medical\_shop/

│── main.py # Entry point of the application( This file will start the application and open the **dashboard**.)

│── auth.py

│── db.py # Database connection file

│── inventory\_manager.py # Inventory Management (CRUD)

│── billing.py # Billing & Sales Management

│── reports.py # Sales & GST Reports

│── email\_fetch.py # Fetch PDFs from Email

│── ui/

│ │── dashboard.py # Dashboard UI (This file will contain the **main window** of the app. )

│ │── inventory\_ui.py # Inventory UI

│ │── billing\_ui.py # Billing UI

│ │── reports\_ui.py # Reports UI

│── assets/

│ │── icon.ico # App icon

│ │── logo.png # App logo

│── requirements.txt # List of required libraries

└── README.md # Project documentation

**Development Plan (Step-by-Step)**

**✅ Phase 1: Set Up Database & Connection**

* Create MySQL database (db.py).
* Connect Python to MySQL.

**✅ Phase 2: Build UI with PyQt6**

* Create dashboard.py (Main window).
* Add navigation buttons.

**✅ Phase 3: Develop Inventory Management**

* Create inventory\_manager.py (CRUD operations).
* Build inventory\_ui.py (UI).

**✅ Phase 4: Develop Billing System**

* Create billing.py (Invoice generation).
* Build billing\_ui.py (UI).

**✅ Phase 5: Generate Reports**

* Create reports.py (Data analysis).
* Build reports\_ui.py (UI).

**✅ Phase 6: Email Integration**

* Create email\_fetch.py (Fetch PDFs).

**✅ Phase 7: Final Testing & Deployment**

* Package as an .exe file.
* Install MySQL on client PC.

**Quries:**

CREATE DATABASE medical\_shop;

USE medical\_shop;

CREATE TABLE users (

user\_id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(50) NOT NULL UNIQUE,

password\_hash VARCHAR(255) NOT NULL,

email VARCHAR(100) UNIQUE,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE medicines (

medicine\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

category VARCHAR(50),

manufacturer VARCHAR(100),

batch\_no VARCHAR(50),

expiry\_date DATE,

price DECIMAL(10,2),

stock\_quantity INT DEFAULT 0,

gst\_percentage DECIMAL(5,2),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE vendors (

vendor\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

contact VARCHAR(15),

email VARCHAR(100),

address TEXT,

gst\_no VARCHAR(20) UNIQUE,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE purchases (

purchase\_id INT AUTO\_INCREMENT PRIMARY KEY,

vendor\_id INT,

medicine\_id INT,

purchase\_date DATE NOT NULL DEFAULT (CURRENT\_DATE), -- FIX: Use parentheses for CURRENT\_DATE

quantity INT NOT NULL,

purchase\_price DECIMAL(10,2) NOT NULL,

invoice\_pdf\_path VARCHAR(255),

FOREIGN KEY (vendor\_id) REFERENCES vendors(vendor\_id),

FOREIGN KEY (medicine\_id) REFERENCES medicines(medicine\_id)

);

CREATE TABLE sales (

sale\_id INT AUTO\_INCREMENT PRIMARY KEY,

medicine\_id INT,

sale\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

quantity INT,

total\_price DECIMAL(10,2),

gst\_amount DECIMAL(10,2),

FOREIGN KEY (medicine\_id) REFERENCES medicines(medicine\_id)

);

CREATE TABLE expiry\_alerts (

alert\_id INT AUTO\_INCREMENT PRIMARY KEY,

medicine\_id INT,

expiry\_date DATE,

status ENUM('pending', 'resolved') DEFAULT 'pending',

FOREIGN KEY (medicine\_id) REFERENCES medicines(medicine\_id)

);

CREATE TABLE notifications (

notification\_id INT AUTO\_INCREMENT PRIMARY KEY,

message TEXT,

notification\_type ENUM('stock', 'expiry', 'payment', 'other'),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

status ENUM('unread', 'read') DEFAULT 'unread'

);

CREATE TABLE settings (

setting\_id INT AUTO\_INCREMENT PRIMARY KEY,

setting\_name VARCHAR(50) NOT NULL UNIQUE,

setting\_value TEXT NOT NULL

);

**Command:**

pip install mysql-connector-python

###########################################

import sys

import os

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "..")))

from inventory\_manager import InventoryManager

**Dashbord**:

import sys

import os

import mysql.connector

import traceback

from PyQt6.QtWidgets import (

    QApplication, QMessageBox, QMainWindow, QWidget, QVBoxLayout, QPushButton, QLabel,

    QHBoxLayout, QTableWidget, QTableWidgetItem, QHeaderView

)

from PyQt6.QtGui import QIcon, QFont

from PyQt6.QtCore import Qt

from datetime import datetime, date

# ✅ Import database connection function

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))

from db import connect\_db

from stock\_alert import StockAlert

class Dashboard(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        # ✅ Establish MySQL connection

        self.conn = connect\_db()

        if not self.conn:

            QMessageBox.critical(self, "Database Error", "Failed to connect to MySQL! Exiting...")

            sys.exit(1)  # ❌ Exit app if DB connection fails

        self.cursor = self.conn.cursor()

        print("✅ Connected to MySQL!")

        # ✅ Set up the window

        self.setWindowTitle("Medical Shop Inventory Management")

        self.setGeometry(100, 100, 1000, 600)

        # ✅ Set absolute icon path

        icon\_path = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "../assets/icon.ico"))

        if os.path.exists(icon\_path):

            self.setWindowIcon(QIcon(icon\_path))

        else:

            print("⚠ Warning: Icon not found at", icon\_path)

        self.setStyleSheet("background-color: #f8f9fa;")

        # ✅ Main Widget and Layout

        main\_widget = QWidget()

        main\_layout = QVBoxLayout()

        main\_widget.setLayout(main\_layout)

        self.setCentralWidget(main\_widget)

        # ✅ Title Label

        title = QLabel("Medical Shop Management System")

        title.setAlignment(Qt.AlignmentFlag.AlignCenter)

        title.setFont(QFont("Arial", 22, QFont.Weight.Bold))

        title.setStyleSheet("color: #343a40; padding: 10px;")

        main\_layout.addWidget(title)

        # ✅ Navigation Buttons

        button\_layout = QHBoxLayout()

        self.dashboard\_btn = QPushButton("Dashboard")

        self.inventory\_btn = QPushButton("Inventory Management")

        self.billing\_btn = QPushButton("Billing & Sales")

        self.reports\_btn = QPushButton("Reports & Analysis")

        for btn in [self.dashboard\_btn, self.inventory\_btn, self.billing\_btn, self.reports\_btn]:

            btn.setFixedSize(250, 50)

            btn.setFont(QFont("Arial", 14, QFont.Weight.Bold))

            btn.setStyleSheet("""

                QPushButton {

                    background-color: #007BFF;

                    color: white;

                    border-radius: 8px;

                    border: none;

                    padding: 10px;

                }

                QPushButton:hover {

                    background-color: #0056b3;

                }

            """)

            btn.setCursor(Qt.CursorShape.PointingHandCursor)

            button\_layout.addWidget(btn)

        main\_layout.addLayout(button\_layout)

        # ✅ Expiry Alert Section

        alert\_label = QLabel("\n⚠ Expiry Date Alerts:")

        alert\_label.setFont(QFont("Arial", 18, QFont.Weight.Bold))

        alert\_label.setStyleSheet("color: red;")

        main\_layout.addWidget(alert\_label)

        self.expiry\_table = QTableWidget()

        self.expiry\_table.setColumnCount(3)

        self.expiry\_table.setHorizontalHeaderLabels(["Medicine Name", "Batch No", "Expiry Date"])

        self.expiry\_table.setStyleSheet("font-size: 14px; background-color: white; border-radius: 5px;")

        # ✅ Set column width and styling

        self.expiry\_table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode.Stretch)

        self.expiry\_table.horizontalHeader().setStyleSheet("font-size: 16px; font-weight: bold; color: #212529; background-color: #e9ecef;")

        self.expiry\_table.verticalHeader().setDefaultSectionSize(30)

        self.load\_expiry\_alerts()

        main\_layout.addWidget(self.expiry\_table)

