DBMS Mini Project Review – 1

Online Medicine Ordering System : MediQuick

Team Details:

Team 21

Name	SRN
Archita Jain	PES1UG23AM063
Bhargavi D	PES1UG23AM077

I. User Requirement Specification:

i. Purpose of the project:

The purpose of this project is to develop a fully functional **Online Medicines Ordering System** named "MediQuick," designed to modernize and streamline the process of purchasing pharmaceuticals. The core objective is to manage the complex logistics of fulfilling a single customer order from multiple inventory sources based on real-time stock availability provided by the pharmacies.

The system is engineered to handle advanced SQL operations, ensuring data integrity and business logic enforcement at the database level. It will implement complex transactions for order placement and splitting, triggers for automated alerts and checks, stored procedures for encapsulating business processes, and a variety of sophisticated queries for reporting and management. The design incorporates fundamental database modeling concepts including weak entities, identifying relationships, composite and multi-valued attributes, and all relationship types (1:1, 1:N, M:N, Ternary, Recursive).

ii. Scope of the project:

The scope of the MediQuick project encompasses the end-to-end development of a web-based application, comprising both a user-friendly front-end interface and a powerful, complex back-end database system. The system will manage key functionalities including user registration and authentication, dynamic product catalog browsing, a secure shopping cart, and a multi-step checkout process that integrates payment processing. It will support the upload and validation of digital prescriptions for restricted medicines. On the administrative side, the scope includes interfaces for pharmacy managers to monitor inventory levels and sales reports, and for delivery agents to receive and update the status of their assigned orders. The successful execution of this project hinges on the implementation of a normalized and well-structured SQL database that efficiently handles all data relationships and business logic.

Detailed description (Case study):

In today's fast-paced world, access to essential healthcare products like medicines needs to be convenient, reliable, and efficient. The MediQuick system is a proposed digital solution designed to bridge the gap between patients and pharmacies by providing a comprehensive online platform for ordering medicines. The system caters to a network of registered pharmacies, allowing them to list their inventory digitally. Customers can browse available medicines, search for substitutes if a particular brand is out of stock, and place orders for either over-the-counter or prescription-based drugs by uploading a digital prescription.

The operational workflow begins when a Customer registers, creating an entry in the Customer table. A customer then adds items to a Cart, which is stored in the Cart and Cart_item tables. When they are ready to purchase, an Order is created in the Order table, drawing from the cart's contents. The system's core logic then engages in a complex transaction: it reserves the items in the Available_stock table, calculates the total cost, and creates a payment record. For prescription orders, the system validates the uploaded prescription, which is recorded in the Prescription table and linked to the Order. Once confirmed and paid for, the order is routed to the specific Pharmacy that supplies the medicines. A ternary relationship then comes into play, formalized by the Sub_order table, which links the Order, Pharmacy, and Delivery_Agent. A delivery agent is assigned to pick up the order from that particular pharmacy and deliver it to the customer, with the status of this delivery tracked in the Sub_order table.
This system operates in an inherently transactional environment with numerous challenges: ensuring that stock is not oversold (data integrity), automatically alerting pharmacists when inventory is low via triggers on the Available_stock table, calculating complex order totals using functions, and providing insights into sales performance and delivery efficiency through advanced aggregate and join queries. The database maintains comprehensive audit trails of all transactions and status changes, enabling robust reporting and analytics capabilities that help pharmacy managers make informed decisions about inventory management and business operations.
Furthermore, the system incorporates sophisticated features such as intelligent medicine substitution recommendations through a recursive relationship on the Medicine table. This is handled by the Substitute table, which links a med_id to a substitute_id and includes a substitution_reason.

The architecture also supports role-based access control for different user types—customers, pharmacy staff, and delivery personnel—each with appropriate permissions and customized interfaces. By maintaining detailed records of customer order history and medicine preferences, the system enables personalized shopping experiences and can generate automated reminders for prescription renewals or medicine restocks.
By tackling these real-world problems through a carefully designed database architecture, the MediQuick system serves as an ideal vessel to demonstrate the power and necessity of advanced SQL programming in a critical, reallife application domain. The implementation showcases how proper database design can ensure data consistency, enable complex business logic, and provide valuable business intelligence while maintaining the high levels of reliability and security required in healthcare applications. This project exemplifies how modern database technologies can transform traditional business processes into efficient, digital workflows that benefit all stakeholders—from patients seeking convenient healthcare solutions to pharmacies looking to expand their reach and optimize their operations.

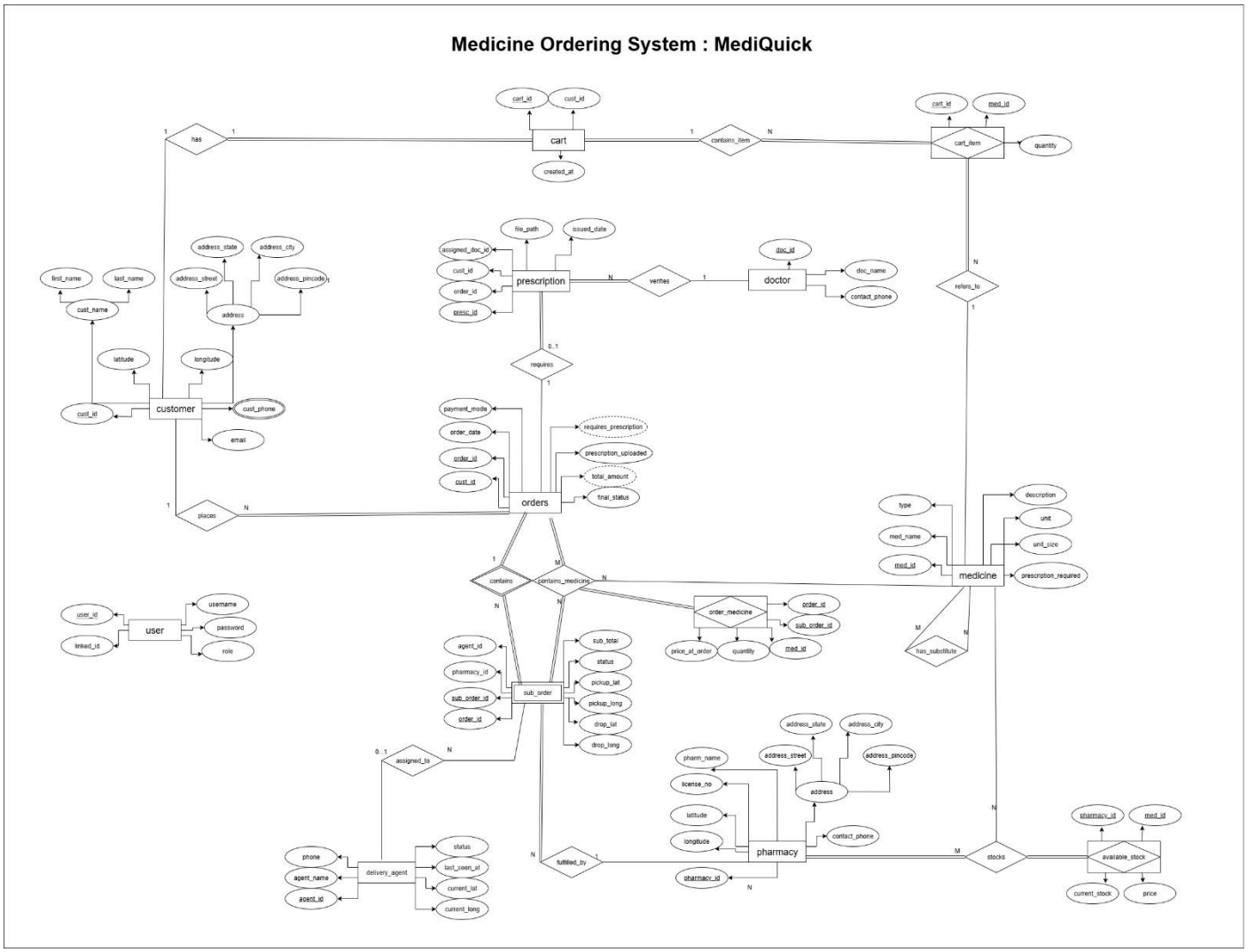
II. List of Softwares/Tools/Programming languages used

- Database**:** MySQL
- Programming Languages:
  - SQL: Used for all database operations, including table creation (DDL), data manipulation (DML), triggers, and stored procedures.
  - JavaScript: Used in all HTML files for front-end logic, including handling user input, calling the backend API (`fetch`), and dynamically updating the page content.
  - Python: Implied by the API endpoints (`http://127.0.0.1:5000/api`) referenced in the JavaScript, suggesting a Flask or similar Python web framework for the backend.
- Front-End Technologies:

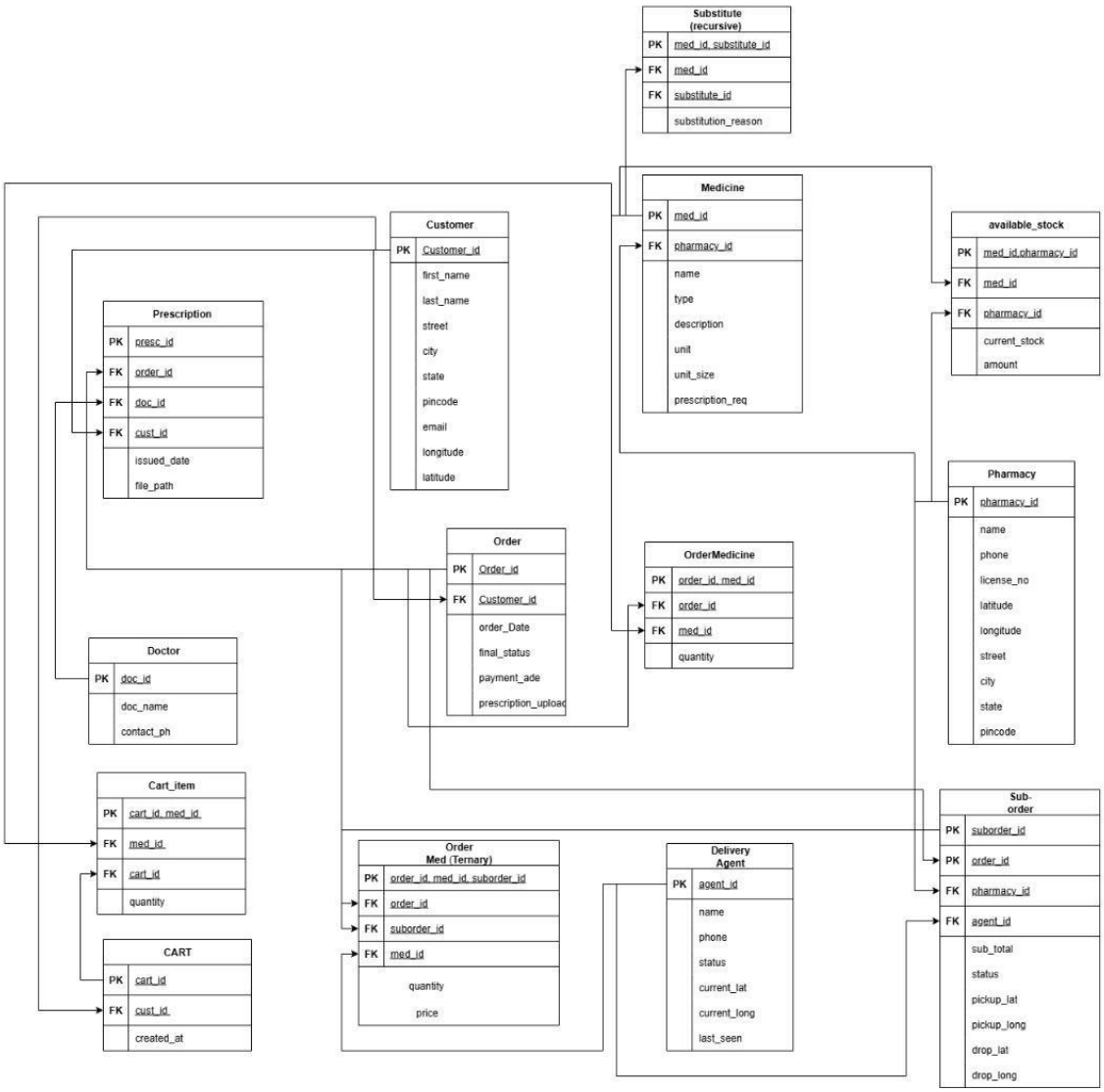
- HTML: Used to structure all web pages.
- Tailwind CSS: A utility-first CSS framework used for all styling, as indicated by the `<script src="https://cdn.tailwindcss.com"></script>` and class names (`bg-white`, `shadow-md`, `p-6`) in the HTML files.

- Tools & Version Control:
  - Git / GitHub: Used for version control and code hosting, as specified in the original report outline.

