# RetailPulse Docs

**None**

*None*

*None*

# Table of contents

# 1. RetailPulse

A simple, web-based system for billing, inventory, and reports for medical wholesalers.

## 1.1 The Problem

Medical wholesalers face three main challenges:

### 1.1.1 1. Manual Bill Generation is Slow

Managers spend too much time writing bills by hand. This leads to wasted hours, mistakes, and lost paper bills.

### 1.1.2 2. Inventory is Hard to Manage

It's difficult to track stock levels in real-time. This causes expired medicines to go unnoticed, stockouts, and errors in records.

### 1.1.3 3. No Business Visibility

Owners lack clear information on sales, profits, and trends. They spend hours reviewing paper bills and cannot make data-driven decisions.

### 1.1.4 Current Workflow (The Hard Way)

Customer arrives → Manager checks stock by hand → Writes bill on paper → Calculates total → Updates inventory book → Files paper bill → Owner reviews bills weekly.

This process is slow, error-prone, and lacks real-time information.

## 1.2 The Solution – RetailPulse

RetailPulse is a web-based system that automates medical wholesale operations through three parts:

1. **Bill Generation**: Fast digital billing with automatic PDF creation.
2. **Inventory Management**: Real-time stock tracking, including expiry dates and batches.
3. **Analytics**: Real-time business insights and performance reports.

### 1.2.1 Transformed Workflow

Customer arrives → Manager creates a digital bill (1 min) → System automatically updates inventory → Bill is saved & PDF generated → Owner sees real-time reports.

## 1.3 Who Uses This System?

| Role | Access Level | What they can do |
|---|---|---|
| Store Manager | Limited | Create bills, view stock, download PDF bills, search products. |
| Admin (Owner) | Full | View all bills, manage inventory, access reports, manage users. |

# 2. Functional Documentation

## 2.1 User Roles

| Role | Access | What they can do |
|---|---|---|
| Admin | Full | See reports, profits, stock trends, add/change/delete products, view all bills, manage users |
| Store Manager | Limited | Create bills, view stock, download PDF bills, search products |

## 2.2 System Modules Overview

RetailPulse has three main parts that help medical wholesale businesses.

## 2.3 Module 1: Bill Generation

### 2.3.1 Purpose

To help store managers make bills quickly and correctly.

### 2.3.2 Features

**1.1 Product Search and Selection**

- Search: Find medicines by name, maker, or type.
- Product Details: See stock, price, and expiry when picking a product.
- Multi-Product: Add many products to one bill.

**1.2 Real-time Stock Validation**

- Availability: Check if a product is in stock before adding to a bill.
- Quantity: Make sure the amount asked for is not more than what's available.

**1.3 Bill Creation Form**

- Customer Info: Add customer name, contact.
- Product List: Shows product name, unit price, quantity, and total price.
- Calculations: Shows subtotal, discount, tax, and grand total.

**1.4 Bill Numbering and Tracking**

- Unique Numbers: Bills get an automatic, unique number.
- Date/Time: Records when the bill was made.
- Created By: Shows which manager made the bill.

**1.5 PDF Generation**

Bills are made in a professional PDF format with the store logo. Includes store info, item list, and payment summary. You can download the PDF instantly.

**1.6 Automatic Inventory Update**

- Stock Deduction: Stock amounts are automatically reduced.
- Sales Log: Every sale is recorded for reports.

---

# 2.4 Module 2: Inventory Management

## 2.4.1 Purpose

To give admins full control over products and stock, with alerts for low or expiring items.

## 2.4.2 Features

**2.1 Product Creation**

- Basic Info: Add product name, category, manufacturer, and description.
- Pricing: Set cost price, selling price, and see profit margin.

**2.2 View Inventory**

- List View: See all products in a sortable table with key details.
- Search and Filter: Find products by name, maker, category, or stock status.

**2.3 Update Inventor**

- Edit Details: Change product prices, reorder levels, or descriptions.
- Stock Adjustments: Manually add or reduce stock, with reasons and a record of who did it.

**2.4 Delete Product**

- Delete: permanently removed.

---

# 2.5 Module 3: Analytics Dashboard

## 2.5.1 Purpose

To give admins real-time business insights for smart decisions.