        # ✅ Connect buttons to open respective windows

        self.inventory\_btn.clicked.connect(self.open\_inventory)

        self.billing\_btn.clicked.connect(self.open\_billing)

        self.reports\_btn.clicked.connect(self.open\_report)

    def open\_inventory(self):

        """Opens the Inventory Management UI."""

        try:

            from ui.inventory\_ui import InventoryUI

            self.inventory\_window = InventoryUI()

            self.inventory\_window.showMaximized()

        except ImportError as e:

            QMessageBox.critical(self, "Error", f"Inventory UI file not found!\n{str(e)}")

            traceback.print\_exc()

    def open\_billing(self):

        """Opens the Billing & Sales UI."""

        try:

            from ui.billing\_ui import BillingUI

            self.billing\_window = BillingUI()

            self.billing\_window.showMaximized()

        except ImportError as e:

            QMessageBox.critical(self, "Error", f"Billing UI file not found!\n{str(e)}")

            traceback.print\_exc()

    def open\_report(self):

        """Opens the Billing & Sales UI."""

        try:

            from ui.report\_ui import ReportUI

            self.report\_window = ReportUI()

            self.report\_window.showMaximized()

        except ImportError as e:

            QMessageBox.critical(self, "Error", f"report UI file not found!\n{str(e)}")

            traceback.print\_exc()

    def load\_expiry\_alerts(self):

        """Loads Expiry Alerts from MySQL Database"""

        if not self.cursor:

            print("❌ Cannot load expiry alerts: No database connection.")

            return

        try:

            today = date.today()

            query = """

            SELECT name, batch\_no, expiry\_date FROM medicines

            WHERE expiry\_date <= DATE\_ADD(%s, INTERVAL 30 DAY)

            ORDER BY expiry\_date ASC

            """

            self.cursor.execute(query, (today,))

            expiry\_data = self.cursor.fetchall()

            self.expiry\_table.setRowCount(len(expiry\_data))

            for row, (name, batch, expiry\_date) in enumerate(expiry\_data):

                self.expiry\_table.setItem(row, 0, QTableWidgetItem(name))

                self.expiry\_table.setItem(row, 1, QTableWidgetItem(batch))

                self.expiry\_table.setItem(row, 2, QTableWidgetItem(str(expiry\_date)))

            expired\_medicines = [f"{name} (Batch: {batch}, Expiry: {expiry\_date})" for name, batch, expiry\_date in expiry\_data if expiry\_date < today]

            if expired\_medicines:

                self.show\_expiry\_alert(expired\_medicines)

        except mysql.connector.Error as e:

            print(f"❌ Error fetching expiry alerts: {e}")

    def show\_expiry\_alert(self, expired\_medicines):

        """Show an alert for expired medicines"""

        alert\_message = "The following medicines are expired:\n\n" + "\n".join(expired\_medicines)

        QMessageBox.warning(self, "Expiry Alert", alert\_message)

    def closeEvent(self, event):

        """Close DB connection when window is closed"""

        if self.cursor:

            self.cursor.close()

        if self.conn:

            self.conn.close()

        print("🔒 Database connection closed.")

        event.accept()

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    window = Dashboard()

    window.showMaximized()

    sys.exit(app.exec())

**login page:**

from PyQt6.QtWidgets import (

    QApplication, QWidget, QLabel, QLineEdit, QPushButton, QVBoxLayout, QHBoxLayout, QMessageBox, QFrame

)

from PyQt6.QtGui import QFont, QPixmap

from PyQt6.QtCore import Qt, QSize

import sys

import subprocess

import os

# ✅ Credentials

CORRECT\_USERNAME = "admin"

CORRECT\_PASSWORD = "password123"

class LoginWindow(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Login - Siddhanath Medical")

        self.resize(1024, 768)

        # ✅ Main Layout

        main\_layout = QVBoxLayout()

        main\_layout.setContentsMargins(0, 0, 0, 0)

        main\_layout.setSpacing(0)

        self.setLayout(main\_layout)

        # ✅ Create Background First

        self.background\_label = QLabel(self)

        self.background\_label.setScaledContents(True)

        # ✅ Card-like Login Frame

        login\_frame = QFrame(self)

        login\_frame.setStyleSheet("""

            background-color: white;

            border-radius: 12px;

            padding: 30px;

            border: 1px solid #ccc;

        """)

        frame\_layout = QVBoxLayout()

        frame\_layout.setSpacing(15)

        login\_frame.setLayout(frame\_layout)

        # ✅ Shop Name

        shop\_label = QLabel("Siddhanath Medical")

        shop\_label.setFont(QFont("Arial", 16, QFont.Weight.Bold))

        shop\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

        shop\_label.setStyleSheet("color: #222;")

        frame\_layout.addWidget(shop\_label)

        # ✅ Owner Name

        owner\_label = QLabel("Owner: Aniket")

        owner\_label.setFont(QFont("Arial", 12))

        owner\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

        owner\_label.setStyleSheet("color: #666;")

        frame\_layout.addWidget(owner\_label)

        # ✅ Username Input

        self.username\_input = QLineEdit()

        self.username\_input.setPlaceholderText("Enter Username")

        self.username\_input.setFont(QFont("Arial", 12))

        self.username\_input.setStyleSheet(self.input\_style())

        frame\_layout.addWidget(self.username\_input)

        # ✅ Password Input

        self.password\_input = QLineEdit()

        self.password\_input.setPlaceholderText("Enter Password")

        self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)

        self.password\_input.setFont(QFont("Arial", 12))

        self.password\_input.setStyleSheet(self.input\_style())

        frame\_layout.addWidget(self.password\_input)

        # ✅ Login Button

        self.login\_button = QPushButton("Login")

        self.login\_button.setFont(QFont("Arial", 14, QFont.Weight.Bold))

        self.login\_button.setStyleSheet(self.button\_style())

        self.login\_button.clicked.connect(self.validate\_login)

        frame\_layout.addWidget(self.login\_button)

        # ✅ Add Login Frame to Main Layout

        main\_layout.addWidget(login\_frame, alignment=Qt.AlignmentFlag.AlignCenter)

        # ✅ Apply Background

        self.set\_background()

    def set\_background(self):

        """ Set a full-screen background image using QLabel. """

        background\_image = QPixmap("login.webp")  # Load background image

        if not background\_image.isNull():

            self.background\_label.setPixmap(background\_image.scaled(

                self.size(), Qt.AspectRatioMode.IgnoreAspectRatio, Qt.TransformationMode.SmoothTransformation

            ))

            self.background\_label.setGeometry(0, 0, self.width(), self.height())  # Fill screen

        # ✅ Ensure UI elements stay in front of the background

        self.background\_label.lower()

    def resizeEvent(self, event):

        """ Ensure background resizes dynamically when window is resized. """

        self.set\_background()

        super().resizeEvent(event)

    def input\_style(self):

        return """

            QLineEdit {

                border: 2px solid #007BFF;

                border-radius: 5px;

                padding: 8px;

                font-size: 14px;

                background-color: white;

            }

            QLineEdit:focus {

                border-color: #0056b3;

            }

        """

    def button\_style(self):

        return """

            QPushButton {

                background-color: #007BFF; color: white;

                border-radius: 8px; padding: 10px;

            }

            QPushButton:hover {

                background-color: #0056b3;

            }

        """

    def validate\_login(self):

        username = self.username\_input.text()

        password = self.password\_input.text()

        if username == CORRECT\_USERNAME and password == CORRECT\_PASSWORD:

            self.open\_dashboard()

        else:

            QMessageBox.warning(self, "Error", "Invalid Username or Password!")

    def open\_dashboard(self):

        self.close()

        dashboard\_path = os.path.join(os.path.dirname(\_\_file\_\_), "ui", "dashboard.py")

        if os.path.exists(dashboard\_path):

            subprocess.Popen(["python", dashboard\_path])

        else:

            QMessageBox.critical(self, "Error", "Dashboard file not found!")