III. ER DIAGRAM



IV. Relational Schema

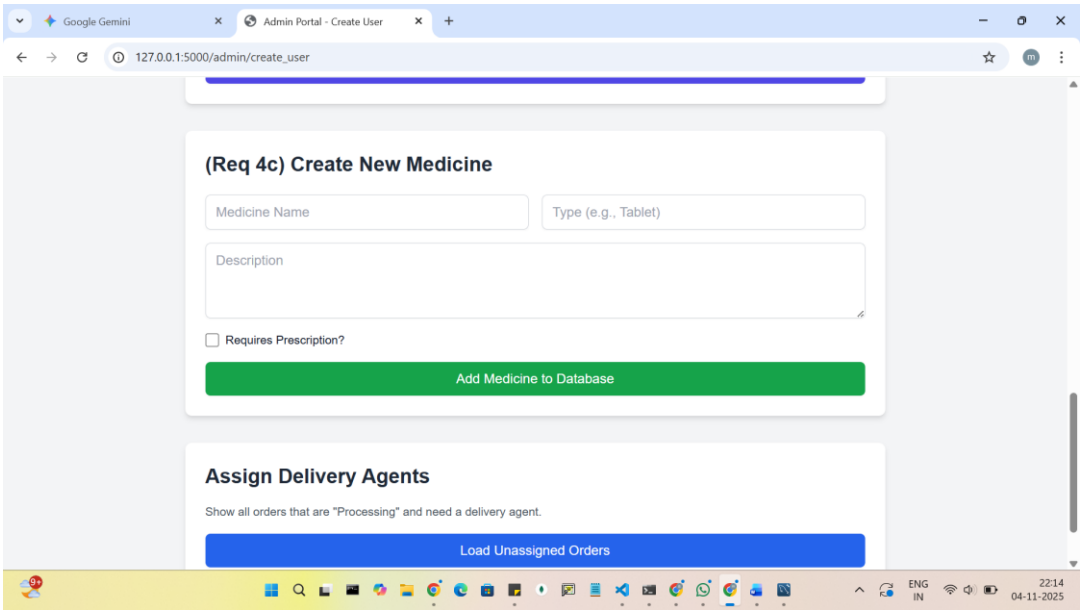


DDL Commands

V. CRUD operation Screenshots

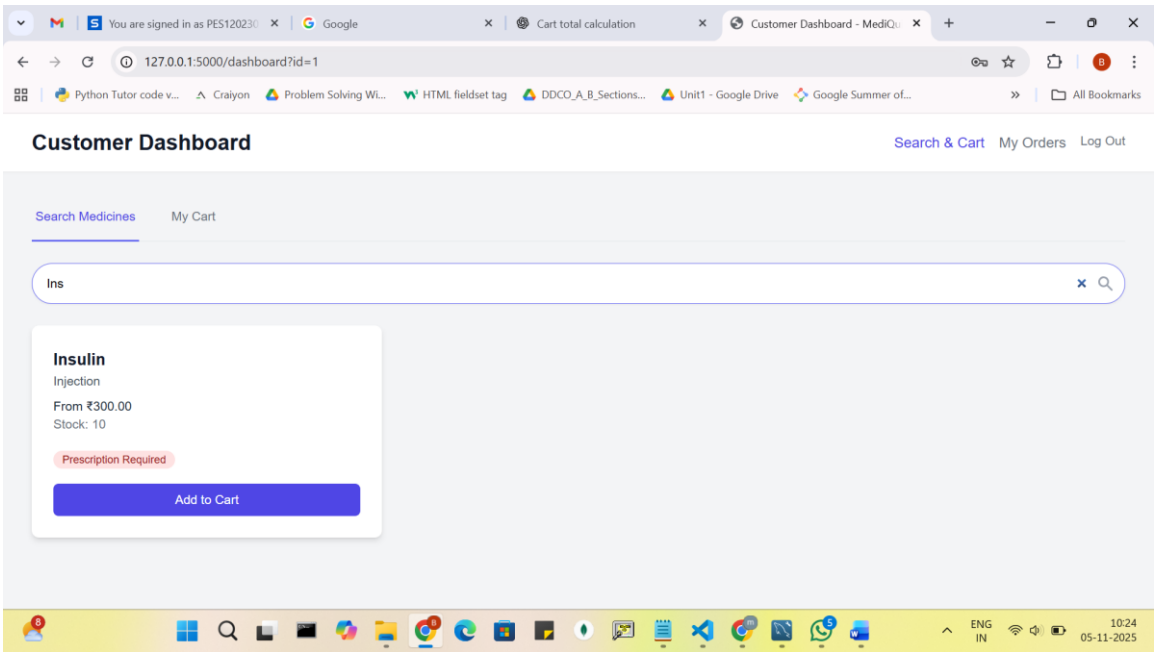
1] **CREATE:**

- **Description:** The Admin Portal (admin_create_user.html) provides a form labeled "(Req 4c) Create New Medicine". This form allows an administrator to **CREATE** a new record in the Medicine table by providing its name, type, description, and prescription status.



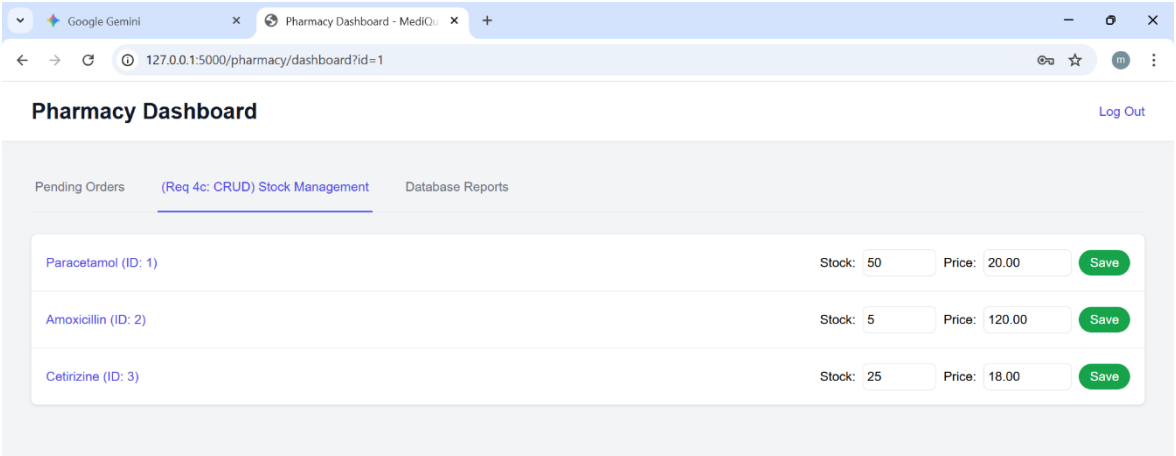
2] **READ:**

- **Description:** The Customer Dashboard (customer_dashboard.html) performs a **READ** operation. When the user types in the "Search Medicines" bar, the front-end fetches and displays all matching records from the Medicine table, joined with Available_Stock to show price and stock levels.



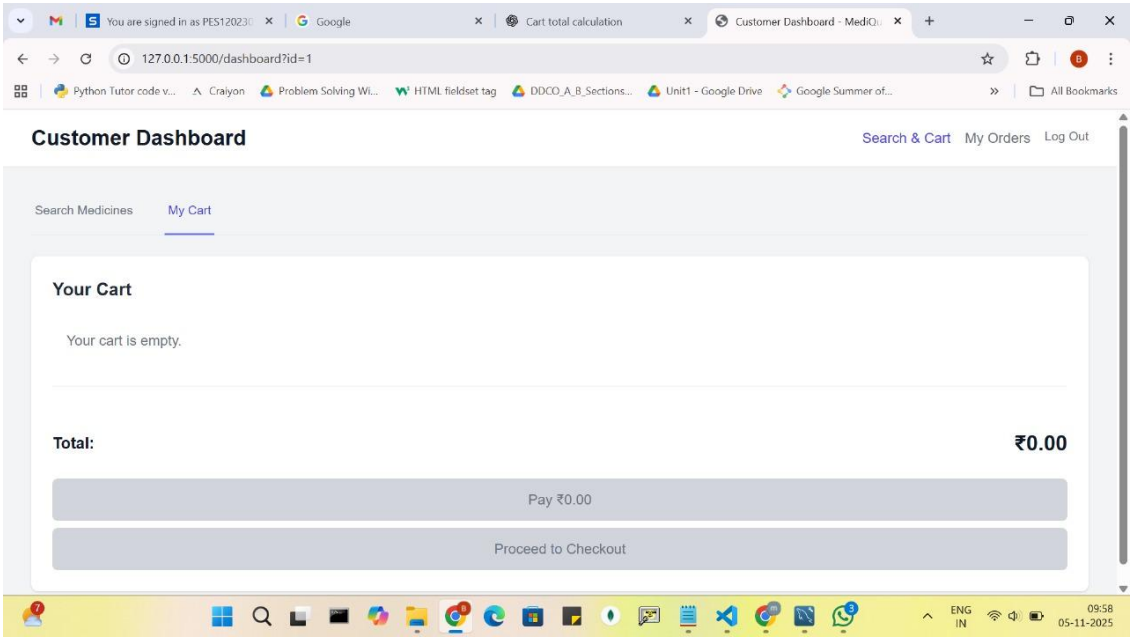
3] **UPDATE:**

- **Description:** The Pharmacy Dashboard (pharmacy_dashboard.html) implements the **UPDATE** operation. On the "Stock Management" tab, a pharmacy manager can see a list of their inventory and directly change the values in the "Stock" and "Price" input fields. Clicking "Save" sends these new values to the backend, which updates the Available_Stock table for that pharmacy_id and med_id.



4] **DELETE:**

- **Description:** The **DELETE** operation is executed when a customer checks out. The sp_process_cart_to_order_modular procedure, after successfully creating an order, calls the fn_clear_cart function. This function runs a DELETE FROM Cart_Item WHERE cart_id = p_cart_id command, deleting all items from the user's cart.

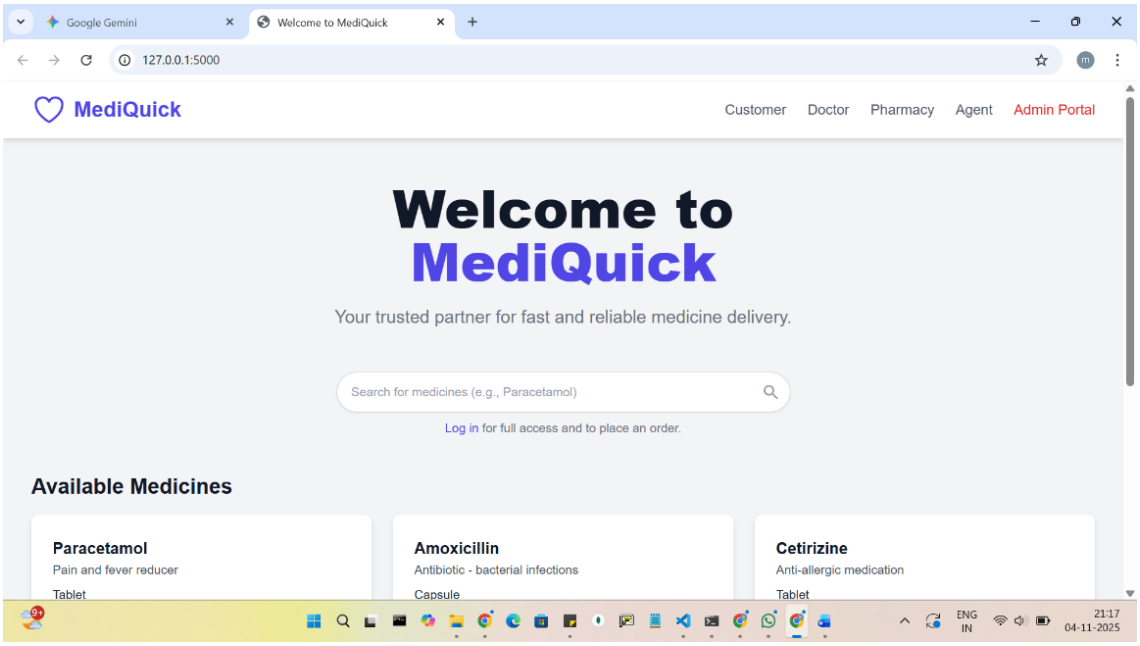


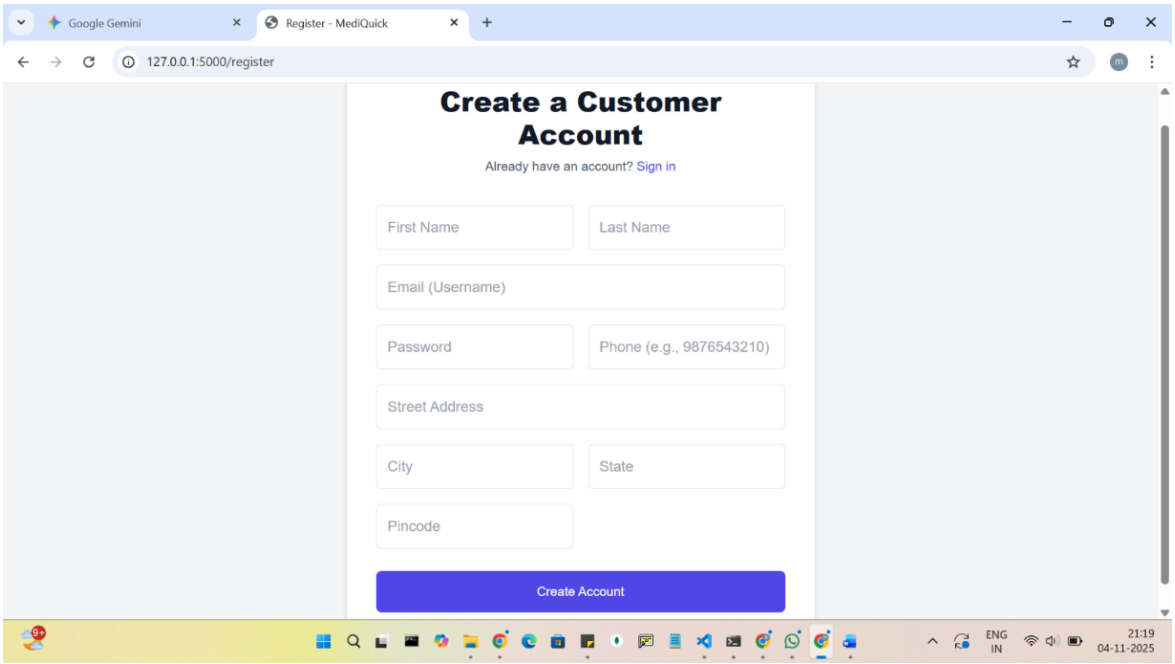
VI. List of functionalities/features of the application and its associated screenshots using front end

The application provides distinct, role-based interfaces for Customers, Pharmacy Staff, Doctors, and Administrators.