## 2.5.2 Features

**3.1 Sales Analytics**

- Revenue: Shows total revenue and trends over time.
- Bills: Shows number of bills, average bill value, and peak hours.
- Profit: Shows total profit, profit margin, and trends.

**3.2 Product Performance**

- Best Sellers: Lists top products by sales, revenue, and profit.
- Slow Movers: Identifies products with low sales.
- Profitability: Shows products with high profit margins or those losing money.

**3.3 Inventory Insights**

- Stock Summary: Shows total inventory value, unique products, and low/out-of-stock items.
- Categories: Shows stock and sales by category.

**3.4 Time Range Filters**

- Quick Views: See data for today, this week, this month, or this year.
- Custom Dates: Pick your own start and end dates.

**3.5 Visual Dashboards**

- Charts: Uses line, bar, and pie charts for easy understanding.
- Interactive: Hover for details, click to see more.

---

## 2.6 Module 4: Authentication and Authorization

### 2.6.1 Purpose

To secure the system and ensure users only access what they are allowed to.

### 2.6.2 Features

**4.1 User Registration**

- Admin Only: Only admins can create new user accounts.
- User Details: Admins add name, username, password, and role.

**4.2 User Login**

- Methods: Log in with username and password.
- Session: Manages user sessions and automatic logouts.

**4.3 Role-Based Access Control**

- Manager Role: Can create bills, view stock, and see their own bill history.
- Admin Role: Has all manager permissions plus full control over inventory, analytics, and user management.

# 3. Use Cases

## 3.1 Primary Use Cases

### 3.1.1 UC-01: Store Manager Creates a Bill

- **Actor**: Store Manager
- **Goal**: To create a bill for a customer quickly.
- **Flow**:
  a. Manager goes to the "Create Bill" page.
  b. Searches for a product and adds it to the bill.
  c. Enters the quantity.
  d. Repeats for all products.
  e. Enters customer name and contact (optional).
  f. Clicks "Generate Bill".
  g. The system saves the bill, updates the stock, and creates a PDF.
  h. Manager can download the PDF.
- **Alternative**: If the quantity is more than the stock, the system shows an error.

---

### 3.1.2 UC-02: Admin Adds New Medicine to Inventory

- **Actor**: Admin
- **Goal**: To add a new product to the system.
- **Flow**:
  a. Admin goes to "Inventory Management" and clicks "Add New Product".
  b. Fills in product details: name, category, cost price, and selling price.
  c. Clicks "Save Product".
  d. The product is now in the inventory and can be sold.
- **Alternative**: If the product name already exists, the system shows an error.

---

### 3.1.3 UC-03: Admin Views Sales Analytics

- **Actor**: Admin
- **Goal**: To see how the business is performing.
- **Flow**:
  a. Admin goes to the "Analytics Dashboard".
  b. The system shows key numbers like Total Revenue, Total Profit, and Bills Count.
  c. The system shows charts for sales trends and top-selling products.
  d. Admin can change the time range (e.g., last 30 days) to see different data.

---

### 3.1.4 UC-04: Admin Updates Product Stock Manually

- **Actor**: Admin

- **Goal**: To manually change the stock quantity of a product.
- **Flow**:
  a. Admin finds a product in the inventory and clicks "Edit Stock".
  b. Enters the new quantity and a reason for the change (e.g., "New purchase").
  c. Clicks "Save Adjustment".
  d. The system updates the stock quantity and logs the change.
- **Alternative**: The system will not allow the stock to go below 0.

---

### 3.1.5 UC-05: Manager Views Their Bill History

- **Actor**: Store Manager
- **Goal**: To see a list of bills they have created.
- **Flow**:
  a. Manager goes to the "My Bills" page.
  b. The system shows a list of their past bills, with date, customer name, and total.
  c. Manager can click on a bill to see full details or download the PDF again.