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    window = LoginWindow()

    window.show()

    sys.exit(app.exec())

**main.py**

import sys

from PyQt6.QtWidgets import QApplication

from auth import LoginWindow  # Import the login window

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    login\_window = LoginWindow()  # Start with login

    login\_window.show()

    sys.exit(app.exec())  # Run event loop

**bd.py**

import mysql.connector

def connect\_db():

    try:

        conn = mysql.connector.connect(

            host="localhost",

            user="root",

            password="Radha@1474",

            database="medical\_shop"

        )

        if conn.is\_connected():

            print("✅ Connected to MySQL Database!")

        return conn

    except mysql.connector.Error as e:

        print(f"❌ Error: {e}")

        return None

# Test Connection

if \_\_name\_\_ == "\_\_main\_\_":

    db\_connection = connect\_db()

    if db\_connection:

        db\_connection.close()

**inventory\_manager.py**

import mysql.connector

from mysql.connector import Error

from datetime import datetime

from db import connect\_db  # ✅ Import the correct function

class InventoryManager:

    def \_\_init\_\_(self):

        self.conn = connect\_db()  # ✅ Use connect\_db() to establish a connection

        if self.conn:

            self.cursor = self.conn.cursor()

        else:

            self.cursor = None

            print("❌ Database connection failed.")

    def add\_medicine(self, name, category, manufacturer, batch\_no, expiry\_date, price, stock\_quantity, gst\_percentage):

        if not self.cursor:

            print("❌ Cannot add medicine: No database connection.")

            return

        query = """

        INSERT INTO medicines (name, category, manufacturer, batch\_no, expiry\_date, price, stock\_quantity, gst\_percentage)

        VALUES (%s, %s, %s, %s, %s, %s, %s, %s)

        """

        values = (name, category, manufacturer, batch\_no, expiry\_date, price, stock\_quantity, gst\_percentage)

        try:

            self.cursor.execute(query, values)

            self.conn.commit()

            print("✅ Medicine added successfully.")

        except Error as e:

            print(f"❌ Error adding medicine: {e}")

    def fetch\_all\_medicines(self):

        if not self.cursor:

            print("❌ Cannot fetch medicines: No database connection.")

            return []

        query = "SELECT \* FROM medicines"

        try:

            self.cursor.execute(query)

            return self.cursor.fetchall()

        except Error as e:

            print(f"❌ Error fetching medicines: {e}")

            return []

    def update\_medicine(self, medicine\_id, name, category, manufacturer, batch\_no, expiry\_date, price, stock\_quantity, gst\_percentage):

        if not self.cursor:

            print("❌ Cannot update medicine: No database connection.")

            return

        query = """

        UPDATE medicines

        SET name=%s, category=%s, manufacturer=%s, batch\_no=%s, expiry\_date=%s, price=%s, stock\_quantity=%s, gst\_percentage=%s

        WHERE medicine\_id=%s

        """

        values = (name, category, manufacturer, batch\_no, expiry\_date, price, stock\_quantity, gst\_percentage, medicine\_id)

        try:

            self.cursor.execute(query, values)

            self.conn.commit()

            print("✅ Medicine updated successfully.")

        except Error as e:

            print(f"❌ Error updating medicine: {e}")

    def delete\_medicine(self, medicine\_id):

        if not self.cursor:

            print("❌ Cannot delete medicine: No database connection.")

            return

        query = "DELETE FROM medicines WHERE medicine\_id=%s"

        try:

            self.cursor.execute(query, (medicine\_id,))

            self.conn.commit()

            print("✅ Medicine deleted successfully.")

        except Error as e:

            print(f"❌ Error deleting medicine: {e}")

    def close(self):

        if self.cursor:

            self.cursor.close()

        if self.conn:

            self.conn.close()

        print("🔒 Database connection closed.")

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

    inventory = InventoryManager()

    # Adding a test medicine

    inventory.add\_medicine("Paracetamol", "Painkiller", "ABC Pharma", "B123", "2025-12-31", 50.00, 100, 5.00)

    # Fetch and print medicines

    medicines = inventory.fetch\_all\_medicines()

    for medicine in medicines:

        print(medicine)

    # Update a medicine

    inventory.update\_medicine(1, "Paracetamol", "Painkiller", "XYZ Pharma", "B123", "2026-01-01", 55.00, 90, 5.00)

    # Delete a medicine

    inventory.delete\_medicine(1)

    # Close connection properly

    inventory.close()

**inventory\_ui.py**

import sys

import os

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))

from inventory\_manager import InventoryManager  # ✅ Importing from the parent folder

from PyQt6.QtWidgets import QApplication, QMainWindow, QLabel, QLineEdit, QPushButton, QVBoxLayout, QHBoxLayout, QWidget, QTableWidget, QTableWidgetItem, QMessageBox

from PyQt6.QtGui import QFont

from PyQt6.QtCore import Qt

from PyQt6.QtWidgets import QHeaderView

class InventoryUI(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Medical Shop Inventory Management")

        self.setGeometry(100, 100, 900, 500)

        self.inventory = InventoryManager()

        self.initUI()

    def initUI(self):

        # Main Layout

        main\_layout = QVBoxLayout()

        # Heading Label

        self.heading = QLabel("Inventory Management")

        self.heading.setStyleSheet("font-size: 22px; font-weight: bold;")

        main\_layout.addWidget(self.heading, alignment=Qt.AlignmentFlag.AlignCenter)

        # Form Layout

        self.entries = {}

        fields = ["Name", "Category", "Manufacturer", "Batch No", "Expiry Date (YYYY-MM-DD)", "Price", "Stock Quantity", "GST %"]

        form\_layout = QVBoxLayout()

        for field in fields:

            label = QLabel(field)

            label.setFont(QFont("Arial", 14))

            label.setFixedWidth(280)

            entry = QLineEdit()

            entry.setFont(QFont("Arial", 14))

            entry.setStyleSheet("border-radius: 8px; padding: 5px; max-width: 200px;")

            self.entries[field] = entry

            row\_layout = QHBoxLayout()

            row\_layout.setAlignment(Qt.AlignmentFlag.AlignLeft)

            row\_layout.addWidget(label)

            row\_layout.addWidget(entry)

            form\_layout.addLayout(row\_layout)

        main\_layout.addLayout(form\_layout)

        # Buttons Layout

        btn\_layout = QHBoxLayout()

        button\_style = "font-size: 15px; padding: 6px 12px; max-width: 200px;"

        self.add\_btn = QPushButton("Add Medicine")

        self.add\_btn.setStyleSheet(f"background-color: green; color: white; {button\_style}")

        self.add\_btn.clicked.connect(self.add\_medicine)

        self.update\_btn = QPushButton("Update Medicine")

        self.update\_btn.setStyleSheet(f"background-color: blue; color: white; {button\_style}")

        self.update\_btn.clicked.connect(self.update\_medicine)

        self.delete\_btn = QPushButton("Delete Medicine")

        self.delete\_btn.setStyleSheet(f"background-color: red; color: white; {button\_style}")

        self.delete\_btn.clicked.connect(self.delete\_medicine)

        self.clear\_btn = QPushButton("Clear Fields")

        self.clear\_btn.setStyleSheet(f"background-color: gray; color: white; {button\_style}")

        self.clear\_btn.clicked.connect(self.clear\_fields)

        btn\_layout.addWidget(self.add\_btn)

        btn\_layout.addWidget(self.update\_btn)

        btn\_layout.addWidget(self.delete\_btn)

        btn\_layout.addWidget(self.clear\_btn)

        main\_layout.addLayout(btn\_layout)

        # Table Layout

        self.table = QTableWidget()

        self.table.setColumnCount(9)

        self.table.setHorizontalHeaderLabels(["ID", "Name", "Category", "Manufacturer", "Batch", "Expiry", "Price", "Stock", "GST"])

        self.table.cellClicked.connect(self.select\_medicine)

        self.table.setStyleSheet("width: 100%;")

        # Make columns expand to take full width

        self.table.horizontalHeader().setStretchLastSection(True)  # Stretch last column

        self.table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode.Stretch)  # Stretch all columns

        # Remove unnecessary width restriction

        self.table.setStyleSheet("border: none;")

        main\_layout.addWidget(self.table)

        self.load\_medicines()

        # Main Widget

        container = QWidget()

        container.setLayout(main\_layout)

        self.setCentralWidget(container)

    def add\_medicine(self):

        data = self.get\_form\_data()

        if data:

            self.inventory.add\_medicine(\*data)

            self.load\_medicines()

            self.clear\_fields()

    def update\_medicine(self):

        selected\_row = self.table.currentRow()

        if selected\_row == -1:

            QMessageBox.warning(self, "Error", "Please select a medicine to update")

            return

        medicine\_id = int(self.table.item(selected\_row, 0).text())

        data = self.get\_form\_data()

        if data:

            self.inventory.update\_medicine(medicine\_id, \*data)

            self.load\_medicines()

            self.clear\_fields()

    def delete\_medicine(self):

        selected\_row = self.table.currentRow()

        if selected\_row == -1:

            QMessageBox.warning(self, "Error", "Please select a medicine to delete")

            return

        medicine\_id = int(self.table.item(selected\_row, 0).text())

        self.inventory.delete\_medicine(medicine\_id)

        self.load\_medicines()

        self.clear\_fields()

    def load\_medicines(self):

        self.table.setRowCount(0)

        medicines = self.inventory.fetch\_all\_medicines()

        for row\_num, medicine in enumerate(medicines):

            self.table.insertRow(row\_num)

            for col\_num, value in enumerate(medicine):

                self.table.setItem(row\_num, col\_num, QTableWidgetItem(str(value)))

    def select\_medicine(self, row, \_):

        for i, field in enumerate(list(self.entries.keys())):

            self.entries[field].setText(self.table.item(row, i+1).text())

    def get\_form\_data(self):

        try:

            name = self.entries["Name"].text().strip()

            category = self.entries["Category"].text().strip()

            manufacturer = self.entries["Manufacturer"].text().strip()

            batch\_no = self.entries["Batch No"].text().strip()

            expiry\_date = self.entries["Expiry Date (YYYY-MM-DD)"].text().strip()

            price = float(self.entries["Price"].text().strip())

            stock\_quantity = int(self.entries["Stock Quantity"].text().strip())

            gst\_percentage = float(self.entries["GST %"].text().strip())

            if not all([name, category, manufacturer, batch\_no, expiry\_date]):

                raise ValueError("All fields must be filled!")

            return name, category, manufacturer, batch\_no, expiry\_date, price, stock\_quantity, gst\_percentage

        except ValueError as e:

            QMessageBox.warning(self, "Input Error", str(e))

            return None

    def clear\_fields(self):

        for entry in self.entries.values():

            entry.clear()

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    window = InventoryUI()

    window.show()

    sys.exit(app.exec())

**dashboard new :**

import sys

import os

import mysql.connector

import traceback

from PyQt6.QtWidgets import QApplication, QMessageBox, QMainWindow, QWidget, QVBoxLayout, QPushButton, QLabel, QHBoxLayout, QTableWidget, QTableWidgetItem, QHeaderView

from PyQt6.QtGui import QIcon, QFont

from PyQt6.QtCore import Qt

from datetime import datetime, date  # ✅ Import date

# ✅ Import database connection function from db.py

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))

from db import connect\_db

class Dashboard(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        # ✅ Establish MySQL connection

        self.conn = connect\_db()

        if self.conn:

            self.cursor = self.conn.cursor()

            print("✅ Connected to MySQL!")

        else:

            self.cursor = None

            QMessageBox.critical(self, "Database Error", "Failed to connect to MySQL!")

        # ✅ Set up the window

        self.setWindowTitle("Medical Shop Inventory Management")

        self.setGeometry(100, 100, 1000, 600)

        self.showMaximized()

        # ✅ Set absolute icon path

        icon\_path = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "../assets/icon.ico"))

        if os.path.exists(icon\_path):

            self.setWindowIcon(QIcon(icon\_path))

        else:

            print("Warning: Icon not found at", icon\_path)

        self.setStyleSheet("background-color: #f8f9fa;")

        # ✅ Main Widget and Layout

        main\_widget = QWidget()

        main\_layout = QVBoxLayout()

        main\_widget.setLayout(main\_layout)

        self.setCentralWidget(main\_widget)

        # ✅ Title Label

        title = QLabel("Medical Shop Management System")

        title.setAlignment(Qt.AlignmentFlag.AlignCenter)

        title.setFont(QFont("Arial", 22, QFont.Weight.Bold))

        title.setStyleSheet("color: #343a40; padding: 10px;")

        main\_layout.addWidget(title)

        # ✅ Navigation Buttons

        button\_layout = QHBoxLayout()

        self.dashboard\_btn = QPushButton("Dashboard")

        self.inventory\_btn = QPushButton("Inventory Management")

        self.billing\_btn = QPushButton("Billing & Sales")

        self.reports\_btn = QPushButton("Reports & Analysis")

        for btn in [self.dashboard\_btn, self.inventory\_btn, self.billing\_btn, self.reports\_btn]:

            btn.setFixedSize(250, 50)

            btn.setFont(QFont("Arial", 14, QFont.Weight.Bold))

            btn.setStyleSheet("""

                QPushButton {

                    background-color: #007BFF;

                    color: white;

                    border-radius: 8px;

                    border: none;

                    padding: 10px;

                }

                QPushButton:hover {

                    background-color: #0056b3;

                }

            """)

            btn.setCursor(Qt.CursorShape.PointingHandCursor)

            button\_layout.addWidget(btn)

        main\_layout.addLayout(button\_layout)

        # ✅ Expiry Alert Section

        alert\_label = QLabel("\n⚠ Expiry Date Alerts:")

        alert\_label.setFont(QFont("Arial", 18, QFont.Weight.Bold))

        alert\_label.setStyleSheet("color: red;")

        main\_layout.addWidget(alert\_label)

        self.expiry\_table = QTableWidget()

        self.expiry\_table.setColumnCount(3)

        self.expiry\_table.setHorizontalHeaderLabels(["Medicine Name", "Batch No", "Expiry Date"])

        self.expiry\_table.setStyleSheet("font-size: 14px; background-color: white; border-radius: 5px;")

        # ✅ Set column width and styling

        self.expiry\_table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode.Stretch)

        self.expiry\_table.horizontalHeader().setStyleSheet("font-size: 16px; font-weight: bold; color: #212529; background-color: #e9ecef;")

        self.expiry\_table.verticalHeader().setDefaultSectionSize(30)  # Adjust row height

        self.load\_expiry\_alerts()

        main\_layout.addWidget(self.expiry\_table)

        # ✅ Connect buttons to open respective windows

        self.inventory\_btn.clicked.connect(self.open\_inventory)

    def open\_inventory(self):

        """Opens the Inventory Management UI."""

        try:

            from ui.inventory\_ui import InventoryUI

            self.inventory\_window = InventoryUI()

            self.inventory\_window.showMaximized()

        except ImportError:

            QMessageBox.critical(self, "Error", "Inventory UI file not found!")

    def load\_expiry\_alerts(self):

        """Loads Expiry Alerts from MySQL Database"""

        if not self.cursor:

            print("❌ Cannot load expiry alerts: No database connection.")

            return

        try:

            today = date.today()  # ✅ Convert today to a date object

            query = """

            SELECT name, batch\_no, expiry\_date FROM medicines

            WHERE expiry\_date <= DATE\_ADD(%s, INTERVAL 30 DAY)

            ORDER BY expiry\_date ASC

            """

            self.cursor.execute(query, (today,))

            expiry\_data = self.cursor.fetchall()

            self.expiry\_table.setRowCount(len(expiry\_data))

            for row, (name, batch, expiry) in enumerate(expiry\_data):

                # ✅ Convert expiry date from str to date for comparison

                expiry\_date = datetime.strptime(expiry, "%Y-%m-%d").date() if isinstance(expiry, str) else expiry

                self.expiry\_table.setItem(row, 0, QTableWidgetItem(name))

                self.expiry\_table.setItem(row, 1, QTableWidgetItem(batch))

                self.expiry\_table.setItem(row, 2, QTableWidgetItem(str(expiry\_date)))

            # ✅ Show Expiry Alert Popup

            expired\_medicines = [f"{name} (Batch: {batch}, Expiry: {expiry\_date})" for name, batch, expiry\_date in expiry\_data if expiry\_date < today]

            if expired\_medicines:

                self.show\_expiry\_alert(expired\_medicines)

        except mysql.connector.Error as e:

            print(f"❌ Error fetching expiry alerts: {e}")

    def show\_expiry\_alert(self, expired\_medicines):

        """Show an alert for expired medicines"""

        alert\_message = "The following medicines are expired:\n\n" + "\n".join(expired\_medicines)

