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 product handling standards effectively.

1.1.2 2. Complex Inventory Management

Tracking product 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. A sales user manually checks for the product. 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 Product 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 sales user creates a digital bill in seconds. The system automatically deducts the product 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, and manages all user accounts.
Store Manager	Assigned Store	Manages a single store. Oversees daily operations, manages staff, and handles local inventory.
Stockist	Assigned Store	Manages the inventory for a single store. Responsible for receiving new product 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 and handles customer interactions.

2.2 System Modules Overview

RetailPulse is structured into Three 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 Product Search and Selection

- **Store-Specific Search**: Find products 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 products to a single bill.

1.2 Real-time Stock Validation

- **Batch-Level Check**: When a product 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 product 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.

2.4 Module 2: Inventory Management

2.4.1 Purpose

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

2.4.2 Features

2.1 Product Master List

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

2.2 Detailed Inventory Tracking

- **Batch & Expiry**: Each product 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 products.

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 product, 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: Authentication and Authorization

2.5.1 Purpose

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

2.5.2 Features

3.1 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.

3.2 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.

3.3 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.

3. Use Cases

3.1 Core System Use Cases

3.1.1 UC-01: Sales User Processes a Customer Order

- **Actor**: Sales
- **Goal**: To efficiently create a bill for a customer, ensuring accurate product selection and inventory deduction.
- **Flow**:
  a. The Sales user logs into the system and navigates to the "Create Bill" interface for their assigned store.
  b. They search for a product by name or code. The system displays available batches of the product, including expiry dates and current quantities in stock.
  c. The Sales user selects the desired product batch and specifies the quantity. The system validates that the requested quantity does not exceed the available stock in that batch.
  d. They can add multiple products to the same bill, repeating the search and selection process.
  e. Optionally, the Sales user enters customer details (e.g., name, contact information).
  f. Upon confirming the order, the Sales user clicks "Generate Bill".
  g. The system records the bill, automatically deducts the sold quantity from the specific inventory batch, and generates a printable PDF of the bill, customized with the store's details.
- **Alternative**: If the requested quantity for a product exceeds the available stock in the selected batch, the system displays an error message, prompting the user to adjust the quantity or select a different batch.

3.1.2 UC-02: Stock Management and Inter-Store Transfers

- **Actors**: Stockist, Company Stockist
- **Goal**: To maintain accurate inventory levels within stores and facilitate efficient product movement between locations.
- **Flow (Stockist - Receiving Inventory)**:
  a. A Stockist logs in and accesses the "Receive Stock" function within their assigned store's inventory management module.
  b. They select a product from the master catalog and enter details for the new shipment, including batch number, manufacturing ID, expiry date, quantity received, cost price, and selling price.
  c. The Stockist assigns a specific physical `location` (e.g., "Storeroom Rack 5") and `storage_category` (e.g., "Cold Storage") for the received batch.
  d. Upon saving, the new batch is added to the store's inventory and becomes available for sale or internal movement.
- **Flow (Company Stockist - Transferring Inventory Between Stores)**:
  a. A Company Stockist identifies a need to transfer products between stores (e.g., Store A has surplus, Store B has a shortage).
  b. They initiate a "New Store Transfer," selecting the product, the "From Store" (e.g., Store A), and the "To Store" (e.g., Store B), along with the quantity to transfer.
  c. The system marks the transferred quantity as "In Transit" in the "From Store's" inventory.
  d. Once the products arrive at the "To Store," a Stockist at that store confirms receipt in the system.
  e. The system then deducts the stock from the "From Store's" inventory and adds it to the "To Store's" inventory, updating locations as necessary.