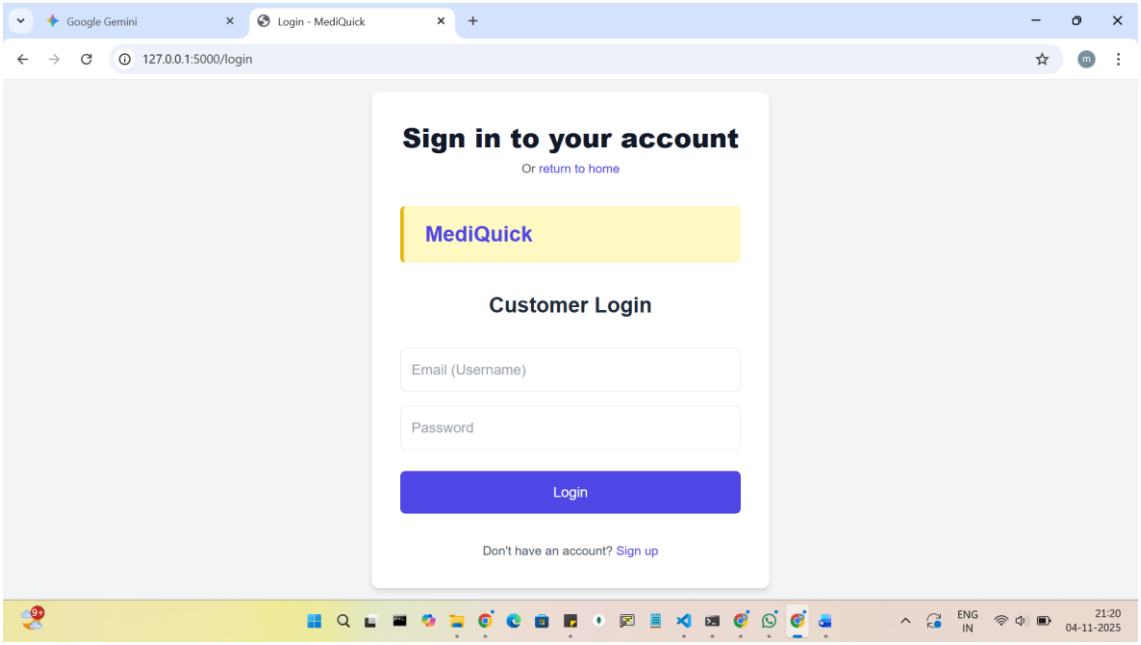
1. Customer Functionality

- **User Registration (`register.html`):** A public-facing page where new customers can create an account by providing their name, email, password, phone, and address.

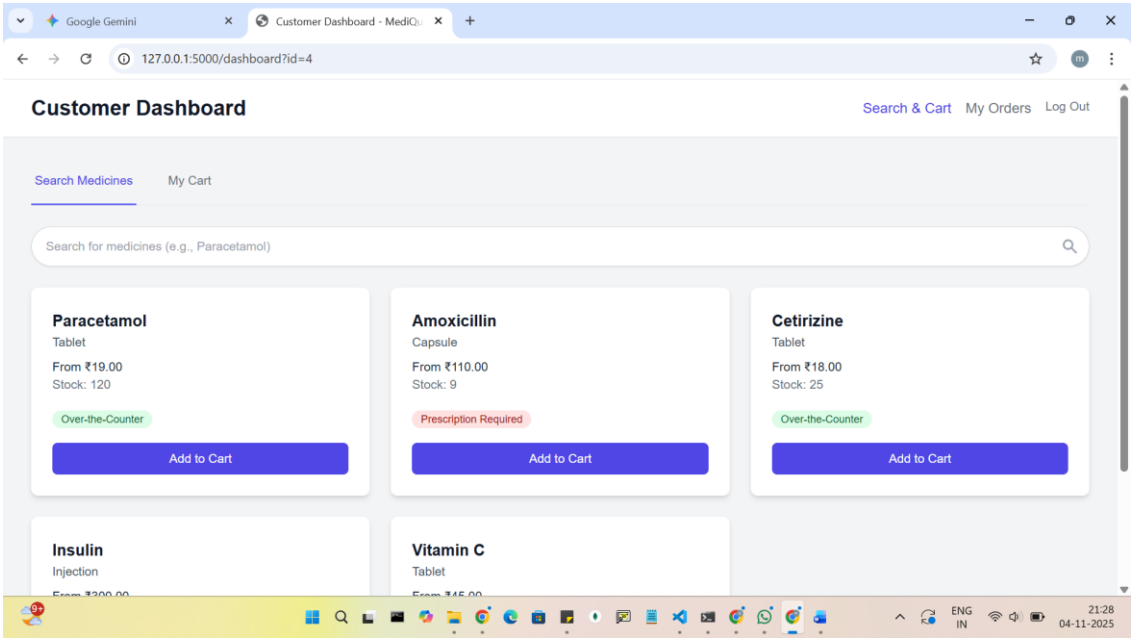




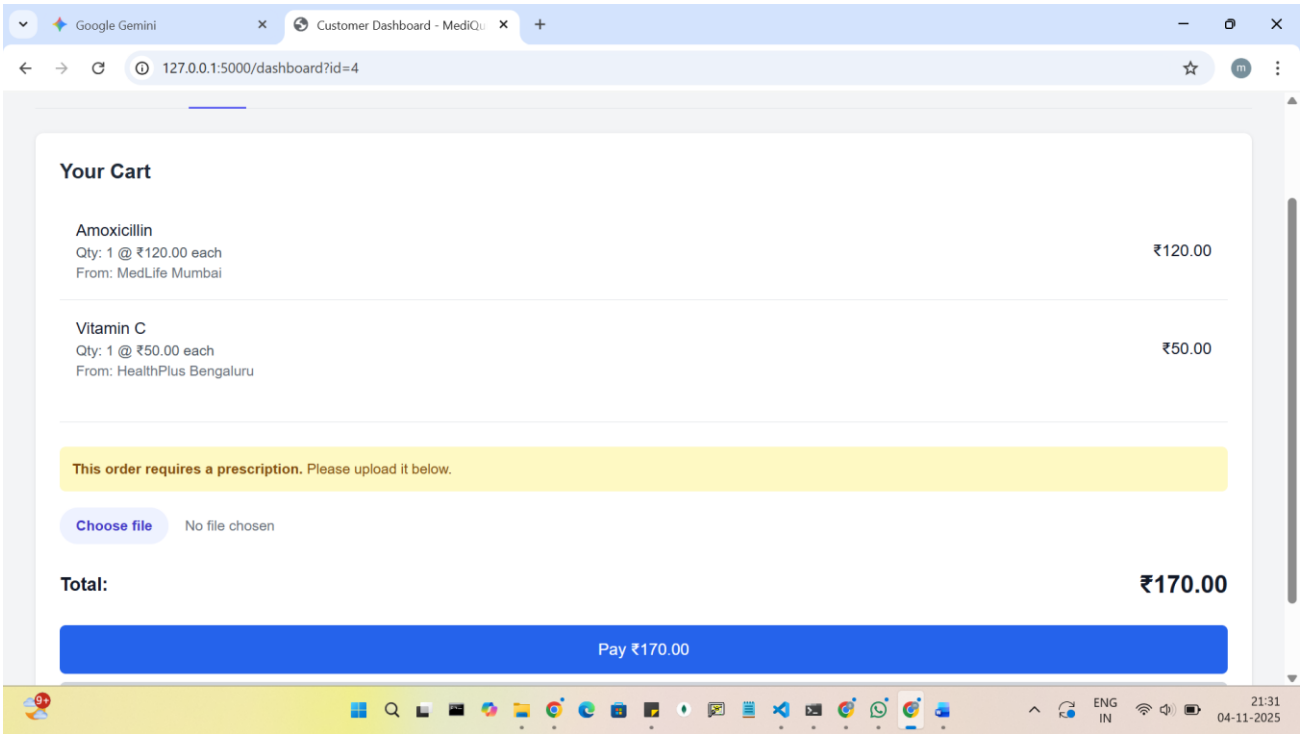
- **Customer Login (`login.html`):** A simple login page for returning customers to enter their email and password.



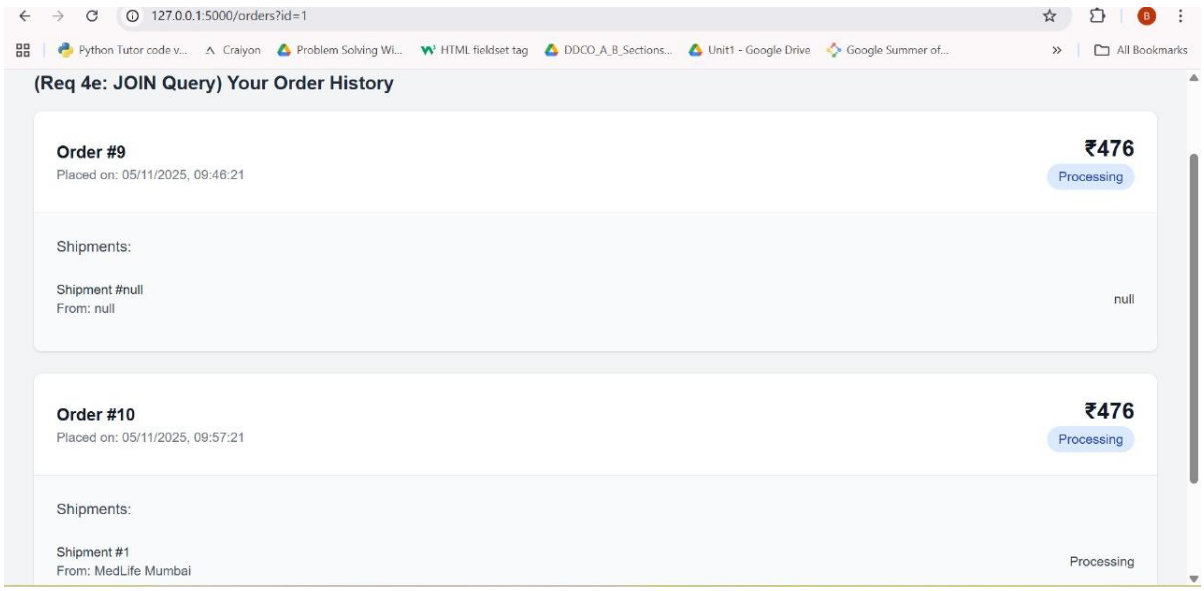
- **Customer Dashboard (`customer_dashboard.html`):** This is the main hub for customers.
  - **Medicine Search:** A search bar allows users to find medicines. Results show the medicine name, type, price, stock status, and whether a prescription is required.
  - **Add to Cart:** A button on each medicine search result allows the user to add the item to their cart. This action invokes the `sp_add_cart_item` procedure on the backend.

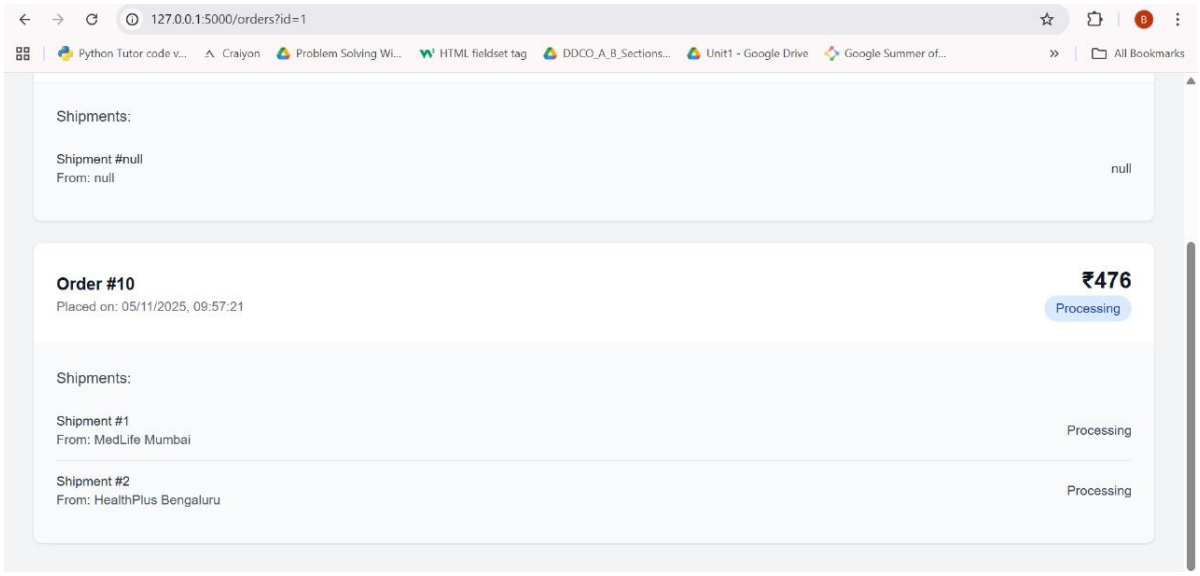


- **Cart Management (`customer_dashboard.html`):**
  - **View Cart:** A "My Cart" tab shows all items, their quantities, prices, and the total amount, which is calculated by `fn_get_cart_total`.
  - **Prescription Upload:** If an item requires a prescription (`fn_is_prescription_required`), a file upload prompt appears.
  - **Payment & Checkout:** A "Pay" button simulates payment, which then enables the "Proceed to Checkout" button. Clicking this invokes the `sp_process_cart_to_order_modular` procedure to finalize the order.