        QMessageBox.warning(self, "Expiry Alert", alert\_message)

    def closeEvent(self, event):

        """Close DB connection when window is closed"""

        if self.cursor:

            self.cursor.close()

        if self.conn:

            self.conn.close()

        print("🔒 Database connection closed.")

        event.accept()

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    window = Dashboard()

    #window.show()

    window.showMaximized()  # Ensure it starts in full-screen mode

    sys.exit(app.exec())

**billing.py**

import pdfkit

import mysql.connector

from datetime import datetime

from db import connect\_db  # Ensure db.py contains connect\_db()

# Configure wkhtmltopdf path

config = pdfkit.configuration(wkhtmltopdf=r"D:\Mydownlod\wkhtmltopdf\bin\wkhtmltopdf.exe")

# Define options to prevent rendering issues

options = {

    'enable-local-file-access': '',  # Required for loading local resources

    'disable-smart-shrinking': '',   # Fix scaling issues

    'no-stop-slow-scripts': '',      # Prevents timeouts on JavaScript

    'load-error-handling': 'ignore', # Ignores errors

    'load-media-error-handling': 'ignore'  # Ignores media loading issues

}

class BillingSystem:

    def \_\_init\_\_(self):

        self.conn = connect\_db()

        self.cursor = self.conn.cursor()

    def fetch\_medicine(self, medicine\_id):

        """Fetch medicine details from the database."""

        self.cursor.execute(

            "SELECT name, price, stock\_quantity, gst\_percentage FROM medicines WHERE medicine\_id = %s",

            (medicine\_id,)

        )

        return self.cursor.fetchone()

    def generate\_invoice(self, customer\_name, items):

        """

        Generate a sales invoice and store sales data.

        items = [{'medicine\_id': 1, 'quantity': 2}]

        """

        total\_price = 0

        total\_gst = 0

        invoice\_items = []

        for item in items:

            medicine = self.fetch\_medicine(item['medicine\_id'])

            if medicine:

                name, price, stock, gst = medicine

                if item['quantity'] > stock:

                    return f"Not enough stock for {name}"

                gst\_amount = (price \* item['quantity']) \* (gst / 100)

                total\_gst += gst\_amount

                subtotal = (price \* item['quantity']) + gst\_amount

                total\_price += subtotal

                invoice\_items.append((name, item['quantity'], price, gst, subtotal))

                # Deduct from stock

                new\_stock = stock - item['quantity']

                self.cursor.execute(

                    "UPDATE medicines SET stock\_quantity = %s WHERE medicine\_id = %s",

                    (new\_stock, item['medicine\_id'])

                )

                # Insert sale record

                self.cursor.execute(

                    "INSERT INTO sales (medicine\_id, quantity, total\_price, gst\_amount) VALUES (%s, %s, %s, %s)",

                    (item['medicine\_id'], item['quantity'], subtotal, gst\_amount)

                )

        self.conn.commit()

        # Generate PDF invoice

        invoice\_path = self.create\_invoice\_pdf(customer\_name, invoice\_items, total\_price, total\_gst)

        return invoice\_path

    def create\_invoice\_pdf(self, customer\_name, items, total, gst):

        """Generate invoice PDF with wkhtmltopdf"""

        invoice\_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        invoice\_html = f"""

        <html>

        <head>

        <style>

            body {{ font-family: Arial, sans-serif; }}

            table {{ width: 100%; border-collapse: collapse; }}

            th, td {{ border: 1px solid black; padding: 8px; text-align: left; }}

            th {{ background-color: #f2f2f2; }}

        </style>

        </head>

        <body>

            <h2>Medical Shop Invoice</h2>

            <p><strong>Customer:</strong> {customer\_name}</p>

            <p><strong>Date:</strong> {invoice\_date}</p>

            <table>

                <tr><th>Medicine</th><th>Qty</th><th>Price</th><th>GST%</th><th>Subtotal</th></tr>

        """

        for item in items:

            name, qty, price, gst, subtotal = item

            invoice\_html += f"<tr><td>{name}</td><td>{qty}</td><td>{price}</td><td>{gst}%</td><td>{subtotal}</td></tr>"

        invoice\_html += f"""

            </table>

            <p><strong>GST Amount:</strong> {gst:.2f}</p>

            <p><strong>Total Amount:</strong> {total:.2f}</p>

        </body>

        </html>

        """

        # Generate the PDF

        invoice\_filename = f"invoices/invoice\_{datetime.now().strftime('%Y%m%d%H%M%S')}.pdf"

        try:

            pdfkit.from\_string(invoice\_html, invoice\_filename, configuration=config, options=options)

            print(f"✅ Invoice saved at: {invoice\_filename}")

        except Exception as e:

            print("❌ Error generating PDF:", str(e))

        return invoice\_filename

**billing\_ui.py**

import sys

import os

from PyQt6.QtWidgets import (

    QApplication, QMainWindow, QFormLayout, QWidget, QLabel, QLineEdit, QPushButton,

    QVBoxLayout, QHBoxLayout, QTableWidget, QTableWidgetItem, QMessageBox

)

from PyQt6.QtCore import Qt

from PyQt6.QtGui import QFont

from PyQt6.QtWidgets import QHeaderView

sys.path.append(r"D:\TY\miniproject\medical\_shop\venv\Lib\site-packages")

import pdfkit

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))

from billing import BillingSystem  # Import your BillingSystem class

class BillingUI(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.entries = {}  # Dictionary to store field entries

        self.setWindowTitle("Billing & Sales")

        self.setGeometry(200, 100, 800, 500)

        # Create Central Widget

        self.central\_widget = QWidget(self)

        self.setCentralWidget(self.central\_widget)

        # Main Layout

        self.main\_layout = QVBoxLayout()

        # Heading Label

        self.heading = QLabel("Generate Bill")

        self.heading.setStyleSheet("font-size: 22px; font-weight: bold;")

        self.main\_layout.addWidget(self.heading, alignment=Qt.AlignmentFlag.AlignCenter)

        # Fields for Customer Information

        customer\_fields = ["Customer Name", "Medicine ID", "Quantity"]

        form\_layout = QVBoxLayout()

        for field in customer\_fields:

            label = QLabel(field)

            label.setFont(QFont("Arial", 14))

            label.setFixedWidth(280)

            entry = QLineEdit()

            entry.setFont(QFont("Arial", 14))

            entry.setStyleSheet("border-radius: 8px; padding: 5px; max-width: 200px;")

            self.entries[field] = entry

            row\_layout = QHBoxLayout()

            row\_layout.setAlignment(Qt.AlignmentFlag.AlignLeft)

            row\_layout.addWidget(label)

            row\_layout.addWidget(entry)

            form\_layout.addLayout(row\_layout)

        self.main\_layout.addLayout(form\_layout)

        # Buttons Layout

        btn\_layout = QHBoxLayout()

        button\_style = "font-size: 15px; padding: 6px 12px; max-width: 200px;"

        self.add\_btn = QPushButton("Add Item")

        self.add\_btn.setStyleSheet(f"background-color: green; color: white; {button\_style}")

        self.add\_btn.clicked.connect(self.add\_item)

        btn\_layout.addWidget(self.add\_btn)

        self.main\_layout.addLayout(btn\_layout)

        # Table for Billing Items

        self.table = QTableWidget(0, 5)

        self.table.setHorizontalHeaderLabels(["Medicine", "Quantity", "Price", "GST%", "Subtotal"])

        # Make columns stretch to fill the entire width

        self.table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode.Stretch)

        # Add table to layout

        self.main\_layout.addWidget(self.table)

        # Generate Invoice Button

        self.generate\_invoice\_button = QPushButton("Generate Invoice")

        self.generate\_invoice\_button.setStyleSheet(f"background-color: blue; color: white; {button\_style}")

        self.generate\_invoice\_button.clicked.connect(self.generate\_invoice)

        # Centering the Button

        button\_layout = QHBoxLayout()

        button\_layout.addWidget(self.generate\_invoice\_button, alignment=Qt.AlignmentFlag.AlignCenter)

        self.main\_layout.addLayout(button\_layout)

        # Set Layout to Central Widget

        self.central\_widget.setLayout(self.main\_layout)

        # Store items in a list

        self.items = []

        self.billing\_system = BillingSystem()  # Instance of BillingSystem

    def add\_item(self):

