RetailPulse Docs

None

None

None

Table of contents

1. RetailPulse

A scalable, web-based system for multi-store billing, inventory, and analytics for medical wholesalers.

1.1 The Problem

As a medical wholesale business grows from a single store to multiple locations, it faces significant operational challenges:

1.1.1 1. Lack of Centralized Control

Owners cannot get a clear, real-time view of performance across all stores. Each location operates in a silo, making it impossible to compare sales, manage stock, or enforce pharmaceutical handling standards effectively.

1.1.2 2. Complex Inventory Management

Tracking medicine inventory becomes a nightmare. It's difficult to know the exact stock levels at each location, leading to overstocking in one store and stockouts in another. Tracking critical details like batch numbers, expiry dates, and storage conditions (e.g., cold storage) across a network of stores is difficult with manual methods.

1.1.3 3. Inefficient Operations

Manual processes like handwriting bills, physically checking stock, and consolidating paper reports are slow, prone to errors, and do not scale as the business expands.

1.1.4 The Old Workflow (The Hard Way)

A pharmacy calls with an order. The clerk manually checks for the medicine. A bill is written on paper. At the end of the week, the store manager sends a report to the head office. The owner has no real-time visibility and can't move surplus stock from one store to another that needs it.

1.2 The Solution – RetailPulse

RetailPulse is a centralized, web-based system that empowers medical wholesalers to manage their entire network of stores from a single platform.

1. **Multi-Store Billing**: Fast, digital billing at each location, with all data instantly available at the head office.
2. **Granular Pharmaceutical Inventory**: Track your entire inventory in real-time, right down to the specific batch, expiry date, and location (e.g., Shelf A1, Cold Storage) in each store.
3. **Centralized Analytics**: Get a bird's-eye view of your entire business or drill down into the performance of a single store.

1.2.1 The New, Transformed Workflow

A customer calls with an order. A clerk creates a digital bill in seconds. The system automatically deducts the medicine from the specific batch in that store's inventory. The sale is instantly reflected in both the store's and the company's analytics. The owner sees everything in real-time.

1.3 Who Uses This System?

The system is designed with a clear role-based structure to support a growing wholesale business.

Role	Scope	Primary Task
Company Admin	All Stores	Oversees the entire business, manages stores, and views company-wide analytics.
Store Manager	Single Store	Manages day-to-day operations and staff for their assigned store.
Stockist	Single Store	Manages all inventory within their assigned store, from receiving to organizing.
Company Stockist	All Stores	Manages high-level inventory, including transfers between stores.
Sales	Assigned Stores	Creates bills for customers at the point of sale and analyzes sales data to track performance and identify trends.

2. Functional Documentation

2.1 User Roles and Permissions

To support a growing multi-store wholesale business, we have expanded our user roles. The system now supports a clear hierarchy, ensuring staff can only access the information they need.

Role	Scope	Key Responsibilities
Company Admin	All Stores	Has complete control. Manages stores, oversees all inventory, views company-wide analytics, and manages all user accounts.
Store Manager	Assigned Store	Manages a single store. Oversees daily operations, manages staff, views store-level analytics, and handles local inventory.
Stockist	Assigned Store	Manages the inventory for a single store. Responsible for receiving new medicine shipments, organizing them, and tracking their location.
Company Stockist	All Stores	A central role for inventory oversight. Can view inventory across all stores and authorize/initiate stock transfers between them.
Sales	Assigned Store(s)	Creates bills for customers at the point of sale, handles customer interactions, and analyzes sales data to track performance and identify trends.

2.2 System Modules Overview

RetailPulse is structured into five main modules, designed to support a multi-store medical wholesale operation.

2.3 Module 1: Bill Generation

2.3.1 Purpose

To enable fast and accurate billing for customers at the store level, with all data automatically synced, primarily handled by the Sales role.

2.3.2 Features

1.1 Medicine Search and Selection

- **Store-Specific Search**: Find medicines available at the user's current store.
- **Detailed View**: See stock availability for different batches (e.g., with different expiry dates) before adding to a bill.
- **Multi-Product**: Add multiple medicines to a single bill.

1.2 Real-time Stock Validation

- **Batch-Level Check**: When a medicine is added to a bill, the system checks the quantity available in a specific batch.
- **Quantity Lock**: Ensures the amount requested does not exceed what's available in the selected batch.

1.3 Bill Creation Form

- **Customer Info**: Add customer name (e.g., pharmacy name) and contact details.
- **Product List**: Shows medicine name, batch number, expiry date, unit price, quantity, and total price.

- **Calculations**: Automatically calculates subtotal, discount, tax, and the grand total.

1.4 Bill Numbering and Tracking

- **Smart Numbering**: Bills get a unique number that includes the store ID (e.g., `STORE1-BILL-00001`).
- **Audit Trail**: Records the date, time, and the Sales user who created the bill.

1.5 PDF Generation

- **Customized PDFs**: Bills are generated in a professional PDF format with the specific store's logo and address.
- **Instant Download**: Download the PDF right after creating the bill.

1.6 Automatic Inventory Update

- **Deduct from Batch**: When a bill is finalized, the stock quantity is automatically deducted from the correct batch in the inventory.
- **Sales Log**: Every sale is logged for analytics.

2.4 Module 2: Inventory Management

2.4.1 Purpose

To provide powerful, granular control over pharmaceutical inventory across all stores, from receiving shipments to tracking their final sale.

2.4.2 Features

2.1 Medicine Master List

- **Central Catalog**: A central list of all medicines the company sells, managed by Company Admins.
- **Basic Info**: Includes medicine name, category, manufacturer, and description.