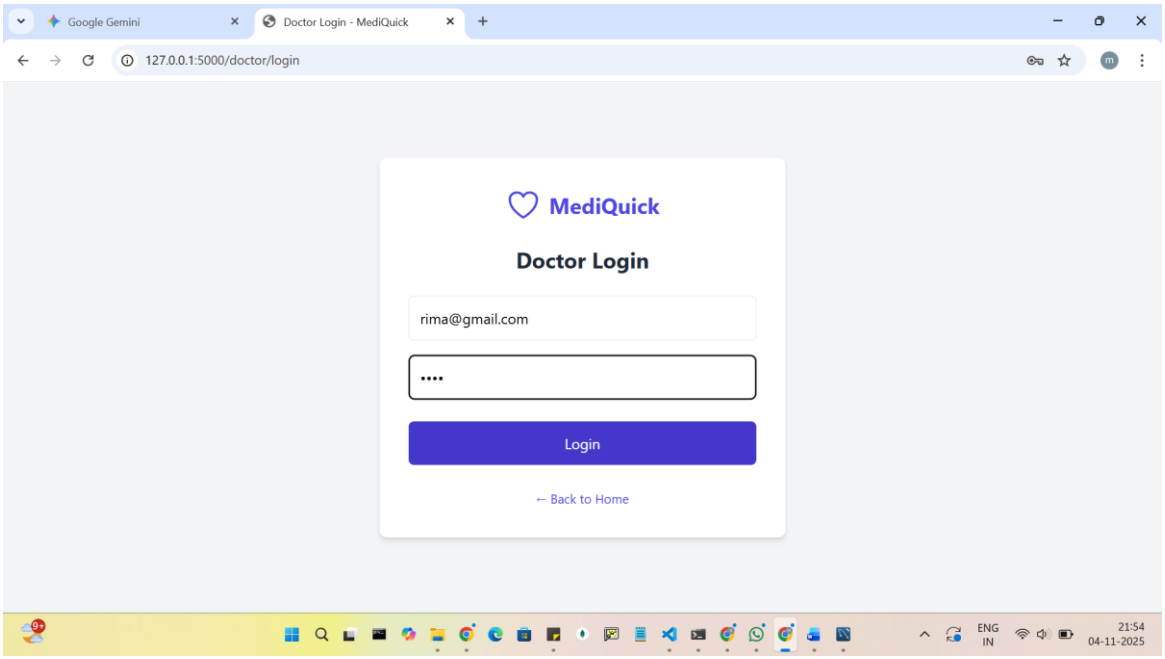
- **Order History (`customer_orders.html`):**
  - **View Orders (JOIN Query):** This page (labeled "Req 4e: JOIN Query") fetches and displays a list of all past orders for the logged-in customer. It shows the main order ID, date, total amount, and final status.
  - **View Sub-Orders:** Each order expands to show the individual sub-orders (shipments), the pharmacy that is fulfilling it, and the status of that specific shipment.


127.0.0.1:5000/orders?id=1
Python Tutor code v...
Craiyon
Problem Solving Wi...
HTML fieldset tag
DDCO_A_B_Sections...
Unit1 - Google Drive
Google Summer of...
All Bookmarks
(Req 4e: JOIN Query) Your Order History
Order #9
Placed on: 05/11/2025, 09:46:21
₹476
Processing
Shipments:
Shipment #null
From: null
null
Order #10
Placed on: 05/11/2025, 09:57:21
₹476
Processing
Shipments:
Shipment #1
From: MedLife Mumbai
Processing

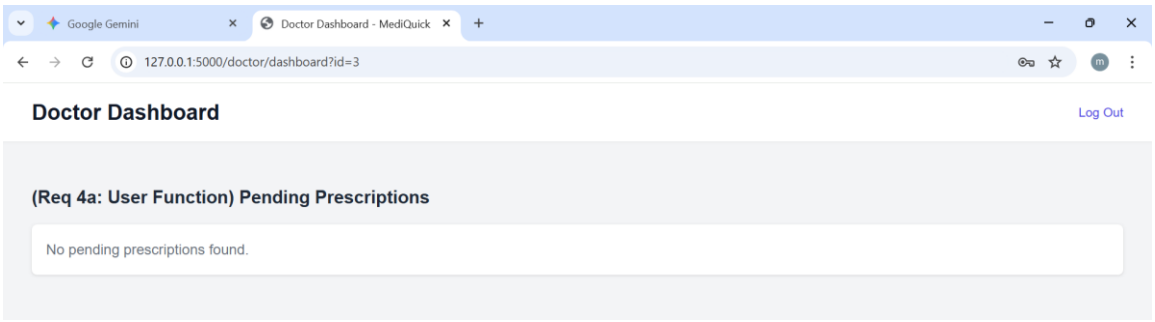

127.0.0.1:5000/orders?id=1
Python Tutor code v...
Craiyon
Problem Solving Wi...
HTML fieldset tag
DDCO_A_B_Sections...
Unit1 - Google Drive
Google Summer of...
All Bookmarks
Shipments:
Shipment #null
From: null
null
Order #10
Placed on: 05/11/2025, 09:57:21
₹476
Processing
Shipments:
Shipment #1
From: MedLife Mumbai
Processing
Shipment #2
From: HealthPlus Bengaluru
Processing

2. Doctor Functionality

- **Doctor Login (`role_login.html`):** A dedicated login page for users with the "Doctor" role.

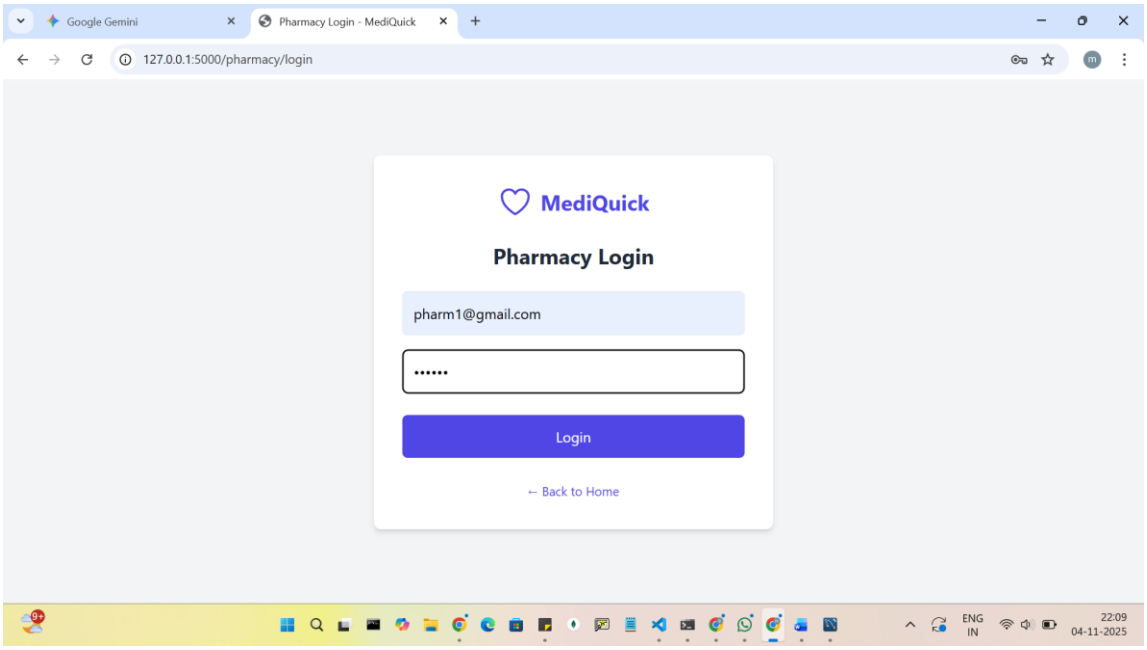


- **Doctor Dashboard (`doctor_dashboard.html`):**
  - **Verify Prescriptions (Procedure):** The dashboard (labeled "Req 4a: User Function") displays a list of all prescriptions with a "To Be Verified" status.
  - **Approve/Reject:** The doctor has "Verify" and "Reject" buttons for each prescription. Clicking these invokes the `sp_verify_prescription` procedure to update the prescription's status in the database.

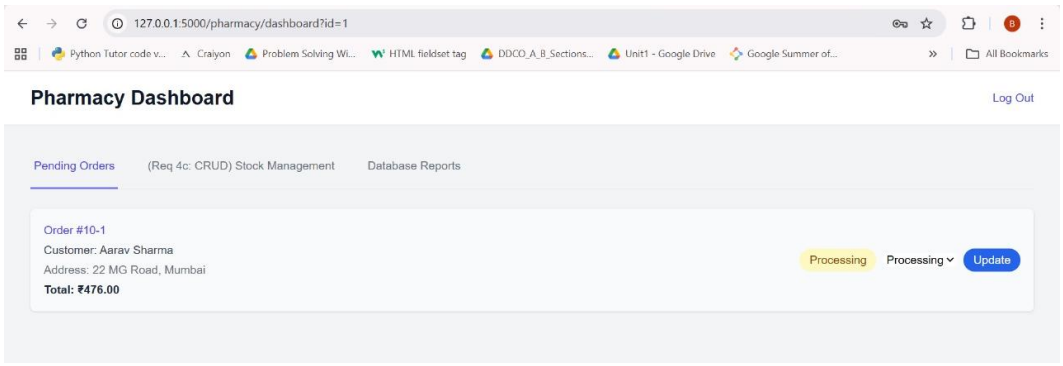


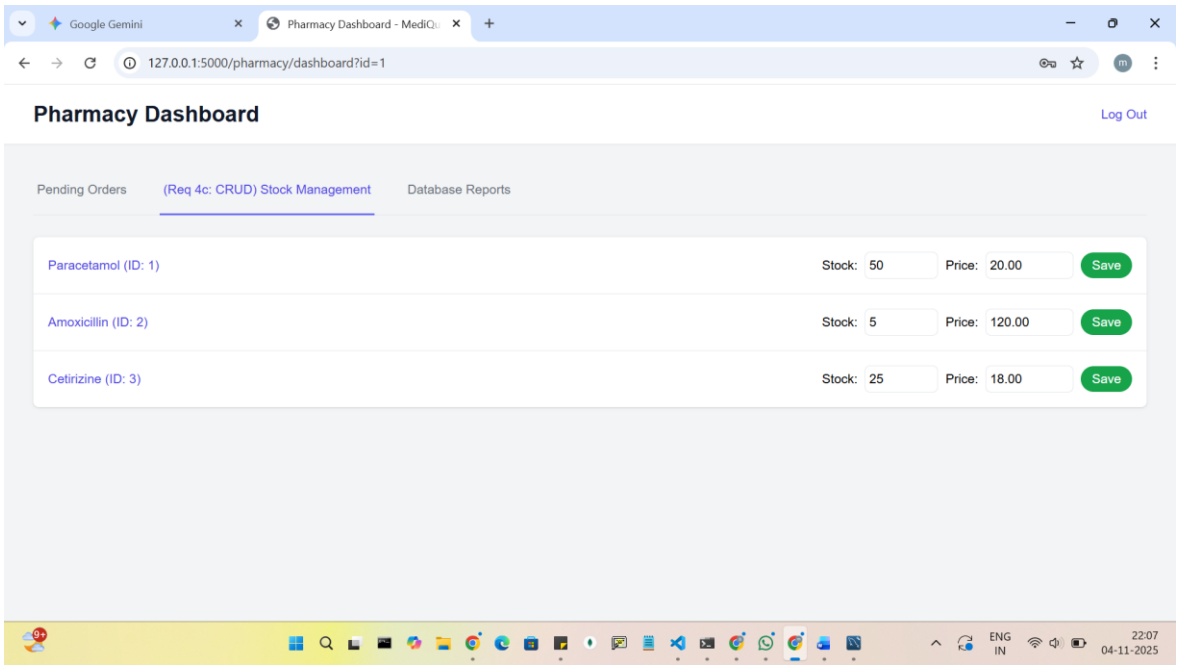
3. Pharmacy Functionality

- **Pharmacy Login (`role_login.html`):** A dedicated login page for users with the "Pharmacy" role.



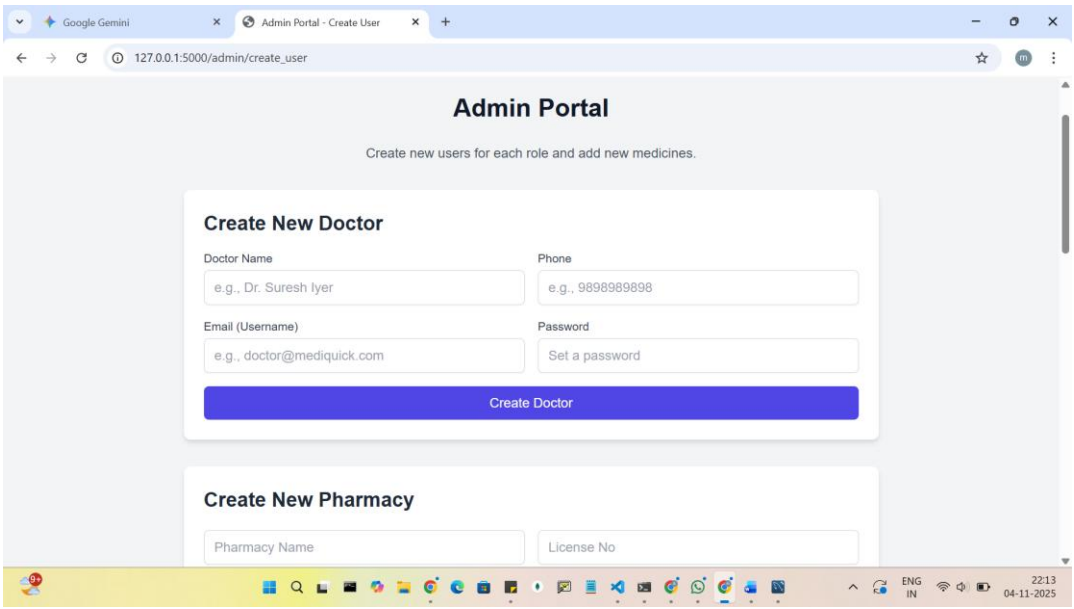
- **Pharmacy Dashboard (`pharmacy_dashboard.html`):**
  - **View Pending Orders:** The main "Pending Orders" tab lists all sub-orders assigned to that specific pharmacy.
  - **Update Order Status (CRUD):** The pharmacy manager can use a dropdown to change a sub-order's status (e.g., from "Processing" to "Shipped") and click "Update."
  - **Stock Management (CRUD):** A "Stock Management" tab (labeled "Req 4c: CRUD") lists all medicines stocked by the pharmacy. The manager can directly update the `current_stock` and `price` for each item and click "Save."