3.1.3 UC-03: Company Admin Manages System Resources

- **Actor**: Company Admin
- **Goal**: To oversee and configure the entire system, including managing stores, users, and the master product catalog.
- **Flow (Managing Stores)**:
  a. The Company Admin logs in and navigates to the "Store Management" section.
  b. They can add new stores by providing details such as name, address, and contact information.
  c. Existing store details can be updated, or stores can be deactivated as needed.
- **Flow (Managing Users)**:
  a. The Company Admin accesses the "User Management" section.
  b. They can create new user accounts, providing a username, password, and full name.
  c. Crucially, the Admin assigns one or more `roles` to the new user (e.g., 'Store Manager', 'Sales') and associates them with one or more `stores` via the `user_roles` and `user_stores` junction tables.
  d. Existing user accounts can be updated (e.g., changing roles, store assignments) or deactivated.
- **Flow (Managing Product Master Catalog)**:
  a. The Company Admin goes to the "Product Master List" section.
  b. They can add new products to the central catalog, specifying details like name, category, manufacturer, and description.
  c. Existing product details can be updated, or products can be marked as inactive if they are no longer sold.

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 and users.

```
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
```

Key Functions: - Manage all stores (create, edit, delete) - Manage all users across stores

4.2.2 2. Store Manager Data Flow

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

```
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
```

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.

```
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 - 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.

```
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.

```
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
```

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
Product Master List					
Create/Edit Products	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

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.

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 ||--|{ user_roles : "has"
    roles ||--o{ user_roles : "has many"
    users ||--|{ user_stores : "is in"
    stores ||--o{ user_stores : "has many"
    roles ||--|{ role_permissions : "has many"
    permissions ||--o{ role_permissions : "has many"
    stores ||--|{ bills : "generates"
    stores ||--|{ inventory_items : "has"
    users ||--o{ bills : "creates"
    users ||--o{ inventory_movements : "performs"
    products ||--o{ inventory_items : "is an instance of"
    bills ||--|{ bill_products : "contains"
    products ||--o{ bill_products : "appears in"
    inventory_items ||--o{ inventory_movements : "is moved in"

    users {
        int id PK
        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
        timestamptz created_at
        timestamptz updated_at
        timestamptz deleted_at
    }

    user_roles {
        int id PK
        int user_id FK
        int role_id FK
        timestamptz created_at
        timestamptz updated_at
        timestamptz deleted_at
    }

    permissions {
        int id PK
        varchar name UK
        text description
        timestamptz created_at
        timestamptz updated_at
        timestamptz deleted_at
    }

    role_permissions {
        int id PK
        int role_id FK
        int permission_id FK
        timestamptz created_at
        timestamptz updated_at
        timestamptz deleted_at
    }

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

```
user_stores {
    int id PK
    int user_id FK
    int store_id FK
    timestamptz created_at
    timestamptz updated_at
    timestamptz deleted_at
}

products {
    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 product_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
    timestamptz created_at
    timestamptz updated_at
    timestamptz deleted_at
}

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_products {
    int id PK
    int bill_id FK
    int product_id FK
    varchar batch_number
    int quantity
    decimal unit_price
    decimal total_price
    timestamptz created_at
    timestamptz updated_at
    timestamptz deleted_at
}
```

4.4.3 Database Tables

`stores`

The `stores` table contains a record for each retail store location. It includes a unique `id`, the store's `name`, `address`, and `contact_phone`. Timestamps for creation, updates, and soft deletes are included.

`users`

The `users` table is the central record for every person who can log in to the system. It contains a unique `id`, `username`, `password_hash`, `full_name`, and an `is_active` flag. A user's roles and store assignments are managed in separate junction tables, so this table does not contain `role_id` or `store_id`. It also includes `created_at`, `updated_at`, and `deleted_at` timestamps.

`roles`

The `roles` table defines the user roles available in the system. It has a unique `id` and a `name` for the role (e.g., 'Company Admin', 'Store Manager', 'Sales'), along with a `description`. It also includes `created_at`, `updated_at`, and `deleted_at` timestamps.