2.2 Detailed Inventory Tracking

- **Batch & Expiry**: Each medicine delivery is stored as a unique batch with its own batch number, manufacturing ID, expiry date, cost price, and selling price.
- **Location Management**: Track exactly where each batch is located within a store (e.g., `Shelf A1`, `Storeroom Rack 3`).
- **Storage Categories**: Assign a storage type to each item, such as `Cold Storage` or `General`, to ensure proper handling of sensitive medicines.

2.3 View Inventory

- **Company-Wide View**: Company Admins and Company Stockists can see inventory levels across all stores.
- **Store-Level View**: Store Managers and Stockists can see a detailed view of the inventory in their own store.
- **Advanced Filtering**: Filter inventory by medicine, category, expiry date (e.g., "expiring in 30 days"), or location.

2.4 Stock Adjustments and Movement

- **Manual Adjustments**: Manually change stock quantity with a mandatory reason (e.g., "Damaged stock," "Cycle count correction").
- **Internal Movement**: Log the movement of stock from one location to another within the same store (e.g., from the storeroom to a shelf).

- **Inter-Store Transfers**: A formal process for Company Stockists to initiate and track the transfer of stock from one store to another.

2.5 Module 3: Analytics Dashboard

2.5.1 Purpose

To provide actionable insights for different roles, from store-level performance to company-wide trends.

2.5.2 Features

3.1 Multi-Store Sales Analytics

- **Company View**: Admins can see total revenue, profit, and sales trends across all stores. They can also compare the performance of different stores.
- **Store View**: Store Managers can see detailed sales analytics for their own store.

3.2 Advanced Product Performance

- **Best Sellers**: Identify top-selling medicines by store, region, or across the entire company.
- **Batch Profitability**: Analyze the profitability of different batches of the same medicine, which may have been purchased at different prices.

3.3 Inventory Insights

- **Aging Inventory**: Generate reports to identify stock that is nearing its expiry date.
- **Stock Valuation**: View the total value of inventory, broken down by store and product category.
- **Movement History**: See a log of how stock has moved, helping to identify bottlenecks or optimize layout.

3.4 Time Range Filters

- **Flexible Filtering**: Filter all reports by standard (today, this week) or custom date ranges.

2.6 Module 4: Authentication and Authorization

2.6.1 Purpose

To secure the system and ensure users only access what their role permits.

2.6.2 Features

4.1 User Registration and Management

- **Admin Control**: Only Company Admins can create or deactivate stores and other users.
- **Store Assignment**: When creating a user, an Admin assigns them a role and, if applicable, a home store.

4.2 User Login

- **Secure Login**: Users log in with a username and password.
- **Session Management**: The system manages user sessions and provides automatic logouts for security.

4.3 Role-Based Access Control (RBAC)

- **Granular Permissions**: The system enforces the permissions defined in the **User Roles** table. For example, a `Sales` user from Store A cannot create a bill for Store B, nor can they see Store B's inventory. A `Company Admin` can do both.

2.7 Module 5: Role-Based Dashboards

2.7.1 Purpose

To provide each user with an immediate, relevant overview of their tasks and key metrics as soon as they log in. Each dashboard is tailored to the user's specific role.

2.7.2 Features

5.1 Company Admin Dashboard

- **View**: Company-wide ("bird's-eye") view.
- **Key Metrics**: Total revenue, total profit, and total sales across all stores.
- **Visuals**: A map showing all store locations, charts comparing store performance.
- **Alerts**: Notifications for system-wide issues, low-performing stores, or large-scale stock shortages.
- **Quick Links**: Manage Stores, Manage Users, Company-Wide Analytics.

5.2 Store Manager Dashboard

- **View**: Focused on their single, assigned store.
- **Key Metrics**: Total revenue, profit, and sales for their store.
- **Visuals**: Charts showing daily/weekly sales trends, top-selling medicines, and inventory value for their store.
- **Alerts**: Notifications for stock expiring soon, low stock levels, and pending inter-store transfers for their store.
- **Quick Links**: Create Bill, View Store Inventory, Store Analytics, Manage Staff (Sales/Stockists).

5.3 Stockist Dashboard

- **View**: Focused on inventory management for their assigned store.
- **Key Metrics**: Total inventory value, number of unique items, and stock-out percentage for their store.
- **Visuals**: A list of pending tasks.
- **Alerts**: Notifications for newly received shipments, pending stock counts, and items with low stock.
- **Quick Links**: View Store Inventory, Receive Stock, Move Stock, View Transfers.

5.4 Company Stockist Dashboard

- **View**: Company-wide inventory overview.
- **Key Metrics**: Total inventory value across all stores, number of pending inter-store transfers.
- **Visuals**: A table showing stock levels of key medicines across all stores for easy comparison.
- **Alerts**: Notifications for new transfer requests and completed transfers.
- **Quick Links**: View All Inventory, Initiate Transfer, View Transfer History.

5.5 Sales Dashboard

- **View**: A comprehensive interface for sales activities and performance analysis.
- **Key Metrics**: Sales revenue vs. target, number of bills created, average bill value, and top-selling medicines.

- **Visuals**: Leaderboards showing top-performing sales users, sales trends, and customer purchase patterns.
- **Alerts**: Notifications for significant sales events, customer inquiries, or when a sales target is met.
- **Quick Links**: Create Bill, View Own Bill History, View Sales Reports, Product Performance Analytics.