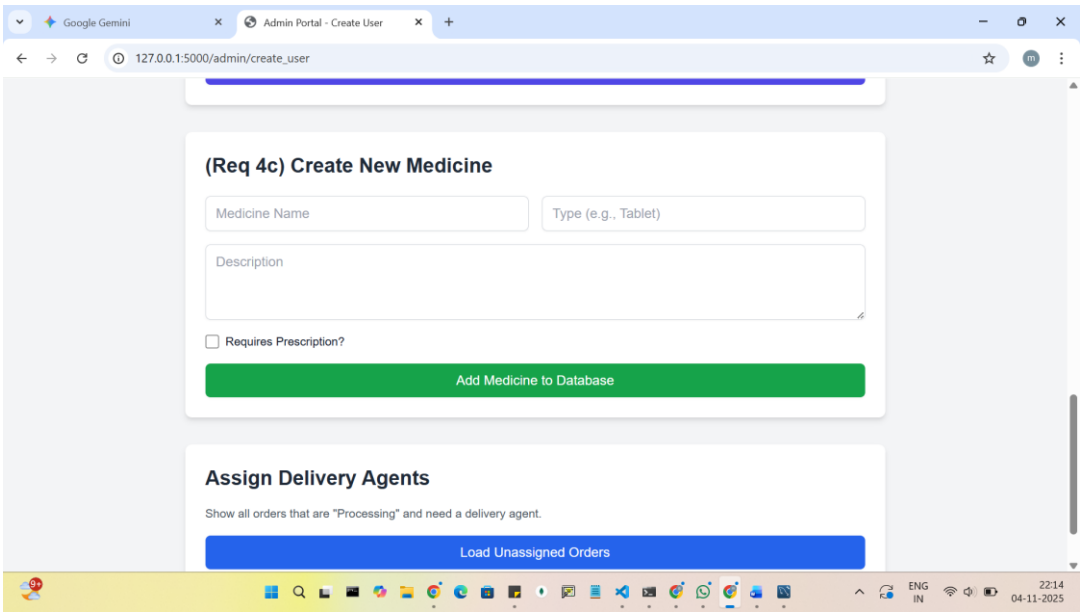


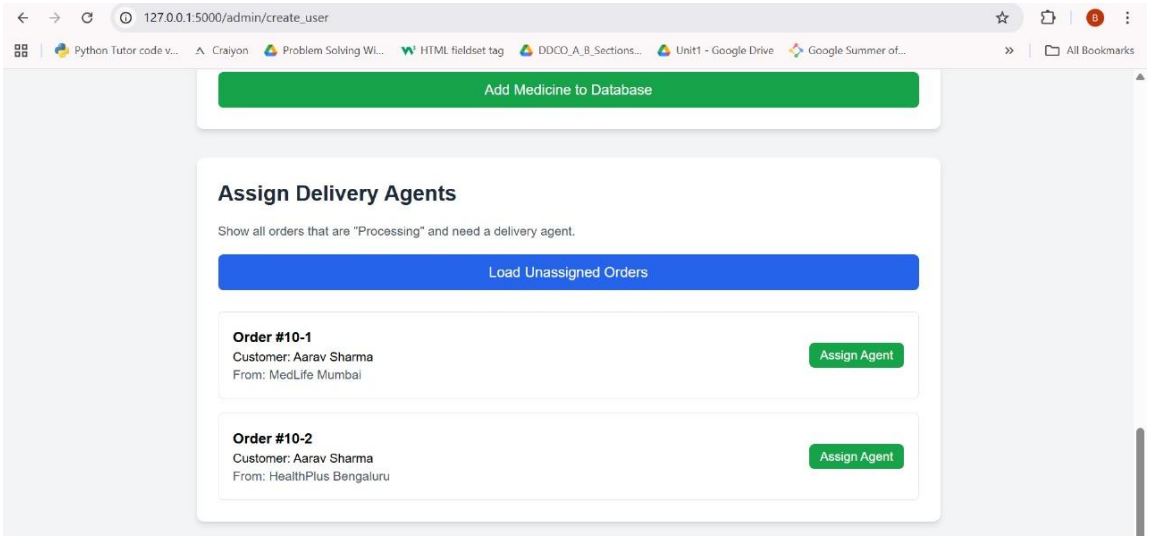


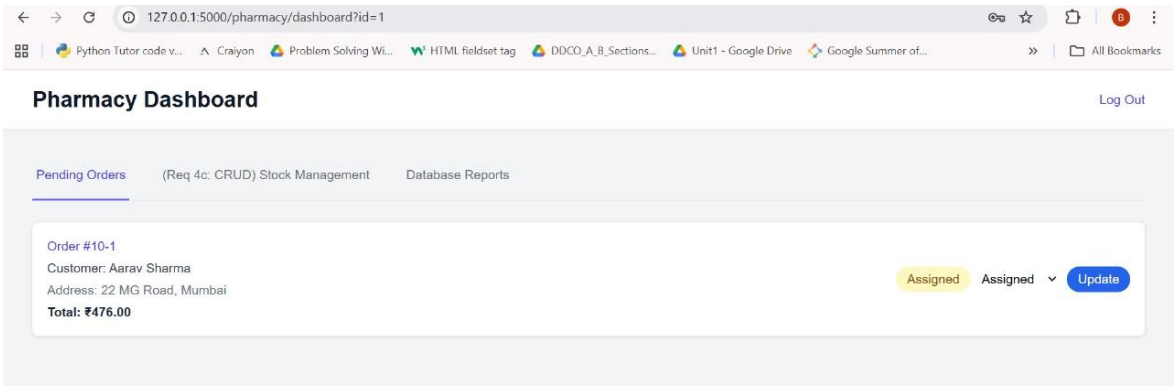
4. Admin Functionality

- **Admin Portal (`admin_create_user.html`):** A central page for administrative tasks.
    - **Create Users (CRUD):** Provides forms to create new Doctors, Pharmacies, and Delivery Agents, including their linked user accounts.
    - **Create Medicine (CRUD):** A form (labeled "Req 4c") allows the admin to add new medicines to the main `Medicine` table.
    - **Assign Delivery Agents (Procedure):** A section loads all unassigned sub-orders. An admin can click "Assign Agent" on an order, which invokes the `sp_assign_delivery_agent` procedure to find an available agent.

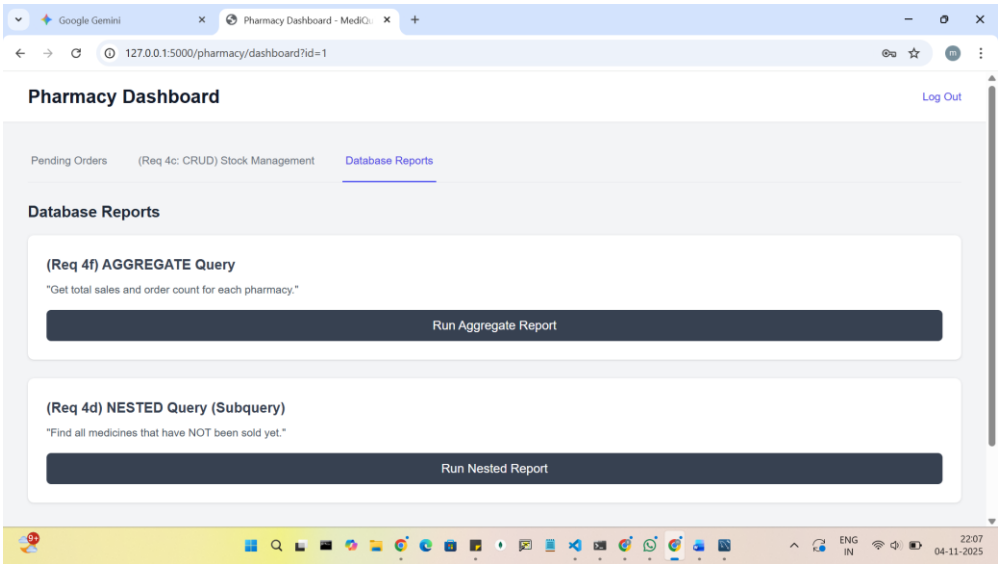

Google Gemini
Admin Portal - Create User
127.0.0.1:5000/admin/create_user
Admin Portal
Create new users for each role and add new medicines.
Create New Doctor
Doctor Name
e.g., Dr. Suresh Iyer
Phone
e.g., 9898989898
Email (Username)
e.g., doctor@mediquick.com
Password
Set a password
Create Doctor
Create New Pharmacy
Pharmacy Name
License No
ENG
IN
22:13
04-11-2025

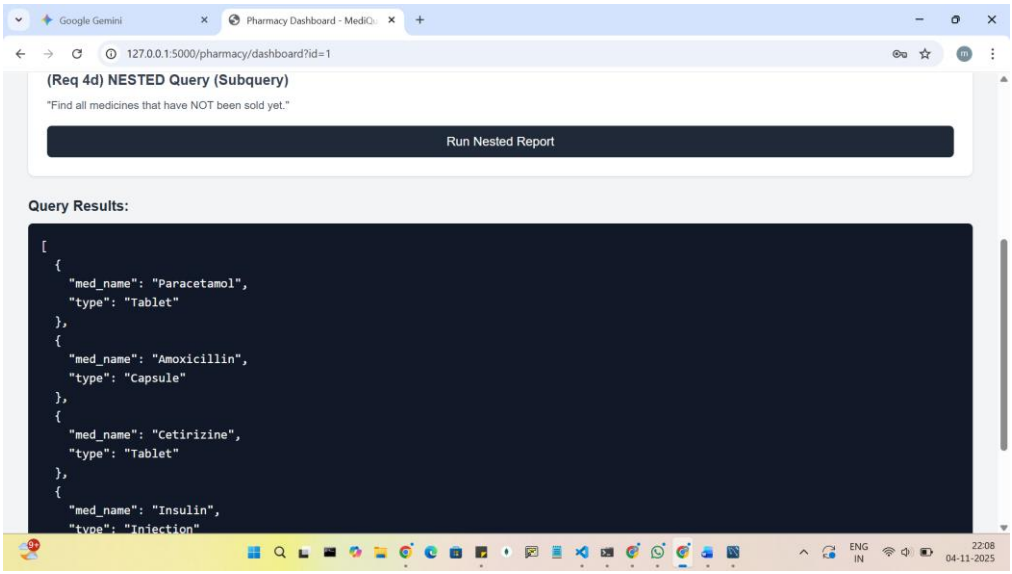

Google Gemini
Admin Portal - Create User
127.0.0.1:5000/admin/create_user
(Req 4c) Create New Medicine
Medicine Name
Type (e.g., Tablet)
Description
Requires Prescription?
Add Medicine to Database
Assign Delivery Agents
Show all orders that are "Processing" and need a delivery agent.
Load Unassigned Orders
ENG
IN
22:14
04-11-2025


127.0.0.1:5000/admin/create_user
Python Tutor code v...
Craiyon
Problem Solving Wi...
HTML fieldset tag
DDCO_A_B_Sections...
Unit1 - Google Drive
Google Summer of...
All Bookmarks
Add Medicine to Database
Assign Delivery Agents
Show all orders that are "Processing" and need a delivery agent.
Load Unassigned Orders
Order #10-1
Customer: Aarav Sharma
From: MedLife Mumbai
Assign Agent
Order #10-2
Customer: Aarav Sharma
From: HealthPlus Bengaluru
Assign Agent

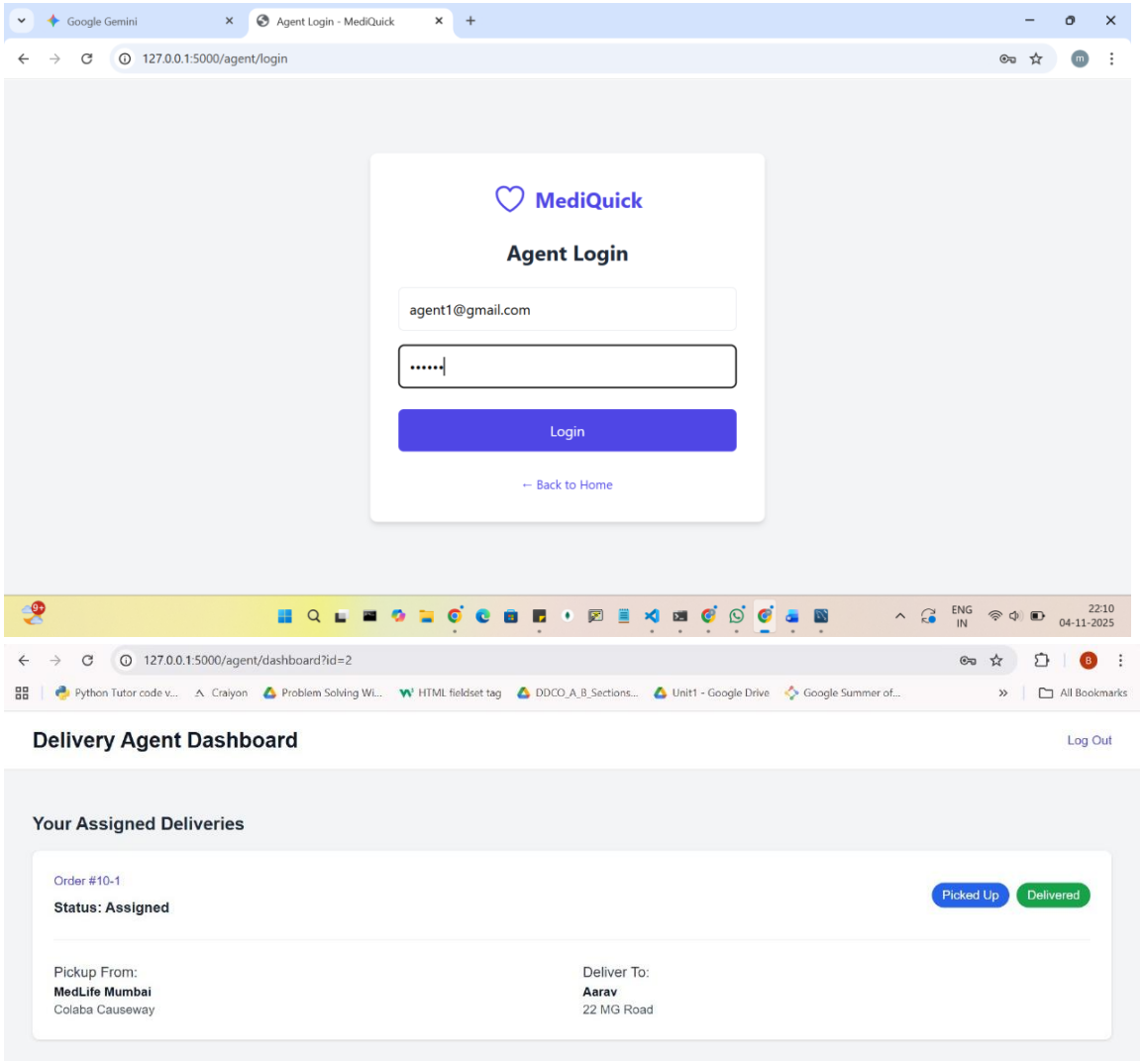


- **Database Reports (`reports.html`):**
  - **Run Aggregate/Nested Queries:** This page provides buttons to run pre-defined reports, such as an **Aggregate Query** ("Get total sales... for each pharmacy") and a **Nested Query** ("Find all medicines... not sold yet").