---

### 3.1.6 UC-06: Admin Manages User Accounts

- **Actor**: Admin
- **Goal**: To create or deactivate user accounts.
- **Flow (Create)**:
  a. Admin goes to "User Management" and clicks "Add New User".
  b. Enters the user's name, username, role (Admin or Manager), and a password.
  c. Clicks "Create User". The new user can now log in.
- **Flow (Deactivate)**:
  a. Admin finds a user and clicks "Deactivate".
  b. The user's account is turned off and they can no longer log in.
- **Alternatives**:
  - Usernames must be unique.
  - An admin cannot deactivate their own account or the last remaining admin account.

---

### 3.1.7 UC-07: User Logs In

- **Actor**: Admin or Store Manager
- **Goal**: To log into the system.
- **Flow**:
  a. User goes to the login page.
  b. Enters their username and password.
  c. Clicks "Login".
  d. The system logs them in and shows the correct dashboard for their role.

- **Alternatives**:
  - If the username or password is wrong, an error is shown.
  - If the account is deactivated, login fails.

# 4. Technical Docs

## 4.1 Tech Stack

**Frontend**

| Technology | Purpose |
|---|---|
| React | UI framework |
| TypeScript | Type safety |
| React Router | Page navigation |
| Material UI | UI components |
| Recharts | Charts |
| Axios | API requests |

**Backend**

| Technology | Purpose |
|---|---|
| Python | Programming language |
| FastAPI | Web framework |
| SQLAlchemy | Database interaction (ORM) |
| PostgreSQL | Database |
| ReportLab | PDF generation |
| Pytest | Testing |
| Uvicorn | Web server |

**DevOps and Deployment**

| Technology | Purpose |
|---|---|
| Docker | App containerization |
| NGINX | Reverse proxy |
| GitHub Actions | CI/CD (testing & deployment) |
| AWS | Cloud hosting |

# 4.2 Data Flow Diagrams

## 4.2.1 1. Bill Creation Flow

This shows the steps for creating a bill.

```
%%{init: {'theme':'base', 'themeVariables': { 'fontSize':'38px', 'fontFamily':'Arial'}}}%%
sequenceDiagram
    actor Manager
    participant UI as React UI
    participant API as Backend
    participant DB as Database

    Manager->>UI: 1. Fills out the bill form
    UI->>API: 2. Sends bill data
    API->>DB: 3. BEGIN TRANSACTION
    API->>DB: 4. Save bill details
    API->>DB: 5. Update product stock
    API->>DB: 6. COMMIT TRANSACTION
    API-->>UI: 7. Confirms bill was created
    UI-->>Manager: 8. Shows success message
```

## 4.2.2 2. User Login Flow

This diagram shows how a user logs in and gets a session token.

```
%%{init: {'theme':'base', 'themeVariables': { 'fontSize':'38px', 'fontFamily':'Arial'}}}%%
sequenceDiagram
    actor User
    participant UI as React UI
    participant API as Backend
    participant DB as Database

    User->>UI: 1. Enters username and password
    UI->>API: 2. Sends credentials
    API->>DB: 3. Queries for user with matching credentials
    DB-->>API: 4. Returns user data (with role)
    alt Credentials are valid
        API->>API: 5. Generates session token (JWT)
        API-->>UI: 6. Returns token to UI
        UI->>User: 7. Stores token and redirects to dashboard
    else Credentials are invalid
        API-->>UI: 8. Returns authentication error
        UI-->>User: 9. Shows error message
    end
```

## 4.2.3 3. Product Creation Flow

This flow describes how an Admin adds a new product to the inventory.

```
%%{init: {'theme':'base', 'themeVariables': { 'fontSize':'38px', 'fontFamily':'Arial'}}}%%
sequenceDiagram
    actor Admin
    participant UI as React UI
    participant API as Backend
    participant DB as Database

    Admin->>UI: 1. Fills out product creation form
    UI->>API: 2. Sends new product data
    API->>API: 3. Validates product data (e.g., check for required fields)
    alt Data is valid
        API->>DB: 4. Inserts new product into 'Products' table
        DB-->>API: 5. Confirms insertion
        API-->>UI: 6. Returns success message
        UI-->>Admin: 7. Shows "Product Created" notification
    else Data is invalid
        API-->>UI: 8. Returns validation error
        UI-->>Admin: 9. Displays error message on the form
    end
```