3. Use Cases

3.1 Primary Use Cases

3.1.1 UC-01: Sales Creates a Bill

- **Actor**: Sales
- **Goal**: To create a bill for a customer (e.g., a pharmacy) at their assigned store.
- **Flow**:
  a. Sales logs in and is on the "Create Bill" page for their store.
  b. Searches for a medicine. The system shows available batches with expiry dates and quantities.
  c. Selects a batch and adds it to the bill.
  d. Enters the quantity.
  e. Repeats for all medicines.
  f. Enters customer name and contact (optional).
  g. Clicks "Generate Bill".
  h. The system saves the bill, deducts the quantity from the specific inventory batch, and generates a PDF with the store's details.
- **Alternative**: If the requested quantity is more than the stock in the selected batch, the system shows an error.

3.1.2 UC-02: Stockist Adds New Inventory

- **Actor**: Stockist
- **Goal**: To add a new shipment of medicines into the store's inventory.
- **Flow**:
  a. Stockist goes to "Inventory Management" and clicks "Receive Stock".
  b. Selects the medicine from the master list.
  c. Enters the new stock details: batch number, manufacturing ID, expiry date, quantity received, cost price, and selling price.
  d. Assigns a location (e.g., "Storeroom Rack 5") and storage category (e.g., "Cold Storage").
  e. Clicks "Save".
  f. The new batch is now part of the store's inventory and is available for sale.
- **Alternative**: If the medicine is not on the master list, the Stockist must ask a Company Admin to add it first.

3.1.3 UC-03: Company Admin Views Company-Wide Analytics

- **Actor**: Company Admin
- **Goal**: To understand and compare the performance of all stores.
- **Flow**:
  a. Admin goes to the "Analytics Dashboard".
  b. The dashboard defaults to a company-wide view, showing total revenue, profit, and top-selling medicines across all stores.
  c. Admin uses a filter to select "Store A" and "Store B" to see a side-by-side comparison of their sales.
  d. Admin then navigates to the "Inventory" analytics tab to see the total stock value per store.

3.1.4 UC-04: Stockist Moves Stock Internally

- **Actor**: Stockist
- **Goal**: To move a medicine from the storeroom to a picking shelf.
- **Flow**:
    a. Stockist finds the inventory item (e.g., Paracetamol, Batch #123) in the system.
    b. Clicks "Move Stock".
    c. Enters the quantity to move (e.g., 20 units).
    d. Sets the "From Location" to "Storeroom Rack 5" and "To Location" to "Picking Shelf C1".
    e. Adds a reason: "Restocking picking shelf".
    f. Clicks "Confirm Movement".
    g. The system updates the location of the 20 units and logs the movement for auditing.

3.1.5 UC-05: Company Stockist Transfers Stock Between Stores

- **Actor**: Company Stockist
- **Goal**: To move surplus stock from one store to another that needs it.
- **Flow**:
    a. Company Stockist identifies that "Store A" has 200 units of a medicine, while "Store B" has only 5.
    b. They initiate a "New Store Transfer".
    c. They select the medicine, the "From Store" (Store A), and the "To Store" (Store B).
    d. They enter the quantity to transfer (e.g., 100 units).
    e. The system creates a transfer order. The 100 units at Store A are marked as "In Transit".
    f. When the stock arrives at Store B, the Stockist there confirms receipt.
    g. The system deducts the stock from Store A's inventory and adds it to Store B's inventory.

3.1.6 UC-06: Company Admin Manages Users and Stores

- **Actor**: Company Admin
- **Goal**: To manage the company's structure, including stores and staff.
- **Flow (Add Store)**:
    a. Admin goes to "Store Management" and clicks "Add New Store".
    b. Enters the store name, address, and contact info. Clicks "Save".
- **Flow (Add User)**:
    a. Admin goes to "User Management" and clicks "Add New User".
    b. Enters the user's name, username, and password.
    c. Assigns a role (e.g., "Store Manager") and a home store ("Store B").
    d. Clicks "Create User". The user can now log in and will only have access to Store B.

3.1.7 UC-07: Store Manager Checks for Expiring Medicines

- **Actor**: Store Manager
- **Goal**: To identify medicines that are nearing their expiry date to prioritize their sale.

- **Flow**:
  a. Manager goes to the "Inventory Reports" section for their store.
  b. Runs the "Expiring Soon" report, with a filter for "Next 60 days".
  c. The system generates a list of all medicine batches that will expire in the next 60 days, showing the medicine name, batch number, quantity, and expiry date.
  d. The manager can use this list to create a sales promotion or instruct staff to sell these batches first.

4. Technical Docs

4.1 Tech Stack

Frontend

Technology	Purpose
React + Vite	UI framework
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
EC2	App Hosting
GitHub Actions	CI/CD (testing & deployment)
AWS	Cloud hosting

4.2 System Data Flow

This document shows detailed data flow diagrams for each user role in the RetailPulse system.

4.2.1 1. Company Admin Data Flow

The Company Admin manages the entire system, including stores, users, and viewing all analytics.

```
%%{init: {'theme':'base', 'themeVariables': { 'fontSize':'18px', 'fontFamily':'Arial'}}}%%
flowchart TD
    Admin[Company Admin] -->|Store details| P1((Manage Stores))
    P1 --> Mgmt[Store & User Management]
    Mgmt -->|Read/Write| DB[(Database)]

    Admin -->|User details| P2((Manage Users))
    P2 --> Mgmt

    Admin -->|Request reports| P3((View Analytics))
    P3 --> Anal[Analytics Module]
    Anal -->|Read| DB
    Anal -->|Reports| Admin
```

Key Functions: - Manage all stores (create, edit, delete) - Manage all users across stores - View company-wide analytics and reports

