Project Documentation: Inventory & Billing Management System

Project Name: Inventrobil-Web

Target Audience: Shop Owners/Staff

Goal: To streamline inventory tracking, stock management, billing, and financial calculations with a user-friendly, scalable web interface.

1. System Architecture & Tech Stack

To ensure the application is robust and scalable, the following stack is recommended:

- **Backend:** Python (Flask or Django) or Node.js (Express). *Recommendation: Python with Flask/FastAPI for easy integration with image processing libraries.*
- **Frontend Integration:** * **JinJa2 Templates:** Standard server-side rendering for the dashboard and inventory.
    - ❖ **Bootstrap 5:** For responsive UI styling via CDN.
    - ❖ **React.js:** Included via CDN within specific template files (like Billing) to handle dynamic cart logic without a separate build process.
- **Database:** PostgreSQL (Production) or SQLite (Development).
- **Image Storage:** AWS S3, Cloudinary, or local static file storage (if hosting locally).
- **Search API (for Auto-Images):** Google Custom Search API or Bing Image Search API.

2. Core Features & Functionalities

A. User Authentication & Roles

- **Admin:** Full access to settings, financial reports, user management, and inventory editing.
- **Staff:** Access to billing and basic inventory viewing (restricted from deleting stock or viewing profit margins).
- **Login System:** Secure JWT-based authentication.

B. Inventory Management

- **Product Dashboard:** Searchable table of all items (Name, Category, Stock Level, Price, SKU).
- **Add/Edit Product:** Form to input details.
- **Add,Delete,Update,Sort,Filter:** Options that do all the basic functions on the data

- **CRUD Operations:** Full ability to Add, Delete, Update, Sort, and Filter product data.
- **Low Stock Alerts:** Visual indicators (e.g., red row highlights) when stock dips below a defined threshold.
- **Categories:** Separate sections for "Plumbing" and "Electronics" to handle different units (e.g., meters for wires/pipes vs. pieces for switches).

C. Smart Image Handling (The "Auto-Image" Feature)

- **Manual Upload:** Admin can upload a specific image during product creation.
- **Fallback Logic:** If no image is uploaded:
    1. The system takes the Product Name + Brand.
    2. Sends a query to an Image Search API (e.g., Bing/Google).
    3. Fetches the first relevant URL.
    4. Save this URL/Image to the database automatically.
- **Default Placeholder:** If the API fails, a generic "No Image Available" icon is used.

D. Billing & Invoicing

- **POS Interface:** A quick-add interface where staff can scan barcodes or search items to add to a cart.
- **Auto-Calculation:** Real-time calculation of Subtotal, GST/Tax, Discounts, and Grand Total.
- **Invoice Generation:** Generate a printable PDF receipt with shop branding.
- **Stock Deduction:** Automatically reduces inventory counts upon checkout.

E. Financial Reporting

- **Daily/Monthly Sales:** Visual graphs showing revenue trends.
- **Top Selling Items:** List of most popular products.
- **Profit/Loss:** Calculation based on (Selling Price - Cost Price).

3. Database Schema (ER Diagram Concepts)

Table: Products

Column Name	Type	Description
id	Integer (PK)	Unique Product ID

name	String	e.g., "1/2 inch PVC Pipe"
category	String	Plumbing / Electronics
price_cost	Decimal	Purchase price
price_sell	Decimal	Selling price
stock_qty	Integer	Current quantity
unit	String	pieces, meters, kg
image_url	String	Path to image or URL
min_stock	Integer	Threshold for alert

Table: Invoices

Column Name	Type	Description
invoice_id	String (PK)	Unique Invoice #
created_at	DateTime	Timestamp
total_amount	Decimal	Final bill amount

payment_mode	String	Cash/Card/UPI
staff_id	Integer (FK)	Who processed the sale

Table: InvoiceItems

Column Name	Type	Description
id	Integer (PK)	
invoice_id	String (FK)	Links to Invoice
product_id	Integer (FK)	Links to Product
quantity	Integer	Amount sold
price_at_sale	Decimal	Price at moment of sale

4. Project Structure

```
InventroWeb/
├── app.py              # Main Backend (Flask/FastAPI)
├── models.py           # Database Schema (Products, Invoices) [cite: 45, 48]
├── static/             # CSS (Bootstrap), JS (React components), Images
│   ├── css/
│   ├── js/             # React scripts or compiled bundles
│   └── uploads/        # Local image storage
├── templates/          # HTML files (served by the backend)
│   ├── dashboard.html
│   ├── inventory.html
│   └── billing.html    # Where the React POS will live
└── requirements.txt
```

5. Implementation Steps

Phase 1: Setup & Integrated Backend

1. Initialize the repository and set up a Virtual Environment.
2. Configure Flask to serve the `templates` and `static` folders.
3. Create API routes for `/products` and `/bill`.
4. Build the `fetch_product_image` helper for the Auto-Image feature.

Phase 2: Frontend & POS Logic

1. Build the Inventory Page using Bootstrap tables.
2. Create the Billing Page: Use **React** (via CDN) within the `billing.html` template to manage the "Cart" state and "Enter" key listeners.

Phase 3: Deployment

1. Deploy the single Flask app to **Render** or **Heroku**.
2. Ensure the SQLite database or PostgreSQL addon is initialized.
3. Set up automated backups for the database file.