        """Handles adding a medicine item to the bill."""

        customer\_name = self.entries["Customer Name"].text().strip()

        medicine\_id = self.entries["Medicine ID"].text().strip()

        quantity\_text = self.entries["Quantity"].text().strip()

        if not medicine\_id or not quantity\_text.isdigit():

            QMessageBox.critical(self, "Error", "Invalid input. Please enter valid data.")

            return

        medicine = self.billing\_system.fetch\_medicine(medicine\_id)

        if medicine:

            name, price, stock, gst = medicine

            quantity = int(quantity\_text)

            if quantity > stock:

                QMessageBox.critical(self, "Error", f"Not enough stock for {name}")

                return

            subtotal = (price \* quantity) + ((price \* quantity) \* gst / 100)

            row\_count = self.table.rowCount()

            self.table.insertRow(row\_count)

            self.table.setItem(row\_count, 0, QTableWidgetItem(name))

            self.table.setItem(row\_count, 1, QTableWidgetItem(str(quantity)))

            self.table.setItem(row\_count, 2, QTableWidgetItem(f"{price:.2f}"))

            self.table.setItem(row\_count, 3, QTableWidgetItem(f"{gst}%"))

            self.table.setItem(row\_count, 4, QTableWidgetItem(f"{subtotal:.2f}"))

            self.items.append({"medicine\_id": int(medicine\_id), "quantity": quantity})

        else:

            QMessageBox.critical(self, "Error", "Medicine not found.")

    def generate\_invoice(self):

        """Handles invoice generation."""

        customer\_name = self.entries["Customer Name"].text().strip()

        if not customer\_name or not self.items:

            QMessageBox.critical(self, "Error", "Customer name and items are required!")

            return

        pdfkit\_config = pdfkit.configuration(wkhtmltopdf="D:/Mydownlod/wkhtmltopdf/bin/wkhtmltopdf.exe")

        invoice\_path = self.billing\_system.generate\_invoice(customer\_name, self.items)

        QMessageBox.information(self, "Invoice Generated", f"Invoice saved at {invoice\_path}")

# Run UI independently

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    window = BillingUI()

    window.show()

    sys.exit(app.exec())

report\_ui.py

import sys

import os

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))

from reports import Reports

from PyQt6.QtWidgets import (

    QApplication, QWidget, QVBoxLayout, QHBoxLayout,

    QPushButton, QTableWidget, QTableWidgetItem,QGroupBox,

    QLabel, QComboBox, QDateEdit, QMessageBox, QLineEdit,QGridLayout

)

from PyQt6.QtCore import QDate

import matplotlib.pyplot as plt

from fpdf import FPDF

class ReportUI(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Medical Shop Reports")

        self.setGeometry(100, 100, 1000, 600)

        self.setStyleSheet("""

    QWidget {

        background-color: #f5f7fa;

        font-family: Segoe UI, sans-serif;

        font-size: 14px;

    }

    QGroupBox {

        border: 2px solid #444;

        border-radius: 10px;

        margin-top: 10px;

        padding: 15px;

        background-color: #ffffff;

    }

    QGroupBox:title {

        subcontrol-origin: margin;

        subcontrol-position: top left;

        padding: 0 3px;

        font-weight: bold;

        color: #333;

    }

    QLabel {

        font-weight: 600;

        color: #222;

    }

    QPushButton {

        background-color: #007BFF;

        color: white;

        border: none;

        border-radius: 20px;

        padding: 8px 16px;

    }

    QPushButton:hover {

        background-color: #0056b3;

    }

    QPushButton:pressed {

        background-color: #3e8e41;

    }

    QComboBox, QLineEdit, QDateEdit {

        border: 2px solid #999;

        border-radius: 10px;

        padding: 6px;

        background-color: white;

    }

    QTableWidget {

        border: 1px solid #ccc;

        border-radius: 10px;

        background-color: #fff;

        gridline-color: #aaa;

    }

    QHeaderView::section {

        font-weight: bold;

        padding: 4px;

        border: none;

    }

    QTableWidget QTableCornerButton::section {

        background-color: ;

    }

    QLineEdit:focus, QComboBox:focus, QDateEdit:focus {

        border: 2px solid;

    }

""")

        self.report\_manager = Reports()

        layout = QVBoxLayout()

        control\_layout = QHBoxLayout()

         # === Top Control Section ===

        top\_group = QGroupBox("Report Options")

        top\_layout = QGridLayout()

        self.report\_type = QComboBox()

        self.report\_type.addItems([

            "Daily Sales Report", "Monthly Sales Report", "Yearly Sales Report",

            "Product-wise Sales Report", "Vendor-wise Sales Report",

            "GST Summary Report", "GST Detailed Report", "GST Liability Report",

            "Profit Summary Report", "Loss Report", "High-Profit Products Report",

            "Expired Products Report", "Near Expiry Report",

            "Current Stock Report", "Low Stock Report", "Out of Stock Report",

            "Purchase Report"

        ])

        top\_layout.addWidget(QLabel("Report Type:"),0,0)

        top\_layout.addWidget(self.report\_type,0,1)

        self.date\_input = QDateEdit()

        self.date\_input.setDate(QDate.currentDate())

        self.date\_input.setCalendarPopup(True)

        top\_layout.addWidget(QLabel("              Date:"),0,2)

        top\_layout.addWidget(self.date\_input,0,3)

        self.view\_btn = QPushButton("View Report")

        self.view\_btn.clicked.connect(self.view\_report)

        top\_layout.addWidget(self.view\_btn,0,4)

        top\_group.setLayout(top\_layout)

        layout.addWidget(top\_group)

        self.table = QTableWidget()

        layout.addWidget(self.table)

        self.setLayout(layout)

        self.current\_report\_data = []

        self.current\_report\_name = ""

        bottom\_group = QGroupBox("Actions")

        bottom\_layout = QHBoxLayout()

        self.export\_btn = QPushButton("Export to Excel")

        self.export\_btn.clicked.connect(self.export\_excel)

        bottom\_layout.addWidget(self.export\_btn)

        self.export\_pdf\_btn = QPushButton("Export to PDF")

        self.export\_pdf\_btn.clicked.connect(self.export\_pdf)

        bottom\_layout.addWidget(self.export\_pdf\_btn)

        self.email\_input = QLineEdit()

        self.email\_input.setPlaceholderText("Enter email to send report")

        bottom\_layout.addWidget(self.email\_input)

        self.send\_email\_btn = QPushButton("Send Email")

        self.send\_email\_btn.clicked.connect(self.send\_email)

        bottom\_layout.addWidget(self.send\_email\_btn)

        self.gst\_graph\_btn = QPushButton("Show Monthly GST Graph")

        self.gst\_graph\_btn.clicked.connect(self.plot\_monthly\_gst)

        bottom\_layout.addWidget(self.gst\_graph\_btn)

        layout.addLayout(bottom\_layout)

        bottom\_group.setLayout(bottom\_layout)

        layout.addWidget(bottom\_group)

        self.setLayout(layout)

        self.current\_report\_data = []

        self.current\_report\_name = ""

    def view\_report(self):

        report\_name = self.report\_type.currentText()

        date = self.date\_input.date().toString("yyyy-MM-dd")

        self.current\_report\_name = report\_name.replace(" ", "\_")

        try:

            data = []

            match report\_name:

                case "Daily Sales Report":

                    data = self.report\_manager.daily\_sales\_report(date)

                case "Monthly Sales Report":

                    year, month = date.split("-")[0], date.split("-")[1]

                    data = self.report\_manager.monthly\_sales\_report(year, month)

                case "Yearly Sales Report":

                    year = date.split("-")[0]

                    data = self.report\_manager.yearly\_sales\_report(year)

                case "Product-wise Sales Report":

                    data = self.report\_manager.product\_wise\_sales\_report()

                case "Vendor-wise Sales Report":

                    data = self.report\_manager.vendor\_wise\_sales\_report()

                case "GST Summary Report":

                    data = self.report\_manager.gst\_summary\_report()

                case "GST Detailed Report":