4.2.2 2. Store Manager Data Flow

The Store Manager oversees a specific store, manages store staff, and views store-level analytics.

```
%%{init: {'theme':'base', 'themeVariables': { 'fontSize':'18px', 'fontFamily':'Arial'}}}%%
flowchart TD
    Manager[Store Manager] -->|Staff details| P1((Manage Staff))
    P1 --> Mgmt[User Management]
    Mgmt -->|Write| DB[(Database)]

    Manager -->|Request inventory| P2((View Inventory))
    P2 --> Inv[Inventory Module]
    Inv -->|Read| DB
    Inv -->|Stock data| Manager

    Manager -->|Request reports| P3((View Reports))
    P3 --> Anal[Analytics Module]
    Anal -->|Read| DB
    Anal -->|Store analytics| Manager
```

Key Functions: - Manage store staff (create Sales, Stockist users) - View store inventory levels - View store-specific sales analytics

4.2.3 3. Sales Data Flow

The Sales user creates bills, searches for products, and handles customer transactions.

```
%%{init: {'theme':'base', 'themeVariables': { 'fontSize':'18px', 'fontFamily':'Arial'}}}%%
flowchart TD
    Sales[Sales] -->|Search query| P1((Search Product))
    P1 --> Inv[Inventory Module]
    Inv -->|Read| DB[(Database)]
    Inv -->|Product list| Sales

    Sales -->|Bill items| P2((Create Bill))
    P2 --> Bill[Billing Module]
    Bill -->|Save bill| DB
    Bill -->|Deduct stock| Inv
    Inv -->|Update stock| DB
    Bill -->|Bill & PDF| Sales
```

Key Functions: - Search for products/medicines - Create customer bills and generate PDF - Automatic inventory deduction

4.2.4 4. Stockist Data Flow

The Stockist manages local store inventory, receives stock, and updates quantities.

```
%%{init: {'theme':'base', 'themeVariables': { 'fontSize':'18px', 'fontFamily':'Arial'}}}%%
flowchart TD
    Stockist[Stockist] -->|Request| P1((View Inventory))
    P1 --> Inv[Inventory Module]
    Inv -->|Read| DB[(Database)]
    Inv -->|Stock with batches| Stockist

    Stockist -->|Stock & batch details| P2((Receive Stock))
    P2 --> Inv
    Inv -->|Add stock| DB
    Inv -->|Confirmation| Stockist
```

Key Functions: - View store inventory with batch details - Receive and add new stock with expiry dates

4.2.5 5. Company Stockist Data Flow

The Company Stockist oversees inventory across all stores and manages inter-store transfers.

```
%%{init: {'theme':'base', 'themeVariables': { 'fontSize':'18px', 'fontFamily':'Arial'}}}%%
flowchart TD
    CompStockist[Company Stockist] -->|Request| P1((View All Inventory))
    P1 --> Inv[Inventory Module]
    Inv -->|Read all stores| DB[(Database)]
    Inv -->|Consolidated data| CompStockist

    CompStockist -->|Transfer details| P2((Transfer Stock))
    P2 --> Inv
    Inv -->|Update stores| DB
    Inv -->|Confirmation| CompStockist

    CompStockist -->|Request| P3((Check Expiry))
    P3 --> Inv
    Inv -->|Read| DB
    Inv -->|Expiry alerts| CompStockist
```

Key Functions: - View inventory across all stores - Manage inter-store stock transfers - Monitor expiry dates company-wide

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 system identifies their role and redirects them to the appropriate dashboard. 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 --> E[Redirect to Role-Specific Dashboard];
```

4.3.2 Authorization Rules

The system uses a granular Role-Based Access Control (RBAC) model to enforce permissions. A user's role determines what they can see and do, and their assigned store (if any) limits the scope of their actions.

Below is a summary of the permissions for each role.

Action	Company Admin	Store Manager	Stockist	Company Stockist	Sales
Store Management					
Create/Edit Stores	Yes	No	No	No	No
User Management					
Create/Edit Users	Yes	No	No	No	No
Medicine Master List					
Create/Edit Medicines	Yes	No	No	No	No
Inventory					
View Own Store Inventory	Yes	Yes	Yes	Yes	Yes
View All Stores' Inventory	Yes	No	Yes	Yes	Yes
Add/Edit Own Store Inventory	Yes	Yes	Yes	No	No
Move Stock (within store)	Yes	Yes	Yes	No	No
Transfer Stock (between stores)	Yes	No	No	Yes	No
Billing					
Create/Edit Bills (own store)	Yes	Yes	No	No	Yes
View Bills (own store)	Yes	Yes	No	No	Yes
View Bills (all stores)	Yes	No	No	No	No
Analytics					
View Own Store Analytics	Yes	Yes	No	No	Yes
View All Stores' Analytics	Yes	No	No	No	No

4.4 Database Schema

4.4.1 Overview

The system uses **PostgreSQL** as the database. The schema is designed to support a multi-store medical retail business, with detailed inventory tracking and a flexible Role-Based Access Control (RBAC) system.

The RBAC system is composed of `roles`, `permissions`, and a `role_permissions` junction table. This allows for granular control over user actions, making the system scalable and easy to manage.

To ensure data integrity and history, all tables include `created_at`, `updated_at`, and `deleted_at` timestamps for soft deletes.

4.4.2 Entity Relationship Diagram