## 4.2.4 4. Inventory Update Flow

This diagram shows how an Admin manually adjusts the stock for a product.

```
%%{init: {'theme':'base', 'themeVariables': { 'fontSize':'38px', 'fontFamily':'Arial'}}}%%
sequenceDiagram
```

```
    actor Admin
    participant UI as React UI
    participant API as Backend
    participant DB as Database

    Admin->>UI: 1. Selects product and enters new stock quantity
    UI->>API: 2. Sends product ID, new quantity, and reason for change
    API->>DB: 3. BEGIN TRANSACTION
    API->>DB: 4. Updates stock quantity for the product
    API->>DB: 5. Logs the change in an 'InventoryAdjustments' table
    API->>DB: 6. COMMIT TRANSACTION
    API-->>UI: 7. Confirms stock was updated
    UI-->>Admin: 8. Shows success message
```

### 4.2.5 5. Analytics Dashboard Data Flow

This shows how the analytics dashboard gets its data.

```
%%{init: {'theme':'base', 'themeVariables': { 'fontSize':'38px', 'fontFamily':'Arial'}}}%%
sequenceDiagram
    actor Admin
    participant UI as React UI
    participant API as Backend
    participant DB as Database

    Admin->>UI: 1. Navigates to Analytics Dashboard
    UI->>API: 2. Requests analytics data (with time filters)
    API->>DB: 3. Runs aggregation queries (e.g., SUM, COUNT, GROUP BY) on Bills and Products tables
    DB-->>API: 4. Returns aggregated sales, profit, and inventory data
    API->>API: 5. Formats data for charts (e.g., JSON for Chart.js)
    API-->>UI: 6. Sends formatted data to UI
    UI->>UI: 7. Renders charts and key metrics
    UI-->>Admin: 8. Displays dashboard
```

### 4.2.6 6. Product Details Update Flow

This diagram shows how an Admin updates the core details of a product, such as its price or description.

```
%%{init: {'theme':'base', 'themeVariables': { 'fontSize':'38px', 'fontFamily':'Arial'}}}%%
sequenceDiagram
    actor Admin
    participant UI as React UI
    participant API as Backend
    participant DB as Database

    Admin->>UI: 1. Selects a product and chooses to edit it
    UI->>UI: 2. Loads product details into a form
    Admin->>UI: 3. Modifies product details (e.g., price, name, category)
    UI->>API: 4. Sends updated product data
    API->>API: 5. Validates incoming data
    alt Data is valid
        API->>DB: 6. Updates the product's record in the 'Products' table
        DB-->>API: 7. Confirms the update
        API-->>UI: 8. Returns success message
        UI-->>Admin: 9. Shows "Product Updated" notification
    else Data is invalid
        API-->>UI: 10. Returns validation error
        UI-->>Admin: 11. Displays error message on the form
    end
```

# 4.3 Authentication and Authorization

## 4.3.1 Authentication Flow

A user enters their username and password on the login page. The system checks if the credentials are correct. If they are, the user is logged in and can access the app. If not, an error message is shown.

```
graph TD
    A[User enters credentials on login page] --> B{Are credentials valid?};
    B -- Yes --> C[Log in user];
    B -- No --> G[Show error message];
    C --> D{Check user role};
    D -- Admin --> E[Redirect to Admin Dashboard];
    D -- Manager --> F[Redirect to Manager Dashboard];
```

## 4.3.2 Authorization Rules

The system uses Role-Based Access Control (RBAC) to decide who can do what.

| Action | Manager | Admin |
|---|---|---|
| Create Bills | ✅ | ✅ |
| View Own Bills | ✅ | ✅ |
| View All Bills | ❌ | ✅ |
| View Analytics | ❌ | ✅ |
| Add/Edit Products | ❌ | ✅ |
| Manage Users | ❌ | ✅ |

# 4.4 Schema Design

## 4.4.1 Database Overview

We use a PostgreSQL database to store all our data. It's reliable and keeps our information safe.