                    # Use last 30 days

                    today = QDate.currentDate()

                    start = today.addDays(-30).toString("yyyy-MM-dd")

                    end = today.toString("yyyy-MM-dd")

                    data = self.report\_manager.gst\_detailed\_report\_filtered(start, end)

                case "GST Liability Report":

                    start = end = date

                    data = self.report\_manager.gst\_liability\_report(start, end)

                case "Profit Summary Report":

                    data = self.report\_manager.profit\_summary\_report()

                case "Loss Report":

                    data = self.report\_manager.loss\_report()

                case "High-Profit Products Report":

                    data = self.report\_manager.high\_profit\_products\_report()

                case "Expired Products Report":

                    data = self.report\_manager.expired\_products\_report()

                case "Near Expiry Report":

                    data = self.report\_manager.near\_expiry\_report()

                case "Current Stock Report":

                    data = self.report\_manager.current\_stock\_report()

                case "Low Stock Report":

                    data = self.report\_manager.low\_stock\_report()

                case "Out of Stock Report":

                    data = self.report\_manager.out\_of\_stock\_report()

                case "Purchase Report":

                    data = self.report\_manager.purchase\_report()

            self.current\_report\_data = data

            self.show\_table(data)

        except Exception as e:

            QMessageBox.critical(self, "Error", str(e))

    def show\_table(self, data):

        self.table.clear()

        if not data:

            self.table.setRowCount(0)

            self.table.setColumnCount(0)

            return

        headers = list(data[0].keys())

        self.table.setColumnCount(len(headers))

        self.table.setHorizontalHeaderLabels(headers)

        self.table.setRowCount(len(data))

        for row\_idx, row\_data in enumerate(data):

            for col\_idx, header in enumerate(headers):

                self.table.setItem(row\_idx, col\_idx, QTableWidgetItem(str(row\_data[header])))

    def export\_excel(self):

        if not self.current\_report\_data:

            QMessageBox.warning(self, "Warning", "No data to export.")

            return

        try:

            self.report\_manager.export\_to\_excel(self.current\_report\_data, self.current\_report\_name, format='excel')

            QMessageBox.information(self, "Success", f"Exported to reports/{self.current\_report\_name}.xlsx")

        except Exception as e:

            QMessageBox.critical(self, "Error", str(e))

    def send\_email(self):

        email = self.email\_input.text().strip()

        if not email:

            QMessageBox.warning(self, "Missing Email", "Please enter an email address.")

            return

        try:

            file\_path = f'reports/{self.current\_report\_name}.xlsx'

            if not os.path.exists(file\_path):

                self.export\_excel()

            subject = f"Medical Shop Report - {self.current\_report\_name}"

            body = f"Attached is the report: {self.current\_report\_name}"

            self.report\_manager.send\_email\_with\_report(email, subject, body, file\_path)

            QMessageBox.information(self, "Success", f"Email sent to {email}")

        except Exception as e:

            QMessageBox.critical(self, "Error", str(e))

    def export\_pdf(self):

        if not self.current\_report\_data:

            QMessageBox.warning(self, "Warning", "No data to export.")

            return

        try:

            self.report\_manager.export\_to\_excel(self.current\_report\_data, self.current\_report\_name, format='pdf')

            QMessageBox.information(self, "Success", f"Exported to reports/{self.current\_report\_name}.pdf")

        except Exception as e:

            QMessageBox.critical(self, "Error", str(e))

    def plot\_monthly\_gst(self):

        try:

            data = self.report\_manager.gst\_liability\_report()

            if not data:

                QMessageBox.warning(self, "No Data", "No GST data available to plot.")

                return

            months = [f"{row['month']}/{row['year']}" for row in data]

            gst = [row['total\_gst\_collected'] for row in data]

            plt.bar(months, gst, color='green')

            plt.xlabel("Month")

            plt.ylabel("Total GST Collected")

            plt.title("Monthly GST Liability")

            plt.xticks(rotation=45)

            plt.tight\_layout()

            plt.show()

        except Exception as e:

            QMessageBox.critical(self, "Error", str(e))

if \_\_name\_\_ == '\_\_main\_\_':

    app = QApplication(sys.argv)

    window = ReportUI()

    window.show()

    sys.exit(app.exec())

reports.py

import mysql.connector

from db import connect\_db

import pandas as pd

import smtplib

from email.message import EmailMessage

import os

from fpdf import FPDF

class Reports:

    def \_\_init\_\_(self):

        self.conn = connect\_db()

        self.cursor = self.conn.cursor(dictionary=True)

    def daily\_sales\_report(self, date):

        try:

            query = """

                SELECT s.sale\_id, m.name AS medicine\_name, s.quantity, s.total\_price, s.gst\_amount, s.sale\_date

                FROM sales s

                JOIN medicines m ON s.medicine\_id = m.medicine\_id

                WHERE DATE(s.sale\_date) = %s

            """

            self.cursor.execute(query, (date,))

            return self.cursor.fetchall()

        except Exception as e:

            print(f"❌ Error fetching report: {e}")

            return []

    def monthly\_sales\_report(self, year, month):

        try:

            query = """

                SELECT m.name AS medicine\_name, SUM(s.quantity) AS total\_quantity, SUM(s.total\_price) AS total\_price

                FROM sales s

                JOIN medicines m ON s.medicine\_id = m.medicine\_id

                WHERE YEAR(s.sale\_date) = %s AND MONTH(s.sale\_date) = %s

                GROUP BY m.name

            """

            self.cursor.execute(query, (year, month))

            return self.cursor.fetchall()

        except Exception as e:

            print(f"❌ Error fetching report: {e}")

            return []

    def yearly\_sales\_report(self, year):

        try:

            query = """

                SELECT m.name AS medicine\_name, SUM(s.quantity) AS total\_quantity, SUM(s.total\_price) AS total\_price

                FROM sales s

                JOIN medicines m ON s.medicine\_id = m.medicine\_id

                WHERE YEAR(s.sale\_date) = %s

                GROUP BY m.name

            """

            self.cursor.execute(query, (year,))

            return self.cursor.fetchall()

        except Exception as e:

            print(f"❌ Error fetching report: {e}")

            return []

    def gst\_summary\_report(self):

        try:

            query = """

                SELECT m.gst\_percentage, SUM(s.total\_price) AS total\_sales, SUM(s.gst\_amount) AS total\_gst

                FROM sales s

                JOIN medicines m ON s.medicine\_id = m.medicine\_id

                GROUP BY m.gst\_percentage

            """

            self.cursor.execute(query)

            return self.cursor.fetchall()

        except Exception as e:

            print(f"❌ Error fetching report: {e}")

            return []

    def expired\_products\_report(self):

        try:

            query = """

                SELECT name, batch\_no, expiry\_date

                FROM medicines

                WHERE expiry\_date < CURDATE()

            """

            self.cursor.execute(query)

            return self.cursor.fetchall()

        except Exception as e:

            print(f"❌ Error fetching report: {e}")

            return []

    def near\_expiry\_report(self):

        try:

            query = """

                SELECT name, batch\_no, expiry\_date

                FROM medicines

                WHERE expiry\_date BETWEEN CURDATE() AND DATE\_ADD(CURDATE(), INTERVAL 30 DAY)

            """

            self.cursor.execute(query)

            return self.cursor.fetchall()

        except Exception as e:

            print(f"❌ Error fetching report: {e}")

            return []

    def low\_stock\_report(self):

        try:

            query = """

                SELECT name, stock\_quantity

                FROM medicines

                WHERE stock\_quantity <= 10

            """

            self.cursor.execute(query)

            return self.cursor.fetchall()

        except Exception as e:

            print(f"❌ Error fetching report: {e}")

            return []

    def current\_stock\_report(self):

        try:

            query = "SELECT name, stock\_quantity FROM medicines"

            self.cursor.execute(query)

            return self.cursor.fetchall()

        except Exception as e:

            print(f"❌ Error fetching report: {e}")

            return []

    def purchase\_report(self):

        try:

            query = """

                SELECT p.purchase\_id, v.name AS vendor\_name, m.name AS medicine\_name,

                       p.quantity, p.purchase\_price, p.purchase\_date

                FROM purchases p

                JOIN vendors v ON p.vendor\_id = v.vendor\_id

                JOIN medicines m ON p.medicine\_id = m.medicine\_id

            """

            self.cursor.execute(query)

            return self.cursor.fetchall()

        except Exception as e:

            print(f"❌ Error fetching report: {e}")

            return []