```
erDiagram
    users {
        int id PK
        int store_id FK
        int role_id FK
        varchar username UK
        varchar password_hash
        varchar full_name
        boolean is_active
        timestamptz created_at
        timestamptz updated_at
        timestamptz deleted_at
    }

    roles {
        int id PK
        varchar name UK
        text description
    }

    permissions {
        int id PK
        varchar name UK
        text description
    }

    role_permissions {
        int role_id PK, FK
        int permission_id PK, FK
    }

    stores {
        int id PK
        varchar name UK
        text address
        varchar contact_phone
        timestamptz created_at
        timestamptz updated_at
        timestamptz deleted_at
    }

    medicines {
        int id PK
        varchar name UK
        varchar category
        varchar manufacturer
        text description
        boolean is_active
        timestamptz created_at
        timestamptz updated_at
        timestamptz deleted_at
    }

    inventory_items {
        int id PK
        int medicine_id FK
        int store_id FK
        varchar batch_number
        varchar manufacturing_id
        date expire_date
        int quantity
        varchar location
        varchar storage_category
        decimal cost_price
        decimal selling_price
        timestamptz created_at
        timestamptz updated_at
        timestamptz deleted_at
    }

    inventory_movements {
        int id PK
        int inventory_item_id FK
        int moved_by_user_id FK
        int quantity_moved
        varchar from_location
        varchar to_location
        text reason
        timestamptz movement_date
    }

    bills {
        int id PK
        int store_id FK
        int created_by_user_id FK
        varchar bill_number UK
        varchar customer_name
        varchar customer_contact
        decimal total_amount
        decimal discount
        decimal tax_amount
        decimal grand_total
        timestamptz bill_date
        timestamptz created_at
        timestamptz updated_at
        timestamptz deleted_at
    }

    bill_items {
        int id PK
        int bill_id FK
        int medicine_id FK
        varchar batch_number
        int quantity
        decimal unit_price
        decimal total_price
        timestamptz created_at
        timestamptz updated_at
        timestamptz deleted_at
    }