`user_roles`

This is a junction table that assigns roles to users, creating a many-to-many relationship. It links `user_id` to `role_id`, allowing a single user to have multiple roles (e.g., a user could be both a 'Store Manager' and 'Sales'). It also includes `created_at`, `updated_at`, and `deleted_at` timestamps.

`permissions`

The `permissions` table defines granular permissions for actions within the system. It has a unique `id` and a `name` for the permission (e.g., 'bills.create', 'analytics.view.all'), with a `description` of what it allows. It also includes `created_at`, `updated_at`, and `deleted_at` timestamps.

`role_permissions`

This is a junction table that assigns permissions to roles, creating a many-to-many relationship. It links `role_id` to `permission_id`, ensuring each permission is assigned to a role only once. It also includes `created_at`, `updated_at`, and `deleted_at` timestamps.

`user_stores`

This is a junction table that assigns users to stores, creating a many-to-many relationship. It links `user_id` to `store_id`, allowing a single user to be associated with multiple stores. It also includes `created_at`, `updated_at`, and `deleted_at` timestamps.

`products`

The `products` table is the master catalog of all products the company sells. It contains the product's `id`, `name`, `category`, `manufacturer`, `description`, and an `is_active` flag. It also includes `created_at`, `updated_at`, and `deleted_at` timestamps.

`inventory_items`

This is the core inventory table, tracking specific batches of products in specific stores and locations. It links to `products` and `stores` and includes a `batch_number`, `manufacturing_id`, `expire_date`, `quantity`, `location`, `storage_category`, `cost_price`, and `selling_price`. It also includes `created_at`, `updated_at`, and `deleted_at` timestamps.

`inventory_movements`

The `inventory_movements` table logs the movement of inventory items from one location to another, creating an audit trail. It records the `inventory_item_id`, the `moved_by_user_id`, `quantity_moved`, `from_location`, `to_location`, a `reason`, and the `movement_date`. It also includes `created_at`, `updated_at`, and `deleted_at` timestamps.

`bills`

The `bills` table stores header information for each bill (receipt). It is linked to the `stores` where it was created and the `users` who created it. It also contains a unique `bill_number`, customer details, and financial totals (`total_amount`, `discount`, `tax_amount`, `grand_total`). It also includes `created_at`, `updated_at`, and `deleted_at` timestamps.

`bill_products`

The `bill_products` table stores the individual line items for each bill. It links to the `bills` table and the specific `products` sold. It includes the `quantity`, `unit_price`, `total_price`, and the `batch_number` of the item sold for accurate inventory tracking. It also includes `created_at`, `updated_at`, and `deleted_at` timestamps.