    #########################################################################

    def out\_of\_stock\_report(self):

        query = """

        SELECT name, category, manufacturer, batch\_no, expiry\_date

        FROM medicines

        WHERE stock\_quantity = 0

        """

        self.cursor.execute(query)

        rows = self.cursor.fetchall()

        return [dict(row) for row in rows]

    def high\_profit\_products\_report(self):

        query = """

        SELECT

            m.name,

            AVG(s.total\_price / s.quantity) AS avg\_selling\_price,

            AVG(p.purchase\_price) AS avg\_purchase\_price,

            (AVG(s.total\_price / s.quantity) - AVG(p.purchase\_price)) AS avg\_profit

        FROM sales s

        JOIN medicines m ON s.medicine\_id = m.medicine\_id

        JOIN purchases p ON p.medicine\_id = m.medicine\_id

        GROUP BY m.medicine\_id

        HAVING avg\_profit > 10  -- Example threshold

        ORDER BY avg\_profit DESC

        """

        self.cursor.execute(query)

        rows = self.cursor.fetchall()

        return [dict(row) for row in rows]

    def product\_wise\_sales\_report(self):

        query = """

        SELECT

            m.name,

            SUM(s.quantity) AS total\_quantity\_sold,

            SUM(s.total\_price) AS total\_sales\_amount

        FROM sales s

        JOIN medicines m ON s.medicine\_id = m.medicine\_id

        GROUP BY m.medicine\_id

        ORDER BY total\_sales\_amount DESC

        """

        self.cursor.execute(query)

        rows = self.cursor.fetchall()

        return [dict(row) for row in rows]

    def vendor\_wise\_sales\_report(self):

        query = """

        SELECT

            v.name AS vendor\_name,

            SUM(s.total\_price) AS total\_sales

        FROM sales s

        JOIN medicines m ON s.medicine\_id = m.medicine\_id

        JOIN vendors v ON m.vendor\_id = v.vendor\_id

        GROUP BY v.vendor\_id

        ORDER BY total\_sales DESC

        """

        self.cursor.execute(query)

        rows = self.cursor.fetchall()

        return [dict(row) for row in rows]

    def gst\_detailed\_report(self):

        query = """

        SELECT

            s.sale\_id,

            m.name AS medicine\_name,

            s.sale\_date,

            s.total\_price,

            m.gst\_percentage,

            ROUND(s.total\_price \* (m.gst\_percentage / (100 + m.gst\_percentage)), 2) AS gst\_amount

        FROM sales s

        JOIN medicines m ON s.medicine\_id = m.medicine\_id

        ORDER BY s.sale\_date DESC

        """

        self.cursor.execute(query)

        rows = self.cursor.fetchall()

        return [dict(row) for row in rows]

    def gst\_liability\_report(self, start\_date=None, end\_date=None):

        base\_query = """

            SELECT

                MONTH(s.sale\_date) AS month,

                YEAR(s.sale\_date) AS year,

                SUM(ROUND(s.total\_price \* (m.gst\_percentage / (100 + m.gst\_percentage)), 2)) AS total\_gst\_collected

            FROM sales s

            JOIN medicines m ON s.medicine\_id = m.medicine\_id

        """

        params = []

        if start\_date and end\_date:

            base\_query += " WHERE s.sale\_date BETWEEN %s AND %s"

            params.extend([start\_date, end\_date])

        base\_query += """

            GROUP BY YEAR(s.sale\_date), MONTH(s.sale\_date)

            ORDER BY year DESC, month DESC

        """

        self.cursor.execute(base\_query, params)

        rows = self.cursor.fetchall()

        return [dict(row) for row in rows]

    def profit\_summary\_report(self):

        query = """

        SELECT

            m.name AS medicine\_name,

            SUM(s.quantity) AS total\_quantity\_sold,

            AVG(s.total\_price / s.quantity) AS avg\_selling\_price,

            AVG(p.purchase\_price) AS avg\_purchase\_price,

            (AVG(s.total\_price / s.quantity) - AVG(p.purchase\_price)) \* SUM(s.quantity) AS total\_profit

        FROM sales s

        JOIN medicines m ON s.medicine\_id = m.medicine\_id

        JOIN purchases p ON p.medicine\_id = m.medicine\_id

        GROUP BY m.medicine\_id

        ORDER BY total\_profit DESC

        """

        self.cursor.execute(query)

        rows = self.cursor.fetchall()

        return [dict(row) for row in rows]

    def loss\_report(self):

        query = """

        SELECT

            m.name AS medicine\_name,

            SUM(s.quantity) AS total\_quantity\_sold,

            AVG(s.total\_price / s.quantity) AS avg\_selling\_price,

            AVG(p.purchase\_price) AS avg\_purchase\_price,

            (AVG(s.total\_price / s.quantity) - AVG(p.purchase\_price)) \* SUM(s.quantity) AS total\_loss

        FROM sales s

        JOIN medicines m ON s.medicine\_id = m.medicine\_id

        JOIN purchases p ON p.medicine\_id = m.medicine\_id

        GROUP BY m.medicine\_id

        HAVING total\_loss < 0

        ORDER BY total\_loss ASC

        """

        self.cursor.execute(query)

        rows = self.cursor.fetchall()

        return [dict(row) for row in rows]

    def export\_to\_excel(self, data, report\_name, format='excel'):

        if not data:

            print("⚠️ No data to export.")

            return

        df = pd.DataFrame(data)

        reports\_dir = 'reports'

        os.makedirs(reports\_dir, exist\_ok=True)

        if format == 'excel':

            file\_path = os.path.join(reports\_dir, f"{report\_name}.xlsx")

            df.to\_excel(file\_path, index=False)

            print(f"✅ Excel report exported to {file\_path}")

        elif format == 'pdf':

            file\_path = os.path.join(reports\_dir, f"{report\_name}.pdf")

            self.\_export\_to\_pdf(df, file\_path)

            print(f"✅ PDF report exported to {file\_path}")

        else:

            print("❌ Unsupported format. Choose 'excel' or 'pdf'.")

            return

        return file\_path

    def \_export\_to\_pdf(self, df, file\_path):

        pdf = FPDF()

        pdf.add\_page()

        pdf.set\_font("Arial", size=10)

        col\_width = pdf.w / (len(df.columns) + 1)

        # Header

        for col in df.columns:

            pdf.cell(col\_width, 10, str(col), border=1)

        pdf.ln()

        # Rows

        for \_, row in df.iterrows():

            for item in row:

                pdf.cell(col\_width, 10, str(item), border=1)

            pdf.ln()

        pdf.output(file\_path)

    def send\_email\_with\_report(self, to\_email, subject, body, attachment\_path):

        msg = EmailMessage()

        msg['Subject'] = subject

        msg['From'] = 'beactive1474@gmail.com'

        msg['To'] = to\_email

        msg.set\_content(body)

        # Attach file

        with open(attachment\_path, 'rb') as f:

            file\_data = f.read()

            file\_name = os.path.basename(attachment\_path)

        msg.add\_attachment(file\_data, maintype='application', subtype='octet-stream', filename=file\_name)

        # Send using Gmail SMTP (you must enable 'less secure apps' or use App Password)

        with smtplib.SMTP\_SSL('smtp.gmail.com', 465) as smtp:

            smtp.login('beactive1474@gmail.com', 'zrtr smiw bfvd droc')  # Use an App Password!

            smtp.send\_message(msg)

        print(f"📧 Email sent to {to\_email}")

    def gst\_detailed\_report\_filtered(self, start\_date, end\_date):

        query = """

        SELECT

            s.sale\_id,

            m.name AS medicine\_name,

            s.sale\_date,

            s.total\_price,

            m.gst\_percentage,

            ROUND(s.total\_price \* (m.gst\_percentage / (100 + m.gst\_percentage)), 2) AS gst\_amount

        FROM sales s

        JOIN medicines m ON s.medicine\_id = m.medicine\_id

        WHERE s.sale\_date BETWEEN %s AND %s

        ORDER BY s.sale\_date DESC

        """

        self.cursor.execute(query, (start\_date, end\_date))

        rows = self.cursor.fetchall()

        return [dict(row) for row in rows]

    def close\_connection(self):

        if self.conn:

            self.cursor.close()

            self.conn.close()