    users ||--|{ roles : "has"
    roles ||--|{ role_permissions : "has many"
    permissions ||--o{ role_permissions : "has many"
    stores ||--|{ users : "employs"
    stores ||--|{ bills : "generates"
    stores ||--|{ inventory_items : "has"
    users ||--o{ bills : "creates"
    users ||--o{ inventory_movements : "performs"
    medicines ||--o{ inventory_items : "is an instance of"
    bills ||--|{ bill_items : "contains"
    medicines ||--o{ bill_items : "appears in"
    inventory_items ||--o{ inventory_movements : "is moved in"
```

4.4.3 Database Tables

1. `stores`

Stores information for each retail store.

Fields:

- `id` (SERIAL PRIMARY KEY): Unique ID for the store.
- `name` (VARCHAR(255) UNIQUE NOT NULL): The name of the store (e.g., "Main Street Pharmacy").
- `address` (TEXT): Physical address of the store.
- `contact_phone` (VARCHAR(50)): Contact phone number for the store.
- `created_at` (TIMESTAMPTZ NOT NULL DEFAULT now()): Timestamp when the record was created.
- `updated_at` (TIMESTAMPTZ NOT NULL DEFAULT now()): Timestamp when the record was last updated.
- `deleted_at` (TIMESTAMPTZ): Timestamp for soft deletes.

2. `users`

Stores user login information and their assigned role.

Fields:

- `id` (SERIAL PRIMARY KEY): Unique user ID.
- `store_id` (INTEGER REFERENCES stores(id)): The store this user belongs to. Can be `NULL` for company-wide users.
- `role_id` (INTEGER NOT NULL REFERENCES roles(id)): The role assigned to this user.
- `username` (VARCHAR(100) UNIQUE NOT NULL): Login username.
- `password_hash` (VARCHAR(255) NOT NULL): Encrypted password.
- `full_name` (VARCHAR(255)): User's full name.
- `is_active` (BOOLEAN NOT NULL DEFAULT true): Whether the user can log in.
- `created_at` (TIMESTAMPTZ NOT NULL DEFAULT now()): Timestamp when the record was created.
- `updated_at` (TIMESTAMPTZ NOT NULL DEFAULT now()): Timestamp when the record was last updated.
- `deleted_at` (TIMESTAMPTZ): Timestamp for soft deletes.

3. `roles`

Defines the user roles available in the system.

Fields:

- `id` (SERIAL PRIMARY KEY): Unique ID for the role.
- `name` (VARCHAR(50) UNIQUE NOT NULL): The name of the role (e.g., 'Company Admin', 'Store Manager', 'Sales').
- `description` (TEXT): A brief description of the role.

4. `permissions`

Defines granular permissions for actions within the system.

Fields:

- `id` (SERIAL PRIMARY KEY): Unique ID for the permission.
- `name` (VARCHAR(100) UNIQUE NOT NULL): The name of the permission (e.g., 'bills.create', 'analytics.view.all').
- `description` (TEXT): A brief description of what the permission allows.

5. `role_permissions`

A junction table that assigns permissions to roles, creating a many-to-many relationship.

Fields:

- `role_id` (INTEGER NOT NULL REFERENCES roles(id)): The role being assigned a permission.
- `permission_id` (INTEGER NOT NULL REFERENCES permissions(id)): The permission being granted.
- PRIMARY KEY (`role_id`, `permission_id`): Ensures each permission is assigned to a role only once.

6. `medicines`

Stores general information about each medicine. This is the master catalog of all pharmaceuticals the company sells.

Fields:

- `id` (SERIAL PRIMARY KEY): Unique medicine ID.
- `name` (VARCHAR(255) UNIQUE NOT NULL): Medicine name.
- `category` (VARCHAR(100)): Medicine category (e.g., "Antibiotic", "Analgesic").
- `manufacturer` (VARCHAR(255)): Brand or manufacturer name.
- `description` (TEXT): Medicine details.
- `is_active` (BOOLEAN NOT NULL DEFAULT true): Whether the medicine is available for sale.
- `created_at` (TIMESTAMPTZ NOT NULL DEFAULT now()): Timestamp when the record was created.
- `updated_at` (TIMESTAMPTZ NOT NULL DEFAULT now()): Timestamp when the record was last updated.
- `deleted_at` (TIMESTAMPTZ): Timestamp for soft deletes.

7. `inventory_items`

This is the core inventory table. It tracks specific batches of medicines in specific stores and locations.

Fields:

- `id` (SERIAL PRIMARY KEY): Unique ID for this inventory item.
- `medicine_id` (INTEGER NOT NULL REFERENCES medicines(id)): The medicine this item refers to.
- `store_id` (INTEGER NOT NULL REFERENCES stores(id)): The store where this item is located.
- `batch_number` (VARCHAR(100)): The batch number for this group of items.
- `manufacturing_id` (VARCHAR(100)): The manufacturing ID, if available.
- `expire_date` (DATE): The expiration date of this batch.
- `quantity` (INTEGER NOT NULL CHECK (quantity >= 0)): The number of units in this batch at this location.
- `location` (VARCHAR(100)): The specific location in the store (e.g., "Shelf A1", "Cold Storage Room").
- `storage_category` (VARCHAR(100)): The type of storage required (e.g., "Cold Storage", "General").
- `cost_price` (DECIMAL(10, 2) NOT NULL): The purchase price per unit for this batch.
- `selling_price` (DECIMAL(10, 2) NOT NULL): The selling price per unit for this batch.
- `created_at` (TIMESTAMPTZ NOT NULL DEFAULT now()): Timestamp when the record was created.
- `updated_at` (TIMESTAMPTZ NOT NULL DEFAULT now()): Timestamp when the record was last updated.
- `deleted_at` (TIMESTAMPTZ): Timestamp for soft deletes.

8. `inventory_movements`

Logs the movement of inventory items from one location to another, creating an audit trail.

Fields:

- `id` (SERIAL PRIMARY KEY): Unique ID for the movement log.
- `inventory_item_id` (INTEGER NOT NULL REFERENCES inventory_items(id)): The inventory item that was moved.