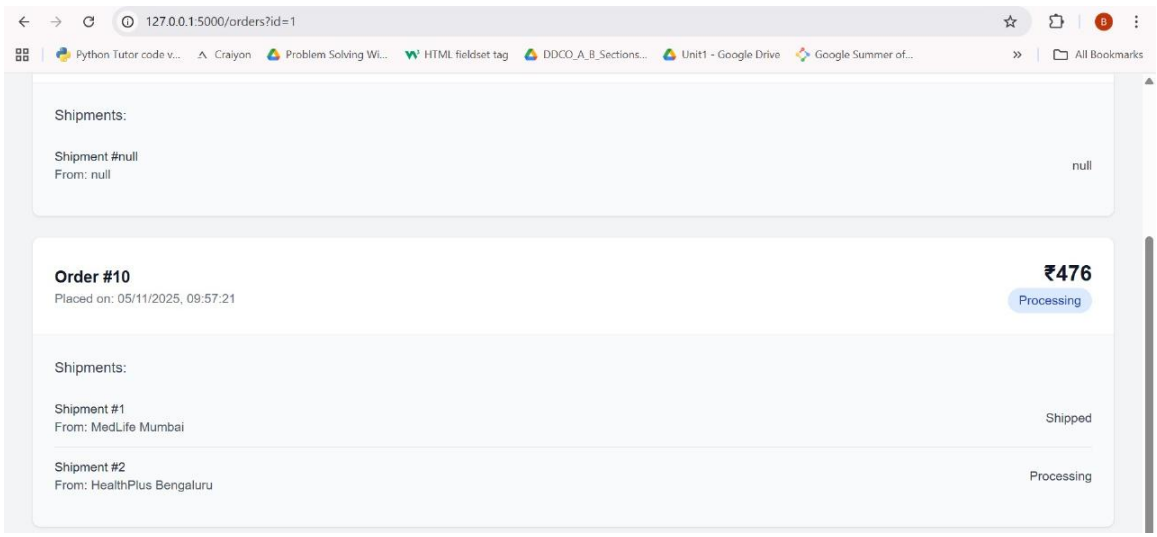




5. Delivery-Agent Dashboard



After the order is Delivered:



VII. Triggers, Procedures/Functions, Nested query, Join, Aggregate queries

This section details the core database logic implemented in the project.

1. Triggers

- trg_check_medicine_substitute (BEFORE INSERT): Prevents a medicine from being listed as its own substitute.
- trg_low_stock_alert (AFTER UPDATE): Automatically inserts a record into the Low_Stock_Alert table if an Available_Stock update causes current_stock to drop below 5.
- trg_cart_item_after_insert / _update / _delete: A set of triggers on the Cart_Item table. Any time an item is added, updated, or removed, these triggers automatically call the fn_get_cart_total and fn_is_prescription_required functions to update the parent Cart table's total_amount and requires_prescription fields, ensuring data consistency.

2. Procedures (Stored Procedures)

- sp_add_cart_item(p_cart_id, p_med_id, p_qty): Adds an item to the cart and then calls sp_assign_single_cart_item to find the best pharmacy for it.
- sp_assign_single_cart_item(p_cart_id, p_med_id): Finds the closest pharmacy that has the required medicine in stock (using fn_simple_distance) and updates the Cart_Item.assigned_pharmacy_id.
- sp_validate_cart_stock(p_cart_id, p_cust_lat, p_cust_lng): A critical procedure run before checkout. It checks if the assigned pharmacy for any cart item has run out of stock. If so, it attempts to re-assign the item to the next-closest available pharmacy. If no pharmacy has the stock, it raises an error.
- sp_process_cart_to_order_modular(p_cart_id): The main procedure for checking out. It performs several steps:

- Checks payment status using fn_check_payment_status.
- Calls sp_validate_cart_stock to fix any stock issues.

1. Creates a new entry in the Orders table.
2. Calls fn_create_sub_orders to split the cart into sub-orders based on the assigned pharmacy.
3. Calls fn_insert_order_medicines to move items from the cart to the Order_Medicine table and deducts the stock.
4. Calls fn_clear_cart to empty the user's cart.

- sp_assign_delivery_agent(p_sub_order_id): Finds an 'Available' delivery agent and assigns them to a sub-order, updating both the Sub_Order and Delivery_Agent tables.
- sp_verify_prescription(p_presc_id, p_doc_id, p_new_status): Used by the Doctor dashboard to update a prescription's status to 'Verified' or 'Rejected' and log the doctor who performed the action.

3. Functions

- **f**n_get_cart_total(p_cart_id): Calculates the total price of a cart by joining Cart_Item with Available_Stock on the assigned_pharmacy_id and summing the quantity * price.
- fn_is_prescription_required(p_cart_id): Returns TRUE if any item in the cart is linked to a medicine where prescription_required = TRUE.
- fn_simple_distance(...): A utility function that calculates the distance between two (lat, lng) points.
- fn_check_payment_status(p_cart_id): Returns the payment_status ('Pending' or 'Paid') of a cart.

4. Join Query (from customer_orders.html)

This query fetches all orders and sub-orders for a specific customer, joining multiple tables to get descriptive names.

```
SELECT
    o.order_id,
    o.order_date,
    o.total_amount,
    o.final_status,
    so.sub_order_id,
```

```
    so.status AS sub_order_status,
    p.pharm_name
FROM
    Orders o
JOIN
    Sub_Order so ON o.order_id = so.order_id
JOIN
    Pharmacy p ON so.pharmacy_id = p.pharmacy_id
WHERE
    o.cust_id = ?; -- (e.g., 1)
```

5. Aggregate Query

This query calculates the total sales and order count, grouping them by pharmacy.

```
SELECT
    p.pharm_name,
    COUNT(so.sub_order_id) AS total_orders_handled,
    SUM(so.sub_total) AS total_sales_value
FROM
    Sub_Order so
JOIN
    Pharmacy p ON so.pharmacy_id = p.pharmacy_id
WHERE
    so.status = 'Delivered'
GROUP BY
    p.pharm_name
ORDER BY
    total_sales_value DESC;
```

6. Nested Query (Subquery)

This query finds all medicines that have not yet been sold (i.e., do not appear in the Order_Medicine table).

```
SELECT
    med_id,
    med_name
FROM
    Medicine
WHERE
```

```
    med_id NOT IN (
        SELECT DISTINCT
            med_id
        FROM
            Order_Medicine
    );
```

VIII. Code snippets for invoking the Procedures/Functions/Trigger

These snippets from 7_tests.sql show how to execute the database logic directly.

1. Invoking a Stored Procedure (sp_add_cart_item) This snippet adds 2 units of medicine ID 1 to cart ID 1. This call will, in turn, trigger sp_assign_single_cart_item and the trg_cart_item_after_insert trigger.

```
-- Add Med 1 (Paracetamol) to Cart 1.
CALL sp_add_cart_item(1, 1, 2);
```

2. Invoking the Main Order Procedure (sp_process_cart_to_order_modular) This snippet processes cart 1, turning it into a formal order.

```
-- Process the order.
CALL sp_process_cart_to_order_modular(1);
```

3. Invoking the Doctor's Procedure (sp_verify_prescription) This snippet simulates Doctor 1 verifying prescription 1.

```
-- Test successful verification
CALL sp_verify_prescription(1, 1, 'Verified');
```

4. Invoking a Trigger (trg_low_stock_alert) This UPDATE statement will cause the trg_low_stock_alert to fire because the new current_stock (3) is less than the threshold (5).

-- Step 1: Update stock below threshold to trigger alert
UPDATE Available_Stock
SET current_stock = 3
WHERE pharmacy_id = 1 AND med_id = 1;

-- Step 2: Check that alert was created
SELECT * FROM Low_Stock_Alert
WHERE pharmacy_id = 1 AND med_id = 1;
5. Invoking a Function (in a SELECT statement) This snippet directly calls the function to check the stock of a specific medicine at a pharmacy.
-- Check stock of Med 1 at P1
SELECT fn_check_medicine_availability(1, 1);

IX. SQL queries(Create, Insert, Triggers, Procedures/Functions, Nested query, Join, Aggregate queries) used in the project in the form of .sql file

This section contains the complete SQL code used to build and populate the database.