## 4.4.2 Entity Relationship Diagram

This diagram shows how the different data tables are connected.

```
erDiagram
    USERS ||--o{ BILLS : creates
    PRODUCTS ||--o{ INVENTORY : has
    PRODUCTS ||--o{ BILL_ITEMS : contains
    BILLS ||--|{ BILL_ITEMS : has
    BILLS ||--o{ TRANSACTIONS : generates
    PRODUCTS ||--o{ TRANSACTIONS : tracks

    USERS {
        int id PK
        varchar username UK
        varchar password_hash
        varchar role
        varchar full_name
        boolean is_active
        timestamp created_at
    }

    PRODUCTS {
        int id PK
        varchar name UK
        varchar category
        varchar manufacturer
        text description
        decimal cost_price
        decimal selling_price
        boolean is_active
        timestamp created_at
        timestamp updated_at
    }

    INVENTORY {
        int id PK
        int product_id FK
        int quantity
        timestamp last_updated
    }

    BILLS {
        int id PK
        varchar bill_number UK
        varchar customer_name
        varchar customer_contact
        decimal total_amount
        decimal discount
        decimal tax_amount
        decimal grand_total
        int created_by FK
        timestamp created_at
        varchar pdf_path
    }

    BILL_ITEMS {
        int id PK
        int bill_id FK
        int product_id FK
        int quantity
        decimal unit_price
        decimal total_price
        decimal cost_price
        decimal profit
    }

    TRANSACTIONS {
        int id PK
        int bill_id FK
        int product_id FK
        int quantity
        decimal revenue
        decimal cost
        decimal profit
        date transaction_date
```

```
        timestamp created_at
    }
```

## 4.4.3 Table Schemas

### 1. users

This table holds information for user login.

```
CREATE TABLE users (
    id              SERIAL PRIMARY KEY,
    username        VARCHAR(50) UNIQUE NOT NULL,
    password_hash   VARCHAR(255) NOT NULL,
    role            VARCHAR(20) NOT NULL CHECK (role IN ('admin', 'manager')),
    full_name       VARCHAR(100),
    is_active       BOOLEAN DEFAULT TRUE NOT NULL,
    created_at      TIMESTAMP DEFAULT NOW() NOT NULL
);
```

- `username` : The name a user types to log in.
- `password_hash` : The user's password, stored safely.
- `role` : What the user is allowed to do (admin or manager).

### 2. products

This table lists all the products we sell.

```
CREATE TABLE products (
    id              SERIAL PRIMARY KEY,
    name            VARCHAR(200) UNIQUE NOT NULL,
    category        VARCHAR(100),
    manufacturer    VARCHAR(100),
    description     TEXT,
    cost_price      DECIMAL(10,2) NOT NULL,
    selling_price   DECIMAL(10,2) NOT NULL,
    is_active       BOOLEAN DEFAULT TRUE NOT NULL,
    created_at      TIMESTAMP DEFAULT NOW() NOT NULL,
    updated_at      TIMESTAMP DEFAULT NOW() NOT NULL
);
```

- `name` : The product's name.
- `cost_price` : How much we pay for the product.
- `selling_price` : How much we sell the product for.

### 3. inventory

This table tracks the stock level for each product.

```
CREATE TABLE inventory (
    id              SERIAL PRIMARY KEY,
    product_id      INTEGER NOT NULL REFERENCES products(id) ON DELETE RESTRICT,
    quantity        INTEGER NOT NULL DEFAULT 0,
    last_updated    TIMESTAMP DEFAULT NOW() NOT NULL
);
```

- `product_id` : Links to the product.
- `quantity` : How much stock is currently available.

### 4. bills

This table stores the main details of each customer bill.

```
CREATE TABLE bills (
    id                  SERIAL PRIMARY KEY,
    bill_number         VARCHAR(50) UNIQUE NOT NULL,
    customer_name       VARCHAR(100),
    customer_contact    VARCHAR(20),
    total_amount        DECIMAL(10,2) NOT NULL,
    discount            DECIMAL(10,2) DEFAULT 0,
    tax_amount          DECIMAL(10,2) DEFAULT 0,
    grand_total         DECIMAL(10,2) NOT NULL,
    created_by          INTEGER NOT NULL REFERENCES users(id),
    created_at          TIMESTAMP DEFAULT NOW() NOT NULL,
    pdf_path            VARCHAR(255)
);
```

- `bill_number` : A unique number for the bill.
- `customer_name` : The customer's name (optional).
- `grand_total` : The final amount the customer pays.
- `created_by` : The user who made this bill.