- `moved_by_user_id` (INTEGER NOT NULL REFERENCES users(id)): The user who performed the movement.
- `quantity_moved` (INTEGER NOT NULL): The number of units moved.
- `from_location` (VARCHAR(100)): The location the items were moved from.
- `to_location` (VARCHAR(100)): The location the items were moved to.
- `reason` (TEXT): The reason for the movement (e.g., "Restocking shelf", "Store transfer").
- `movement_date` (TIMESTAMPTZ NOT NULL DEFAULT now()): When the movement occurred.

9. `bills`

Stores header information for each bill (receipt).

Fields:

- `id` (SERIAL PRIMARY KEY): Unique bill ID.
- `store_id` (INTEGER NOT NULL REFERENCES stores(id)): The store that issued the bill.
- `created_by_user_id` (INTEGER NOT NULL REFERENCES users(id)): The user who created the bill.
- `bill_number` (VARCHAR(100) UNIQUE NOT NULL): Unique bill number (e.g., "STORE1-BILL-00001").
- `customer_name` (VARCHAR(255)): Customer's name (e.g., "City Pharmacy").
- `customer_contact` (VARCHAR(100)): Customer's phone/email (optional).
- `total_amount` (DECIMAL(10, 2) NOT NULL): Sum of all items before discounts/taxes.
- `discount` (DECIMAL(10, 2) DEFAULT 0): Discount amount.
- `tax_amount` (DECIMAL(10, 2) DEFAULT 0): Tax/GST amount.
- `grand_total` (DECIMAL(10, 2) NOT NULL): Final amount to be paid.
- `bill_date` (TIMESTAMPTZ NOT NULL DEFAULT now()): The date and time the bill was created.
- `created_at` (TIMESTAMPTZ NOT NULL DEFAULT now()): Timestamp when the record was created.
- `updated_at` (TIMESTAMPTZ NOT NULL DEFAULT now()): Timestamp when the record was last updated.
- `deleted_at` (TIMESTAMPTZ): Timestamp for soft deletes.

10. `bill_items`

Stores the individual line items for each bill.

Fields:

- `id` (SERIAL PRIMARY KEY): Unique ID for the bill line item.
- `bill_id` (INTEGER NOT NULL REFERENCES bills(id)): The bill this item belongs to.
- `medicine_id` (INTEGER NOT NULL REFERENCES medicines(id)): The medicine that was sold.
- `batch_number` (VARCHAR(100)): The specific batch number of the item sold, for accurate tracking.
- `quantity` (INTEGER NOT NULL): Number of units sold.
- `unit_price` (DECIMAL(10, 2) NOT NULL): Price per unit at the time of sale.
- `total_price` (DECIMAL(10, 2) NOT NULL): `quantity` × `unit_price`.
- `created_at` (TIMESTAMPTZ NOT NULL DEFAULT now()): Timestamp when the record was created.
- `updated_at` (TIMESTAMPTZ NOT NULL DEFAULT now()): Timestamp when the record was last updated.
- `deleted_at` (TIMESTAMPTZ): Timestamp for soft deletes.

4.5 Architecture

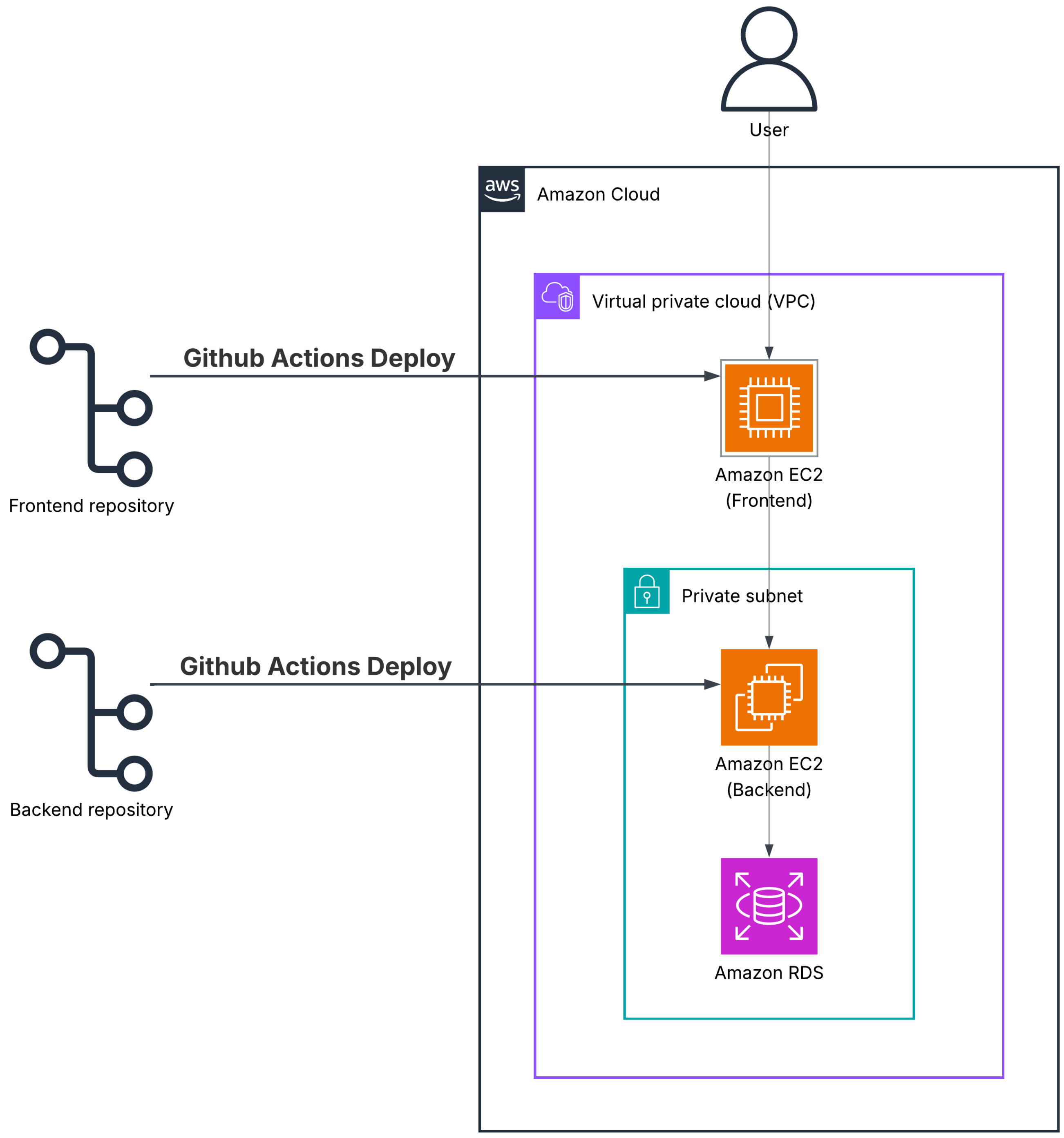
4.5.1 System Architecture Overview

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 AWS-Specific Architecture Diagram

This diagram shows a simplified view of the system hosted on AWS.


User
Amazon Cloud
Virtual private cloud (VPC)
Github Actions Deploy
Frontend repository
Amazon EC2
(Frontend)
Private subnet
Github Actions Deploy
Backend repository
Amazon EC2
(Backend)
Amazon RDS

4.6 API Design

The API is designed following RESTful principles to provide a logical, hierarchical structure for managing a multi-store medical wholesale business. Resources are nested where it makes sense (e.g., a bill belongs to a store).

Authentication Endpoints

No change here. Authentication is the entry point.

Method	Endpoint	Description
`POST`	`/auth/login`	User login to get a JWT access token.
`POST`	`/auth/logout`	User logout (invalidates token).

Store Endpoints

For Company Admins to manage the store network.

Method	Endpoint	Description
`POST`	`/stores`	Create a new store.
`GET`	`/stores`	List all stores in the company.
`GET`	`/stores/{store_id}`	Get details for a specific store.
`PATCH`	`/stores/{store_id}`	Update a store's details.

User Management Endpoints

For Company Admins to manage users.

Method	Endpoint	Description
`POST`	`/users`	Create a new user (can assign a role and store).
`GET`	`/users`	List all users (can filter by store, role).
`GET`	`/users/{user_id}`	Get details for a specific user.
`PATCH`	`/users/{user_id}`	Update a user's details (e.g., role, store).
`DELETE`	`/users/{user_id}`	Deactivate a user account.

Medicine Master Endpoints

For managing the company-wide medicine catalog.

Method	Endpoint	Description
`POST`	`/medicines`	Create a new medicine in the master catalog.
`GET`	`/medicines`	List all medicines in the master catalog.
`GET`	`/medicines/{medicine_id}`	Get details for a specific master medicine.
`PATCH`	`/medicines/{medicine_id}`	Update a master medicine's details.

Inventory Endpoints

The core of inventory management. Granular control over stock.

Method	Endpoint	Description
`POST`	`/stores/{store_id}/inventory-items`	Add a new inventory item (a new batch) to a store.
`GET`	`/stores/{store_id}/inventory-items`	Get all inventory items for a specific store.
`GET`	`/inventory-items/{item_id}`	Get details of a specific inventory item (batch).
`PATCH`	`/inventory-items/{item_id}`	Update an inventory item (e.g., adjust quantity, change price).
`POST`	`/inventory-items/movements`	Log an internal movement of stock within a store.
`POST`	`/inventory-transfers`	Initiate a transfer of stock between two stores.
`PATCH`	`/inventory-transfers/{transfer_id}`	Update the status of a transfer (e.g., confirm receipt).

Bill Endpoints

Store-specific endpoints for creating and viewing bills.

Method	Endpoint	Description
`POST`	`/stores/{store_id}/bills`	Create a new bill for a specific store.