1. DDL Commands (Table Creations)

```sql
/* 1) CUSTOMER */
CREATE TABLE Customer (
  cust_id INT AUTO_INCREMENT PRIMARY KEY,
  first_name VARCHAR(50) NOT NULL,
  last_name VARCHAR(50) NOT NULL,
  email VARCHAR(100) UNIQUE,
  address_street VARCHAR(150),
  address_city VARCHAR(50),
  address_state VARCHAR(50),
  address_pincode CHAR(6),
  latitude DECIMAL(10,6),
  longitude DECIMAL(10,6),
```

```
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

/* 2) CUSTOMER_PHONE (multivalued attribute) */
CREATE TABLE Customer_Phone (
  cust_id INT NOT NULL,
  phone VARCHAR(15) NOT NULL,
  PRIMARY KEY (cust_id, phone),
  FOREIGN KEY (cust_id) REFERENCES Customer(cust_id) ON DELETE CASCADE ON UPDATE CASCADE
);

/* 3) PHARMACY */
CREATE TABLE Pharmacy (
  pharmacy_id INT AUTO_INCREMENT PRIMARY KEY,
  license_no VARCHAR(50) UNIQUE,
  pharm_name VARCHAR(100) NOT NULL,
  contact_phone VARCHAR(15),
  address_street VARCHAR(150),
  address_city VARCHAR(50),
  address_state VARCHAR(50),
  address_pincode CHAR(6),
  latitude DECIMAL(10,6),
  longitude DECIMAL(10,6),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

/* 4) MEDICINE */
CREATE TABLE Medicine (
  med_id INT AUTO_INCREMENT PRIMARY KEY,
  med_name VARCHAR(150) NOT NULL,
  type VARCHAR(50),
  description TEXT,
  unit VARCHAR(20),
  unit_size VARCHAR(20),
  prescription_required BOOLEAN NOT NULL DEFAULT FALSE,
  UNIQUE (med_name)
);

/* 5) DOCTOR */
CREATE TABLE Doctor (
  doc_id INT AUTO_INCREMENT PRIMARY KEY,
  doc_name VARCHAR(100) NOT NULL,
  contact_phone VARCHAR(15),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

/* 6) DELIVERY_AGENT */
CREATE TABLE Delivery_Agent (
```

```
  agent_id INT AUTO_INCREMENT PRIMARY KEY,
  agent_name VARCHAR(100) NOT NULL,
  phone VARCHAR(15),
  current_lat DECIMAL(10,6),
  current_lng DECIMAL(10,6),
  last_seen_at TIMESTAMP NULL,
  status ENUM('Available','Busy','Offline') NOT NULL DEFAULT 'Offline'
);

/* 7) AVAILABLE_STOCK (associative) */
CREATE TABLE Available_Stock (
  pharmacy_id INT NOT NULL,
  med_id INT NOT NULL,
  current_stock INT NOT NULL DEFAULT 0,
  price DECIMAL(10,2) NOT NULL DEFAULT 0.00,
  PRIMARY KEY (pharmacy_id, med_id),
  FOREIGN KEY (pharmacy_id) REFERENCES Pharmacy(pharmacy_id) ON DELETE CASCADE ON UPDATE
CASCADE,
  FOREIGN KEY (med_id) REFERENCES Medicine(med_id) ON DELETE CASCADE ON UPDATE CASCADE,
  CHECK (current_stock >= 0),
  CHECK (price >= 0)
);


/* 8) CART */
CREATE TABLE Cart (
  cart_id INT AUTO_INCREMENT PRIMARY KEY,
  cust_id INT NOT NULL,
  cart_created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  requires_prescription BOOLEAN DEFAULT FALSE,  -- auto-calculated from items
  prescription_status ENUM('Not Uploaded','To Be Verified','Verified','Rejected') DEFAULT 'Not
Uploaded',
  total_amount DECIMAL(12,2) NOT NULL DEFAULT 0.00,
  payment_status ENUM('Pending','Paid') DEFAULT 'Pending',
  FOREIGN KEY (cust_id) REFERENCES Customer(cust_id) ON DELETE CASCADE ON UPDATE CASCADE,
  CHECK (total_amount >= 0)
);

/* 9) CART_ITEM (Associative Entity) */
CREATE TABLE Cart_Item (
  cart_id INT NOT NULL,
  med_id INT NOT NULL,
  quantity INT NOT NULL DEFAULT 1,
  assigned_pharmacy_id INT NULL,      -- for internal processng, we won't ask customer
  PRIMARY KEY (cart_id, med_id),
  FOREIGN KEY (cart_id) REFERENCES Cart(cart_id) ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (med_id) REFERENCES Medicine(med_id) ON DELETE RESTRICT ON UPDATE CASCADE,
  CHECK (quantity >= 0)
```

```
);

/* 10) USER (auth) */
CREATE TABLE `User` (
  user_id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(80) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL,
  role ENUM('Customer','Pharmacy','Doctor','Agent') NOT NULL,
  linked_id INT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

/* 11) ORDERS (parent for Sub_Order, Prescription, Order_Medicine) */
CREATE TABLE Orders (
  order_id INT AUTO_INCREMENT PRIMARY KEY,
  cust_id INT NOT NULL,
  order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  final_status ENUM('Processing','Partially Delivered','Delivered','Cancelled') DEFAULT
'Processing',
  total_amount DECIMAL(12,2) NOT NULL DEFAULT 0.00,
  FOREIGN KEY (cust_id) REFERENCES Customer(cust_id) ON DELETE CASCADE ON UPDATE CASCADE,
  CHECK (total_amount >= 0)
);


/* 12) SUB_ORDER (weak entity) */
CREATE TABLE Sub_Order (
  order_id INT NOT NULL,
  sub_order_id INT NOT NULL,
  pharmacy_id INT NOT NULL,
  agent_id INT NULL,
  sub_total DECIMAL(12,2) NOT NULL DEFAULT 0.00,
  status ENUM('Processing','Assigned','Shipped','Delivered','Cancelled') NOT NULL DEFAULT
'Processing',
  pickup_lat DECIMAL(10,6),
  pickup_lng DECIMAL(10,6),
  drop_lat DECIMAL(10,6),
  drop_lng DECIMAL(10,6),
  PRIMARY KEY (order_id, sub_order_id),
  FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (pharmacy_id) REFERENCES Pharmacy(pharmacy_id) ON DELETE RESTRICT ON UPDATE
CASCADE,
  FOREIGN KEY (agent_id) REFERENCES Delivery_Agent(agent_id) ON DELETE SET NULL ON UPDATE
CASCADE,
  CHECK (sub_total >= 0)
);

/* 13) ORDER_MEDICINE (ternary resolution) */
```

```sql
CREATE TABLE Order_Medicine (
   order_item_id INT AUTO_INCREMENT PRIMARY KEY,  -- unique row identifier
   order_id INT NOT NULL,
   sub_order_id INT NULL,                         -- only used if split
   med_id INT NOT NULL,
   quantity INT NOT NULL DEFAULT 1,
   price_at_order DECIMAL(10,2) NOT NULL DEFAULT 0.00,
   dosage_instructions VARCHAR(255),

   FOREIGN KEY (order_id) REFERENCES Orders(order_id)
      ON DELETE CASCADE ON UPDATE CASCADE,

   FOREIGN KEY (med_id) REFERENCES Medicine(med_id)
      ON DELETE RESTRICT ON UPDATE CASCADE,

   CHECK (quantity > 0),
   CHECK (price_at_order >= 0)
);


/* Composite FK from Order_Medicine to Sub_Order */
ALTER TABLE Order_Medicine
  ADD CONSTRAINT fk_ordermedicine_suborder
  FOREIGN KEY (order_id, sub_order_id)
  REFERENCES Sub_Order(order_id, sub_order_id)
  ON DELETE CASCADE ON UPDATE CASCADE;


/* 14) PRESCRIPTION */
CREATE TABLE Prescription (
  presc_id INT AUTO_INCREMENT PRIMARY KEY,
  order_id INT NOT NULL,
  cust_id INT NOT NULL,
  file_path VARCHAR(255) UNIQUE NOT NULL,
  assigned_doc_id INT NULL,
  issued_date DATE,
  uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  status ENUM('To Be Verified','Verified','Rejected') DEFAULT 'To Be Verified',
  verified_at TIMESTAMP NULL,
  FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (cust_id) REFERENCES Customer(cust_id) ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (assigned_doc_id) REFERENCES Doctor(doc_id) ON DELETE SET NULL ON UPDATE CASCADE
);
-- executed till here

/* 15) MEDICINE_SUBSTITUTE (recursive M:N) */
CREATE TABLE Medicine_Substitute (
  med_id INT NOT NULL,
```

```
  substitute_med_id INT NOT NULL,
  PRIMARY KEY (med_id, substitute_med_id),
  FOREIGN KEY (med_id) REFERENCES Medicine(med_id) ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (substitute_med_id) REFERENCES Medicine(med_id) ON DELETE CASCADE ON UPDATE
CASCADE
);

/* 16) Low_Stock_Alert (helper for low-stock trigger) */
CREATE TABLE Low_Stock_Alert (
  alert_id INT AUTO_INCREMENT PRIMARY KEY,
  pharmacy_id INT NOT NULL,
  med_id INT NOT NULL,
  stock_level INT NOT NULL,
  alert_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (pharmacy_id) REFERENCES Pharmacy(pharmacy_id) ON DELETE CASCADE ON UPDATE
CASCADE,
  FOREIGN KEY (med_id) REFERENCES Medicine(med_id) ON DELETE CASCADE ON UPDATE CASCADE
);

/* 17) SubOrder_Audit (helper for audit trigger) */
CREATE TABLE SubOrder_Audit (
  audit_id INT AUTO_INCREMENT PRIMARY KEY,
  order_id INT NOT NULL,
  sub_order_id INT NOT NULL,
  old_status VARCHAR(50),
  new_status VARCHAR(50),
  changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (order_id, sub_order_id) REFERENCES Sub_Order(order_id, sub_order_id) ON DELETE
CASCADE ON UPDATE CASCADE
);

/* 18) Useful Indexes */
CREATE INDEX idx_available_stock_med ON Available_Stock(med_id);
CREATE INDEX idx_orders_cust ON Orders(cust_id);
CREATE INDEX idx_suborder_pharm ON Sub_Order(pharmacy_id);
CREATE INDEX idx_ordermedicine_med ON Order_Medicine(med_id);
```

2. DML Commands (DATA POPULATION)

```
-- 1 CUSTOMERS
INSERT INTO Customer (first_name, last_name, email, address_street, address_city,
address_state, address_pincode, latitude, longitude)
VALUES
('Aarav', 'Sharma', 'aarav@gmail.com', '22 MG Road', 'Mumbai', 'Maharashtra', '400001',
19.0760, 72.8777),
```

```
('Priya', 'Mehta', 'priya@gmail.com', '101 Indiranagar', 'Bengaluru', 'Karnataka', '560038',
12.9716, 77.5946),
('Rahul', 'Verma', 'rahulv@gmail.com', '55 Gariahat', 'Kolkata', 'West Bengal', '700019',
22.5726, 88.3639);

-- 2 CUSTOMER PHONES
INSERT INTO Customer_Phone VALUES
(1, '9876543210'), (1, '9822221111'),
(2, '9900002233'),
(3, '9811122233');

-- 3 PHARMACIES
INSERT INTO Pharmacy (license_no, pharm_name, contact_phone, address_street, address_city,
address_state, address_pincode, latitude, longitude)
VALUES
('LIC1001', 'MedLife Mumbai', '9123456789', 'Colaba Causeway', 'Mumbai', 'Maharashtra',
'400005', 18.9218, 72.8330),
('LIC1002', 'HealthPlus Bengaluru', '9012345678', 'Koramangala 5th Block', 'Bengaluru',
'Karnataka', '560095', 12.9352, 77.6245),
('LIC1003', 'WellCare Kolkata', '9333344444', 'Salt Lake Sector 5', 'Kolkata', 'West Bengal',
'700091', 22.5790, 88.4273);

-- 4 MEDICINES
INSERT INTO Medicine (med_name, type, description, unit, unit_size, prescription_required)
VALUES
('Paracetamol', 'Tablet', 'Pain and fever reducer', 'strip', '10 tablets', FALSE),
('Amoxicillin', 'Capsule', 'Antibiotic - bacterial infections', 'strip', '10 capsules', TRUE),
('Cetirizine', 'Tablet', 'Anti-allergic medication', 'strip', '10 tablets', FALSE),
('Insulin', 'Injection', 'Diabetes treatment', 'vial', '10ml', TRUE),
('Vitamin C', 'Tablet', 'Immunity booster', 'strip', '10 tablets', FALSE);

-- 5 MEDICINE SUBSTITUTE (recursive)
INSERT INTO Medicine_Substitute VALUES
(1, 3), (2, 4), (3, 1);

-- 6 AVAILABLE STOCK
INSERT INTO Available_Stock (pharmacy_id, med_id, current_stock, price)
VALUES
(1, 1, 50, 20.00),   -- Paracetamol in Mumbai
(1, 2, 5, 120.00),    -- Amoxicillin - edge case low stock
(1, 3, 25, 18.00),

(2, 1, 40, 22.00),
(2, 4, 10, 300.00),
(2, 5, 20, 50.00),

(3, 1, 30, 19.00),
(3, 2, 4, 110.00),    -- will trigger low-stock alert after deduction
```

```
(3, 5, 10, 45.00);

-- 7️⃣ DOCTORS
INSERT INTO Doctor (doc_name, contact_phone)
VALUES
('Dr. Suresh Iyer', '9898989898'),
('Dr. Ritu Bansal', '9876500011');

-- 8️⃣ DELIVERY AGENTS
INSERT INTO Delivery_Agent (agent_name, phone, current_lat, current_lng, status)
VALUES
('Ramesh Kumar', '9001112222', 19.08, 72.87, 'Available'),
('Sita Das', '9123344556', 12.97, 77.59, 'Available'),
('Karan Joshi', '9330093300', 22.57, 88.36, 'Offline');

-- 9️⃣ CARTS
INSERT INTO Cart (cust_id, payment_status)
VALUES
(1, 'Paid'),    -- Aarav - fully paid
(2, 'Pending'),-- Priya - unpaid (to show error)
(3, 'Paid');    -- Rahul - paid

-- 🔟 CART ITEMS
INSERT INTO Cart_Item (cart_id, med_id, quantity) VALUES
(1, 1, 2),  -- Paracetamol
(1, 2, 1),  -- Amoxicillin (needs prescription)
(1, 5, 3),  -- Vitamin C
(2, 1, 1),
(2, 4, 2),
(3, 4, 1),  -- Insulin (needs prescription)
(3, 5, 2);

-- ============================================================
-- 🔐 USER ACCOUNTS AND MYSQL USERS WITH ROLES
-- ============================================================
-- Note: MySQL roles must be created first (see 2_roles.sql)

-- 1️⃣ CUSTOMER USERS (3 customers)
-- Create User table entries
INSERT INTO User (username, password, role, linked_id) VALUES
('cust1@gmail.com', 'cust1', 'Customer', 1),
('cust2@gmail.com', 'cust2', 'Customer', 2),
('cust3@gmail.com', 'cust3', 'Customer', 3);

-- Create MySQL users with customer_role (using short format: role_id)
-- MySQL username limit is 32 characters, so we use: cust1, cust2, etc.
CREATE USER IF NOT EXISTS 'cust1'@'localhost' IDENTIFIED BY 'cust1';
GRANT 'customer_role'@'localhost' TO 'cust1'@'localhost';
```

```
SET DEFAULT ROLE 'customer_role'@'localhost' TO 'cust1'@'localhost';

CREATE USER IF NOT EXISTS 'cust2'@'localhost' IDENTIFIED BY 'cust2';
GRANT 'customer_role'@'localhost' TO 'cust2'@'localhost';
SET DEFAULT ROLE 'customer_role'@'localhost' TO 'cust2'@'localhost';

CREATE USER IF NOT EXISTS 'cust3'@'localhost' IDENTIFIED BY 'cust3';
GRANT 'customer_role'@'localhost' TO 'cust3'@'localhost';
SET DEFAULT ROLE 'customer_role'@'localhost' TO 'cust3'@'localhost';

-- 2️⃣ DOCTOR USERS (2 doctors)
-- Create User table entries
INSERT INTO User (username, password, role, linked_id) VALUES
('doc1@gmail.com', 'doc1', 'Doctor', 1),
('doc2@gmail.com', 'doc2', 'Doctor', 2);

-- Create MySQL users with doctor_role (using short format: role_id)
CREATE USER IF NOT EXISTS 'doc1'@'localhost' IDENTIFIED BY 'doc1';
GRANT 'doctor_role'@'localhost' TO 'doc1'@'localhost';
SET DEFAULT ROLE 'doctor_role'@'localhost' TO 'doc1'@'localhost';

CREATE USER IF NOT EXISTS 'doc2'@'localhost' IDENTIFIED BY 'doc2';
GRANT 'doctor_role'@'localhost' TO 'doc2'@'localhost';
SET DEFAULT ROLE 'doctor_role'@'localhost' TO 'doc2'@'localhost';

-- 3️⃣ PHARMACY USERS (3 pharmacies)
-- Create User table entries
INSERT INTO User (username, password, role, linked_id) VALUES
('pharm1@gmail.com', 'pharm1', 'Pharmacy', 1),
('pharm2@gmail.com', 'pharm2', 'Pharmacy', 2),
('pharm3@gmail.com', 'pharm3', 'Pharmacy', 3);

-- Create MySQL users with pharmacy_role (using short format: role_id)
CREATE USER IF NOT EXISTS 'pharm1'@'localhost' IDENTIFIED BY 'pharm1';
GRANT 'pharmacy_role'@'localhost' TO 'pharm1'@'localhost';
SET DEFAULT ROLE 'pharmacy_role'@'localhost' TO 'pharm1'@'localhost';

CREATE USER IF NOT EXISTS 'pharm2'@'localhost' IDENTIFIED BY 'pharm2';
GRANT 'pharmacy_role'@'localhost' TO 'pharm2'@'localhost';
SET DEFAULT ROLE 'pharmacy_role'@'localhost' TO 'pharm2'@'localhost';

CREATE USER IF NOT EXISTS 'pharm3'@'localhost' IDENTIFIED BY 'pharm3';
GRANT 'pharmacy_role'@'localhost' TO 'pharm3'@'localhost';
SET DEFAULT ROLE 'pharmacy_role'@'localhost' TO 'pharm3'@'localhost';

-- 4️⃣ DELIVERY AGENT USERS (3 agents)
-- Create User table entries
INSERT INTO User (username, password, role, linked_id) VALUES
```

```
('agent1@gmail.com', 'agent1', 'Agent', 1),
('agent2@gmail.com', 'agent2', 'Agent', 2),
('agent3@gmail.com', 'agent3', 'Agent', 3);

-- Create MySQL users with agent_role (using short format: role_id)
CREATE USER IF NOT EXISTS 'agent1'@'localhost' IDENTIFIED BY 'agent1';
GRANT 'agent_role'@'localhost' TO 'agent1'@'localhost';
SET DEFAULT ROLE 'agent_role'@'localhost' TO 'agent1'@'localhost';

CREATE USER IF NOT EXISTS 'agent2'@'localhost' IDENTIFIED BY 'agent2';
GRANT 'agent_role'@'localhost' TO 'agent2'@'localhost';
SET DEFAULT ROLE 'agent_role'@'localhost' TO 'agent2'@'localhost';

CREATE USER IF NOT EXISTS 'agent3'@'localhost' IDENTIFIED BY 'agent3';
GRANT 'agent_role'@'localhost' TO 'agent3'@'localhost';
SET DEFAULT ROLE 'agent_role'@'localhost' TO 'agent3'@'localhost';

-- Flush privileges to apply all changes
FLUSH PRIVILEGES;
```