---

### 5. bill_items

This table lists each product sold within a bill.

```
CREATE TABLE bill_items (
    id              SERIAL PRIMARY KEY,
    bill_id         INTEGER NOT NULL REFERENCES bills(id) ON DELETE CASCADE,
    product_id      INTEGER NOT NULL REFERENCES products(id),
    quantity        INTEGER NOT NULL,
    unit_price      DECIMAL(10,2) NOT NULL,
    total_price     DECIMAL(10,2) NOT NULL,
    cost_price      DECIMAL(10,2),
    profit          DECIMAL(10,2) GENERATED ALWAYS AS (total_price - (cost_price * quantity)) STORED
);
```

- `bill_id` : Links to the bill this item is part of.
- `product_id` : Links to the product that was sold.
- `quantity` : How many units of the product were sold.
- `profit` : The profit made on this item.

---

### 6. transactions

This table is a simplified record of sales, used for quick reports.

```
CREATE TABLE transactions (
    id                  SERIAL PRIMARY KEY,
    bill_id             INTEGER NOT NULL REFERENCES bills(id) ON DELETE CASCADE,
    product_id          INTEGER NOT NULL REFERENCES products(id),
    quantity            INTEGER NOT NULL,
    revenue             DECIMAL(10,2) NOT NULL,
    cost                DECIMAL(10,2) NOT NULL,
    profit              DECIMAL(10,2) NOT NULL,
    transaction_date    DATE NOT NULL,
    created_at          TIMESTAMP DEFAULT NOW() NOT NULL
);
```

- `product_id` : Which product was sold.
- `quantity` : How many units were sold.
- `revenue` : The money earned from this sale.
- `profit` : The profit from this sale.
- `transaction_date` : The date of the sale.

---

## 4.4.4 Summary

This database design is simple and focuses on the core needs of the application: creating bills, managing stock, and seeing reports.

# 4.5 Architecture

## 4.5.1 System Architecture Overview

RetailPulse is a standard web application with three layers:

1. **Frontend**: A React application that users see in their web browser.
2. **Backend**: A FastAPI application that contains all the business logic.
3. **Database**: A PostgreSQL database that stores all the data.

## 4.5.2 High-Level Architecture Diagram

This diagram shows how the parts of the system connect.

```
graph TB
    subgraph Client["Client Layer"]
        Browser[Web Browser]
    end

    subgraph CDN["Static Hosting"]
        React[React App]
    end

    subgraph Application["Application Layer"]
        LB[Load Balancer]
        API1[Backend App 1]
        API2[Backend App 2]
    end

    subgraph Data["Data Layer"]
        DB[(PostgreSQL Database)]
        S3[S3 File Storage]
    end

    Browser --> React
    React -->|HTTPS| LB
    LB --> API1
    LB --> API2
    API1 --> DB
    API2 --> DB
    API1 --> S3
    API2 --> S3
```

## 4.5.3 AWS-Specific Architecture Diagram

This diagram shows a simplified view of the system hosted on AWS, with the frontend on Vercel.

```
graph TB
    subgraph Internet["Internet"]
        Users[Users]
    end

    Frontend[(Vercel Frontend)]

    subgraph AWS["AWS Cloud"]
        subgraph VPC["VPC"]
            subgraph PrivateSubnet["Private Subnet"]
                Backend[Containerized Backend]
                DB[(PostgreSQL Database)]
            end
        end
    end

    Users -->|HTTPS| Frontend
    Frontend -->|HTTPS| Backend
    Backend --> DB
```

## 4.6 API Design

The API is designed in a RESTful way. Here are the main endpoints.