4.5 Architecture

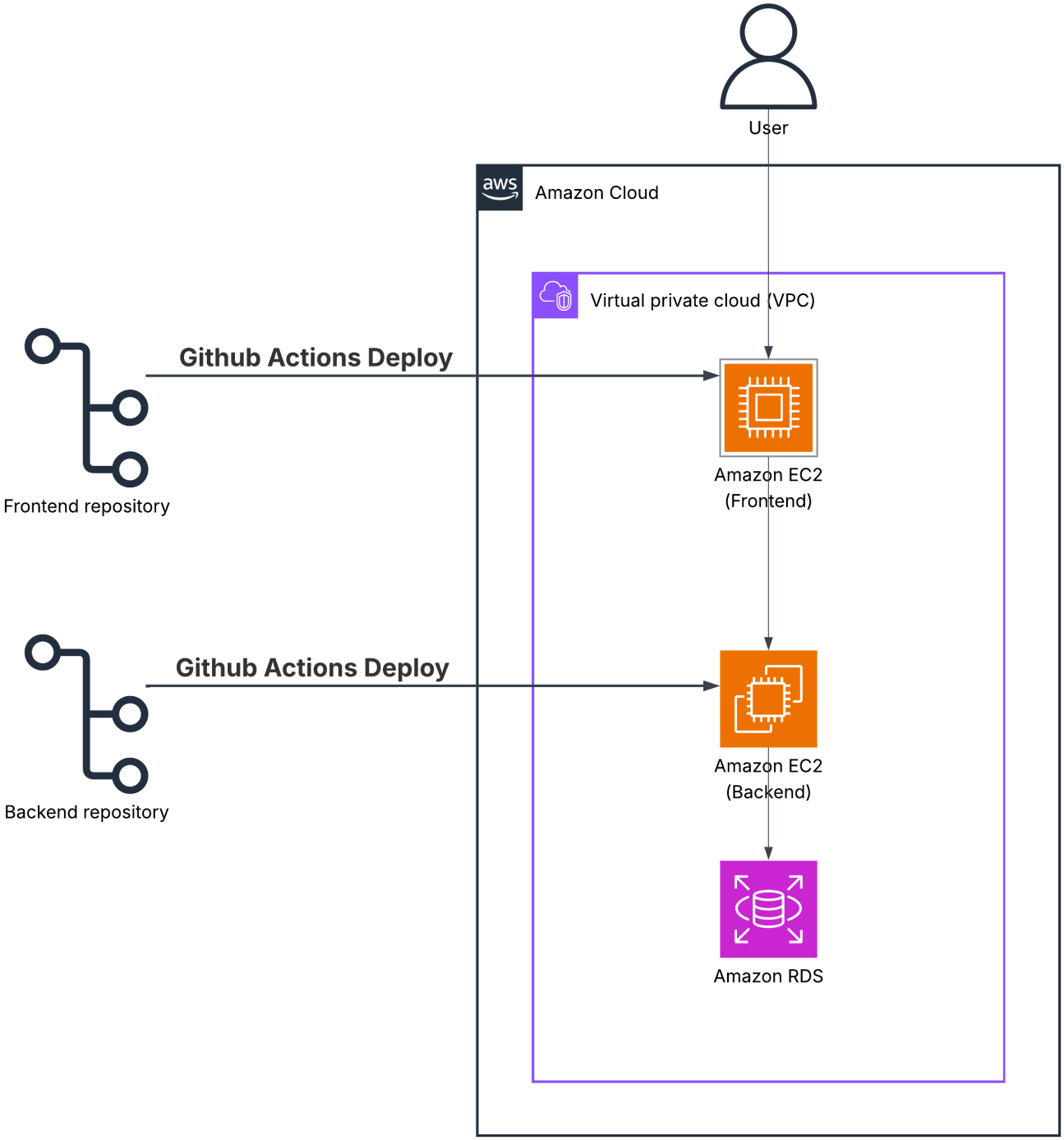
4.5.1 System Architecture Overview

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
aws
Amazon Cloud
Virtual private cloud (VPC)
Github Actions Deploy
Frontend repository
Amazon EC2
(Frontend)
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

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

Store Endpoints

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

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.

Product Master Endpoints

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

Inventory Endpoints

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

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).

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 products.
  - **Solution**: The system requires at least one product to be in the bill.
- **Problem**: A product is not found in the store's inventory or is inactive.
  - **Solution**: The system shows a "Product 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 product with a name that already exists.
  - **Solution**: The system shows a "Product 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 product 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. **Analytics Dashboard**: A comprehensive dashboard to provide actionable insights into sales, product performance, and inventory. This will include features like time-range filtering, visual charts, and reports on:
    - Sales trends (revenue, profit, best-sellers)
    - Inventory insights (stock valuation, aging inventory)
    - Advanced analytics (customer segmentation, purchase patterns)
2. Quick product search and billing using barcode scanner
3. Send bill PDFs directly to customer's WhatsApp
4. Store customer contact details and view their purchase history
5. Track credit sales and outstanding payments from customers
6. Maintain supplier details and track purchase orders sent to suppliers
7. Automatic email/SMS notifications when products fall below the minimum stock level
8. Send bills and payment reminders to customers via email
9. Role-based dashboard customization, allowing users to personalize their workspace