3. Triggers

```
-- T1) Medicine_Substitute: prevent self-substitution
DELIMITER $$
CREATE TRIGGER trg_check_medicine_substitute
BEFORE INSERT ON Medicine_Substitute
FOR EACH ROW
BEGIN
    IF NEW.med_id = NEW.substitute_med_id THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'med_id cannot equal substitute_med_id';
    END IF;
END$$
DELIMITER ;

-- T2) Available_Stock: low stock alert
DELIMITER $$
```

```
CREATE TRIGGER trg_low_stock_alert
AFTER UPDATE ON Available_Stock
FOR EACH ROW
BEGIN
    IF NEW.current_stock < 5 THEN
        INSERT INTO Low_Stock_Alert(pharmacy_id, med_id, stock_level, alert_time)
        VALUES (NEW.pharmacy_id, NEW.med_id, NEW.current_stock, NOW());
    END IF;
END$$
DELIMITER ;

-- T3) cart_item_after_insert: Automatically update Cart totals
DELIMITER $$
CREATE TRIGGER trg_cart_item_after_insert
AFTER INSERT ON Cart_Item
FOR EACH ROW
BEGIN
    UPDATE Cart
    SET
        total_amount = fn_get_cart_total(NEW.cart_id),
        requires_prescription = fn_is_prescription_required(NEW.cart_id)
    WHERE cart_id = NEW.cart_id;
END$$
DELIMITER ;

-- T4) cart_item_after_update
DELIMITER $$
CREATE TRIGGER trg_cart_item_after_update
AFTER UPDATE ON Cart_Item
FOR EACH ROW
BEGIN
    UPDATE Cart
    SET
        total_amount = fn_get_cart_total(NEW.cart_id),
        requires_prescription = fn_is_prescription_required(NEW.cart_id)
    WHERE cart_id = NEW.cart_id;
END$$
DELIMITER ;

-- T5) cart_item_after_delete
DELIMITER $$
CREATE TRIGGER trg_cart_item_after_delete
AFTER DELETE ON Cart_Item
FOR EACH ROW
BEGIN
    UPDATE Cart
    SET
        total_amount = fn_get_cart_total(OLD.cart_id),
```

```
        requires_prescription = fn_is_prescription_required(OLD.cart_id)
    WHERE cart_id = OLD.cart_id;
END$$
DELIMITER ;
```

4. Procedures and Functions

```
-- Function 1: Get Cart Total
DELIMITER $$
CREATE FUNCTION fn_get_cart_total(p_cart_id INT)
RETURNS DECIMAL(12,2)
DETERMINISTIC
BEGIN
    DECLARE total DECIMAL(12,2);

    SELECT COALESCE(SUM(ci.quantity * s.price), 0)
    INTO total
    FROM Cart_Item ci
     JOIN Available_Stock s
       ON ci.med_id = s.med_id
       AND ci.assigned_pharmacy_id = s.pharmacy_id
    WHERE ci.cart_id = p_cart_id;

    RETURN total;
END$$
DELIMITER ;

-- Function 2: Check Prescription Requirement
DELIMITER $$
CREATE FUNCTION fn_is_prescription_required(p_cart_id INT)
RETURNS BOOLEAN
DETERMINISTIC
BEGIN
    DECLARE v_requires_prescription BOOLEAN;

    SELECT EXISTS(
        SELECT 1
```

```
        FROM Cart_Item ci
        JOIN Medicine m ON ci.med_id = m.med_id
        WHERE ci.cart_id = p_cart_id
          AND m.prescription_required = TRUE
    )
    INTO v_requires_prescription;

    RETURN v_requires_prescription;
END$$
DELIMITE ;

-- Function 3: Simple Distance
DELIMITER $$
CREATE FUNCTION fn_simple_distance(
    lat1 DECIMAL(10,6),
    lng1 DECIMAL(10,6),
    lat2 DECIMAL(10,6),
    lng2 DECIMAL(10,6)
) RETURNS DECIMAL(12,6) DETERMINISTIC
BEGIN
    RETURN ABS(lat1 - lat2) + ABS(lng1 - lng2);
END$$
DELIMITER ;

-- (Other helper functions like fn_check_payment_status, fn_create_sub_orders, etc. are also
included)

-- Procedure 1: Assign Single Cart Item
DELIMITER $$
CREATE PROCEDURE sp_assign_single_cart_item(IN p_cart_id INT, IN p_med_id INT)
BEGIN
    DECLARE v_cust_lat DECIMAL(10,6);
    DECLARE v_cust_lng DECIMAL(10,6);
    DECLARE v_quantity INT;

    SELECT c.latitude, c.longitude, ci.quantity
    INTO v_cust_lat, v_cust_lng, v_quantity
    FROM Cart ca
    JOIN Customer c ON ca.cust_id = c.cust_id
    JOIN Cart_Item ci ON ca.cart_id = ci.cart_id
    WHERE ca.cart_id = p_cart_id AND ci.med_id = p_med_id
    LIMIT 1;

    -- Update the Cart_Item row with the best pharmacy
    UPDATE Cart_Item ci
    JOIN (
        SELECT
            a.pharmacy_id,
```

```
            fn_simple_distance(v_cust_lat, v_cust_lng, p.latitude, p.longitude) AS dist
        FROM Available_Stock a
        JOIN Pharmacy p ON p.pharmacy_id = a.pharmacy_id
        WHERE a.med_id = p_med_id
          AND a.current_stock >= v_quantity
        ORDER BY dist ASC
        LIMIT 1
    ) best_ph
    SET ci.assigned_pharmacy_id = best_ph.pharmacy_id
    WHERE ci.cart_id = p_cart_id AND ci.med_id = p_med_id;
END$$
DELIMITER ;

-- Procedure 2: Add Item to Cart
DELIMITER $$
CREATE PROCEDURE sp_add_cart_item(IN p_cart_id INT, IN p_med_id INT, IN p_qty INT)
BEGIN
    INSERT INTO Cart_Item(cart_id, med_id, quantity)
    VALUES (p_cart_id, p_med_id, p_qty)
    ON DUPLICATE KEY UPDATE quantity = quantity + p_qty;

    CALL sp_assign_single_cart_item(p_cart_id, p_med_id);
END$$
DELIMITER ;

-- Procedure 3: Validate Cart Stock
DELIMITER $$
CREATE PROCEDURE sp_validate_cart_stock(
    IN p_cart_id INT,
    IN p_cust_lat DECIMAL(10,6),
    IN p_cust_lng DECIMAL(10,6)
)
BEGIN
    DECLARE v_failed_med_name VARCHAR(150) DEFAULT NULL;

    -- (Self-healing logic to re-assign cart items if stock ran out)
    WITH BadItems AS (
        SELECT ci.med_id, ci.quantity, ci.assigned_pharmacy_id
        FROM Cart_Item ci
        JOIN Available_Stock av
            ON ci.med_id = av.med_id
            AND ci.assigned_pharmacy_id = av.pharmacy_id
        WHERE ci.cart_id = p_cart_id
          AND ci.quantity > av.current_stock
    ),
    RankedNewPharmacies AS (
        SELECT
            bi.med_id,
```

```
            a.pharmacy_id AS new_pharmacy_id,
            ROW_NUMBER() OVER(
                PARTITION BY bi.med_id
                ORDER BY fn_simple_distance(p_cust_lat, p_cust_lng, p.latitude, p.longitude)
ASC
            ) as rnk
        FROM BadItems bi
        JOIN Available_Stock a ON bi.med_id = a.med_id
        JOIN Pharmacy p ON a.pharmacy_id = p.pharmacy_id
        WHERE
            a.current_stock >= bi.quantity
            AND a.pharmacy_id != bi.assigned_pharmacy_id
    )
    UPDATE Cart_Item ci
    JOIN RankedNewPharmacies rnp
        ON ci.med_id = rnp.med_id AND ci.cart_id = p_cart_id
    SET
        ci.assigned_pharmacy_id = rnp.new_pharmacy_id
    WHERE
        rnp.rnk = 1;

    -- (Check for unfixable items and throw error)
    SELECT m.med_name
    INTO v_failed_med_name
    FROM Cart_Item ci
    JOIN Available_Stock av
        ON ci.med_id = av.med_id
        AND ci.assigned_pharmacy_id = av.pharmacy_id
    JOIN Medicine m ON ci.med_id = m.med_id
    WHERE ci.cart_id = p_cart_id
      AND ci.quantity > av.current_stock
    LIMIT 1;

    IF v_failed_med_name IS NOT NULL THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = CONCAT('Error: "', v_failed_med_name, '" is out of stock at all
nearby pharmacies.');
    END IF;

END$$
DELIMITER ;


-- Procedure 4: Process Cart to Order
DELIMITER $$
CREATE PROCEDURE sp_process_cart_to_order_modular(IN p_cart_id INT)
BEGIN
    DECLARE v_cust_id INT;
```

```
    DECLARE paid_status ENUM('Pending','Paid');
    DECLARE v_final_total DECIMAL(12,2);
    DECLARE new_order_id INT;
    DECLARE v_cust_lat DECIMAL(10,6);
    DECLARE v_cust_lng DECIMAL(10,6);

    SELECT fn_check_payment_status(p_cart_id) INTO paid_status;
    IF paid_status != 'Paid' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Payment not completed for this cart.';
    END IF;

    SELECT ca.cust_id, ca.total_amount, c.latitude, c.longitude
    INTO v_cust_id, v_final_total, v_cust_lat, v_cust_lng
    FROM Cart ca
    JOIN Customer c ON ca.cust_id = c.cust_id
    WHERE ca.cart_id = p_cart_id;

    CALL sp_validate_cart_stock(p_cart_id, v_cust_lat, v_cust_lng);

    SELECT total_amount INTO v_final_total
    FROM Cart
    WHERE cart_id = p_cart_id;

    INSERT INTO Orders(cust_id, total_amount)
    VALUES (v_cust_id, v_final_total);
    SET new_order_id = LAST_INSERT_ID();

    SELECT fn_create_sub_orders(p_cart_id, new_order_id);
    SELECT fn_insert_order_medicines(p_cart_id, new_order_id);
    SELECT fn_clear_cart(p_cart_id);

END$$
DELIMITER ;

-- Procedure 5: Assign Delivery Agent
DELIMITER $$
CREATE PROCEDURE sp_assign_delivery_agent(IN p_sub_order_id INT)
BEGIN
    DECLARE assigned_agent INT;

    SELECT da.agent_id INTO assigned_agent
    FROM Delivery_Agent da
    JOIN Sub_Order so ON so.sub_order_id = p_sub_order_id
    JOIN Order_Medicine om ON om.order_id = so.order_id
    WHERE da.status = 'Available' AND fn_check_medicine_availability(om.med_id,
so.pharmacy_id) > 0
    LIMIT 1;
```

```
    IF assigned_agent IS NOT NULL THEN
        UPDATE Sub_Order
        SET agent_id = assigned_agent,
            status = 'Assigned'
        WHERE sub_order_id = p_sub_order_id;

        UPDATE Delivery_Agent
        SET status = 'Busy'
        WHERE agent_id = assigned_agent;
    END IF;
END$$
DELIMITER ;

-- Procedure 6: Verify Prescription
DELIMITER $$
CREATE PROCEDURE sp_verify_prescription(
    IN p_presc_id INT,
    IN p_doc_id INT,
    IN p_new_status ENUM('Verified', 'Rejected')
)
BEGIN
    UPDATE Prescription
    SET
        status = p_new_status,
        assigned_doc_id = p_doc_id,
        verified_at = CURRENT_TIMESTAMP
    WHERE
        presc_id = p_presc_id
        AND status = 'To Be Verified';
END$$
DELIMITER ;
```

X. Github repo link

https://github.com/archi829/MediQuick