.2 Stock Endpoints

Stock In

```
POST /api/stock/in
Content-Type: application/json

{
    "product_id": 1,
    "quantity": 10,
    "notes": "Purchase order #123"
}
```

Stock Out

```
POST /api/stock/out
Content-Type: application/json

{
    "product_id": 1,
    "quantity": 5,
    "notes": "Sale to customer"
}
```

6.3 Utility Endpoints

Get Categories

```
GET /api/categories
```

Returns distinct categories from products.

Get Subcategories

```
GET /api/subcategories?category=<category>
```

Returns subcategories, optionally filtered by category.

Get Brands

```
GET /api/brands
```

Returns distinct brands from products.

Get Alerts

```
GET /api/alerts
```

Returns products with quantity below threshold.

Get Report Summary

```
GET /api/reports/summary
```

Returns comprehensive report data.

Export CSV

```
GET /api/reports/export-csv
```

Downloads inventory report as CSV file.

Seed Database

```
POST /api/seed
```

Adds sample products to empty database.

Cleanup Duplicates

```
POST /api/cleanup-duplicates
```

Removes duplicate products (keeps first occurrence).

7. Business Logic

7.1 Stock Management Rules

1. **Stock In**: Always increases product quantity
2. **Stock Out**: Validates available quantity before processing
3. **Negative Stock Prevention**: System prevents stock out if quantity exceeds available stock
4. **Transaction Logging**: All stock movements are recorded with timestamp

7.2 Low Stock Alert System

- **Threshold**: Configurable (default: 10 units)
- **Automatic Detection**: System checks quantity on every product load

- **Visual Indicators**: Status badges show “Low Stock” for products below threshold
- **Alert List**: Dedicated monitoring tab displays all low stock items

7.3 Data Validation

- Required fields: Product name, category, price
- Price validation: Must be positive decimal number
- Quantity validation: Must be non-negative integer
- Stock out validation: Quantity cannot exceed available stock

8. Installation and Setup

8.1 Prerequisites

- Python 3.7 or higher
- pip (Python package manager)

8.2 Installation Steps

1. **Install Dependencies**

   ```
   pip install -r requirements.txt
   ```

2. **Run Application**

   ```
   python app.py
   ```

3. **Access Application**

   - Open browser and navigate to: `http://localhost:5000`

8.3 Initial Setup

- Database is automatically created on first run
- Sample products can be loaded using “Load Sample Products” button
- Database file: `inventory.db` (SQLite)

9. File Structure

```
project5_inventory/
├── app.py                  # Flask application and API routes
├── requirements.txt        # Python dependencies
├── README.txt             # User documentation
├── PROJECT_REPORT.md      # This report
├── inventory.db           # SQLite database (created on first run)
├── templates/
│   └── index.html         # Main HTML template
└── static/
    ├── style.css          # Stylesheet
    └── script.js          # JavaScript functionality
```

10. Key Features Implementation

10.1 Multi-Level Filtering

The system implements hierarchical filtering: - Brand → Category → Subcategory - Filters work independently or in combination - Subcategory filter dynamically updates based on selected category - Search works across product names and brands

10.2 Real-Time Updates

- AJAX-based communication for instant updates
- No page refresh required for most operations
- Automatic filter refresh after product modifications
- Live status updates for stock levels

10.3 Data Organization

- Products organized by: Brand → Category → Subcategory → Name
- Consistent sorting across all views
- Hierarchical data structure for better organization

10.4 Error Handling

- Client-side validation for user input
- Server-side validation for data integrity
- User-friendly error messages
- Transaction rollback on errors

11. Security Considerations

- Input sanitization for SQL queries (parameterized queries)
- XSS prevention through proper data escaping
- CSRF protection via Flask secret key
- SQL injection prevention through parameterized statements

12. Performance Optimizations

- Efficient database queries with proper indexing
- Minimal database connections (connection per request)
- Client-side filtering where applicable
- Optimized SQL queries with proper WHERE clauses

13. Testing and Validation

13.1 Functional Testing

- Product CRUD operations
- Stock in/out transactions
- Filter functionality
- Search functionality
- Report generation
- CSV export

13.2 Data Validation Testing

- Required field validation
- Price and quantity validation
- Stock out quantity validation
- Duplicate product handling

14. Future Enhancements

Potential improvements for future versions:

- User authentication and authorization
- Multi-user support with role-based access
- Advanced reporting with charts and graphs
- Barcode scanning integration

- Email notifications for low stock
- Product image support
- Batch import/export functionality
- Inventory history and audit trail
- Supplier management
- Purchase order management

15. Technical Specifications

15.1 Database Configuration

- **Database Type**: SQLite3
- **File Location**: `inventory.db`
- **Connection**: Per-request connection pattern
- **Row Factory**: sqlite3.Row for dictionary-like access

15.2 Application Configuration

- **Debug Mode**: Enabled for development
- **Secret Key**: Configured for session management
- **Port**: Default Flask port (5000)
- **Host**: Localhost

15.3 Frontend Configuration

- **Icons**: Font Awesome CDN (version 6.4.0)
- **No External Frameworks**: Pure JavaScript implementation
- **CSS**: Custom stylesheet with responsive design
- **Browser Compatibility**: Modern browsers (Chrome, Firefox, Edge, Safari)

16. Code Organization

16.1 Backend Structure

- **Routes**: Organized by functionality (products, stock, reports, utilities)
- **Database Functions**: Centralized connection management
- **Error Handling**: Consistent error response format
- **Data Validation**: Input validation at API level

16.2 Frontend Structure

- **Event Handlers**: Organized by module
- **API Calls**: Centralized fetch functions
- **DOM Manipulation**: Separate functions for UI updates
- **State Management**: Client-side state through DOM

17. Conclusion

The Inventory Management System successfully implements all required functionality for efficient inventory tracking and management. The system provides a user-friendly interface, robust data management, and comprehensive reporting capabilities. The modular architecture allows for easy maintenance and future enhancements.