**Authentication Endpoints**

| Method | Endpoint | Description |
|---|---|---|
| `POST` | `/auth/login` | User login to get an access token. |

**Bill Endpoints**

| Method | Endpoint | Description |
|---|---|---|
| `POST` | `/bills` | Create a new bill. |
| `GET` | `/bills` | List bills (Admins see all, Managers see their own). |
| `GET` | `/bills/:id` | Get details for one bill. |

**Product Endpoints**

| Method | Endpoint | Description |
|---|---|---|
| `GET` | `/products` | List all products. |
| `POST` | `/products` | Create a new product (Admin only). |
| `PATCH` | `/products/:id` | Update a product (Admin only). |
| `DELETE` | `/products/:id` | Delete a product (Admin only). |

**Analytics Endpoints**

| Method | Endpoint | Description |
|---|---|---|
| `GET` | `/analytics/dashboard` | Get main dashboard metrics (Admin only). |
| `GET` | `/analytics/sales` | Get sales report data (Admin only). |

**User Management Endpoints**

| Method | Endpoint | Description |
|---|---|---|
| `GET` | `/users` | List all users (Admin only). |
| `POST` | `/users` | Create a new user (Admin only). |
| `PATCH` | `/users/:id` | Update a user (Admin only). |
| `DELETE` | `/users/:id` | Deactivate a user (Admin only). |

# 5. Edge Cases and Error Handling

## 5.1 Overview

This document explains how RetailPulse handles common problems and unexpected situations.

## 5.2 1. Login and Access Problems

- **Problem**: User tries to log in with no username or password.
  - **Solution**: The system shows a "Username and password are required" message.
- **Problem**: User enters the wrong username or password.
  - **Solution**: The system shows an "Invalid username or password" message.
- **Problem**: A deactivated user tries to log in.
  - **Solution**: The system shows a "Your account has been deactivated" message.
- **Problem**: A manager tries to access an admin-only page.
  - **Solution**: The system shows an "Insufficient permissions" message.

## 5.3 2. Bill Creation Problems

- **Problem**: User tries to create a bill with no products.
  - **Solution**: The system requires at least one product to be in the bill.
- **Problem**: A product is not found or is inactive.
  - **Solution**: The system shows a "Product not found or inactive" message.
- **Problem**: The quantity requested is more than the available stock.
  - **Solution**: The system shows an error message like "Only 5 units available".
- **Problem**: The quantity is zero or a negative number.
  - **Solution**: The system requires the quantity to be greater than zero.
- **Problem**: The final bill total is zero.
  - **Solution**: The system requires the bill total to be greater than zero.
- **Problem**: The database connection is lost while creating a bill.
  - **Solution**: The entire transaction is cancelled to prevent partial data. The user is asked to try again.

## 5.4 3. Inventory Management Problems

- **Problem**: Admin tries to create a product with a name that already exists.
  - **Solution**: The system shows a "Product with this name already exists" message.
- **Problem**: Admin tries to create a product with a negative price.
  - **Solution**: The system requires the price to be a positive number.
- **Problem**: Admin tries to delete a product that has been sold in past bills.
  - **Solution**: The system prevents deletion to keep historical sales data intact.

- **Problem**: Admin tries to adjust stock to a negative quantity.
  - **Solution**: The system shows a "Stock cannot be negative" message.
- **Problem**: Admin adjusts stock without giving a reason.
  - **Solution**: The system requires a reason for all stock adjustments to maintain a clear history.

---

## 5.5 4. Analytics Problems

- **Problem**: There is no sales data for the selected date range.
  - **Solution**: The system shows a "No data available for this time period" message.
- **Problem**: The start date is after the end date in a custom date range.
  - **Solution**: The system shows a "Start date must be before end date" message.

---

## 5.6 5. User Management Problems

- **Problem**: Admin tries to create a user with a username that already exists.
  - **Solution**: The system shows a "Username already exists" message.
- **Problem**: Admin tries to deactivate their own account.
  - **Solution**: The system prevents this to ensure there's always an active admin.
- **Problem**: Admin tries to deactivate the last remaining admin account.
  - **Solution**: The system prevents this and asks the admin to promote another user first.

---

## 5.7 Summary

This document helps make sure:

- The system is strong and handles errors well.
- Users get clear messages when something goes wrong.
- The data stays correct and consistent.

# 6. Future Functional Enhancements

- Barcode scanning for quick product selection.
- WhatsApp bill delivery.
- Customer management (credit, purchase history).
- Multi-store support.
- Supplier management.
- Purchase order automation.
- Prescription validation (OCR).
- Loyalty programs and discounts.