`GET`	`/stores/{store_id}/bills`	List all bills for a specific store.
`GET`	`/bills/{bill_id}`	Get details for a specific bill (accessible across stores by Admins).

Analytics Endpoints

Flexible endpoints for company-wide or store-specific insights.

Method	Endpoint	Description
`GET`	`/analytics/sales`	Get sales report data. Can be filtered by `store_id`. If no filter, returns company-wide data.
`GET`	`/analytics/inventory`	Get inventory reports (e.g., stock valuation, expiring soon). Can be filtered by `store_id`.
`GET`	`/analytics/performance`	Get performance metrics (e.g., top medicines). Can be filtered by `store_id`.

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 `Store Manager` tries to access a `Company Admin`-only page.
  - **Solution**: The system shows an "Insufficient permissions" message and denies access.

5.3 2. Bill Creation Problems

- **Problem**: User tries to create a bill with no medicines.
  - **Solution**: The system requires at least one medicine to be in the bill.
- **Problem**: A medicine is not found in the store's inventory or is inactive.
  - **Solution**: The system shows a "Medicine not found or inactive" message.
- **Problem**: The quantity requested is more than the available stock in the selected batch.
  - **Solution**: The system shows an error message like "Only 5 units available in this batch".
- **Problem**: The quantity is zero or a negative number.
  - **Solution**: The system requires the quantity to be greater than zero.
- **Problem**: The database connection is lost while creating a bill.
  - **Solution**: The entire transaction is cancelled (rolled back) to prevent partial data. The user is asked to try again.

5.4 3. Inventory Management Problems

- **Problem**: `Company Admin` tries to create a medicine with a name that already exists.
  - **Solution**: The system shows a "Medicine with this name already exists" message.
- **Problem**: `Stockist` tries to adjust stock to a negative quantity.
  - **Solution**: The system shows a "Stock quantity cannot be negative" message. The database has a `CHECK` constraint to enforce this.
- **Problem**: A user adjusts stock without giving a reason.
  - **Solution**: The system requires a reason for all manual stock adjustments to maintain a clear audit trail.

5.5 4. Analytics Problems

- **Problem**: There is no sales data for the selected date range in a specific store.
  - **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 filter.
  - **Solution**: The system shows a "Start date must be before end date" message.

5.6 5. User Management Problems

- **Problem**: `Company Admin` tries to create a user with a username that already exists.
  - **Solution**: The system shows a "Username already exists" message.
- **Problem**: `Company Admin` tries to deactivate their own account.
  - **Solution**: The system prevents this to ensure there's always at least one active super admin.

5.7 6. Multi-Store and Advanced Inventory Edge Cases

- **Problem**: A `Sales` user tries to sell a medicine from a batch that has expired.
  - **Solution**: The system should not show expired batches in the search results on the billing page. If an API call is made directly, the backend should reject the request with a "Cannot sell from expired batch" error.
- **Problem**: Two `Sales` users try to sell the last item of a batch at the same time (a race condition).
  - **Solution**: The system uses database-level transaction isolation. The first transaction to commit will succeed. The second transaction will fail when it attempts to update the quantity, as the stock will already be zero. It will receive an error like "Insufficient stock".
- **Problem**: A stock transfer is sent from Store A, but Store B claims it was never received or the quantity is wrong.
  - **Solution**: The transfer remains in an "In Transit" state until the receiving store confirms it. This creates a clear paper trail. If there's a dispute, a `Company Admin` can manually intervene to reconcile the inventory, and the discrepancy is logged.
- **Problem**: A branch store loses internet connection to the central server.
  - **Solution**: In the current design, the system requires a live connection to the central database. If the connection is lost, the store's terminal will not be able to create new bills or look up inventory. (A future enhancement could be a limited "offline mode" that syncs data once the connection is restored).
- **Problem**: A `Company Admin` reassigns a `Store Manager` from Store A to Store B.
  - **Solution**: The user's `store_id` is updated in the `users` table. The change should take effect on their next login. When they log in again, their session data will be for Store B, and they will no longer have access to Store A's data or settings.

6. Future Functional Enhancements

1. Quick product search and billing using barcode scanner
2. Send bill PDFs directly to customer's WhatsApp
3. Store customer contact details and view their purchase history
4. Track credit sales and outstanding payments from customers
5. Maintain supplier details and track purchase orders sent to suppliers
6. Automatic email/SMS notifications when products fall below the minimum stock level
7. Send bills and payment reminders to customers via email
8. Advanced analytics, such as customer segmentation and purchase patterns
9. Role-based dashboard customization, allowing users to personalize their workspace