VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"Jnanasangama", Belagavi-590018, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY
K.R. Road, V.V.Puram, Bangalore-560 004

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DATABASE MANAGEMENT SYSTEM

BCS403

"MEDICAL SUPPLY MANAGEMENT SYSTEM"

Submitted By

NAME	USN
PRAJWAL R	1BI23CS149
RACHANA N	1BI23CS162
SHRADDHA C S	1BI23CS196

for the academic year 2024-25

Department of Computer Science & Engineering
Bangalore Institute of Technology
K.R. Road, V.V.Puram, Bangalore-560 00

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"Jnanasangama", Belagavi-590018, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY
K.R. Road, V.V.Puram, Bengaluru-560 004

Department of Computer Science & Engineering

Certificate

This is to certify that the implementation of **DBMS** is entitled

"----------" has been successfully completed by

USN: ……. **NAME: ……**

of IV semester B.E. for the partial fulfillment of the requirements for the bachelor's degree in computer science & engineering of the Visvesvaraya Technological University during the academic year 2024-2025.

Faculty In charge:
Prof. Nethravathy. V
Assistant Professor
Department of CS&E
Bangalore Institute of Technology
Bangalore

CONTENT

CHAPTER 1
INTRODUCTION

The Medical Supply Management System is a database-driven mini project designed to streamline the inventory and supply chain processes within a medical store or healthcare setting. Built as part of a Database Management Systems (DBMS) course project, the system focuses on efficiently managing essential components like medicines, suppliers, customers, billing, and orders. The project utilizes MySQL as the backend and Python (with Tkinter) as the frontend interface, offering a simple yet effective GUI for user interactions. It aims to reduce manual effort and human error while enhancing real-time inventory tracking and order handling capabilities.

1.1 FUNCTIONALITIES

The Medical Supply Management System offers a comprehensive set of features tailored to streamline operations in a medical store or pharmacy. At the core of the system is the Medicine Management module, which allows users to add, update, delete, and search for medicines. Each medicine entry includes essential details such as batch number, expiry date, quantity, and price, enabling precise inventory tracking and minimizing the risk of dispensing expired products.

Another key component is Supplier Management, which maintains detailed records of suppliers, including their contact information and the specific medicines they provide. This linkage between medicines and suppliers simplifies order placement and restocking processes. Complementing this is the Customer Management feature, which stores customer details and supports billing-related operations. This module helps in building a customer database that can be useful for tracking purchase history and generating invoices.

The system also integrates a functional Billing System, capable of generating invoices for customer purchases. It calculates total costs based on selected medicines and their quantities, ensuring accurate and consistent billing. Alongside billing, the Order Management module handles the placement and tracking of orders, keeping records of incoming stock and facilitating the management of stock levels.

To enhance usability, the application includes robust Search and Filter capabilities. Users can quickly locate specific medicines using parameters like name, ID, or expiry date. Finally, all these operations are accessible through a clean and intuitive Graphical User Interface (GUI) developed using Tkinter. This user-friendly interface ensures that even those with minimal technical experience can navigate and operate the system efficiently.

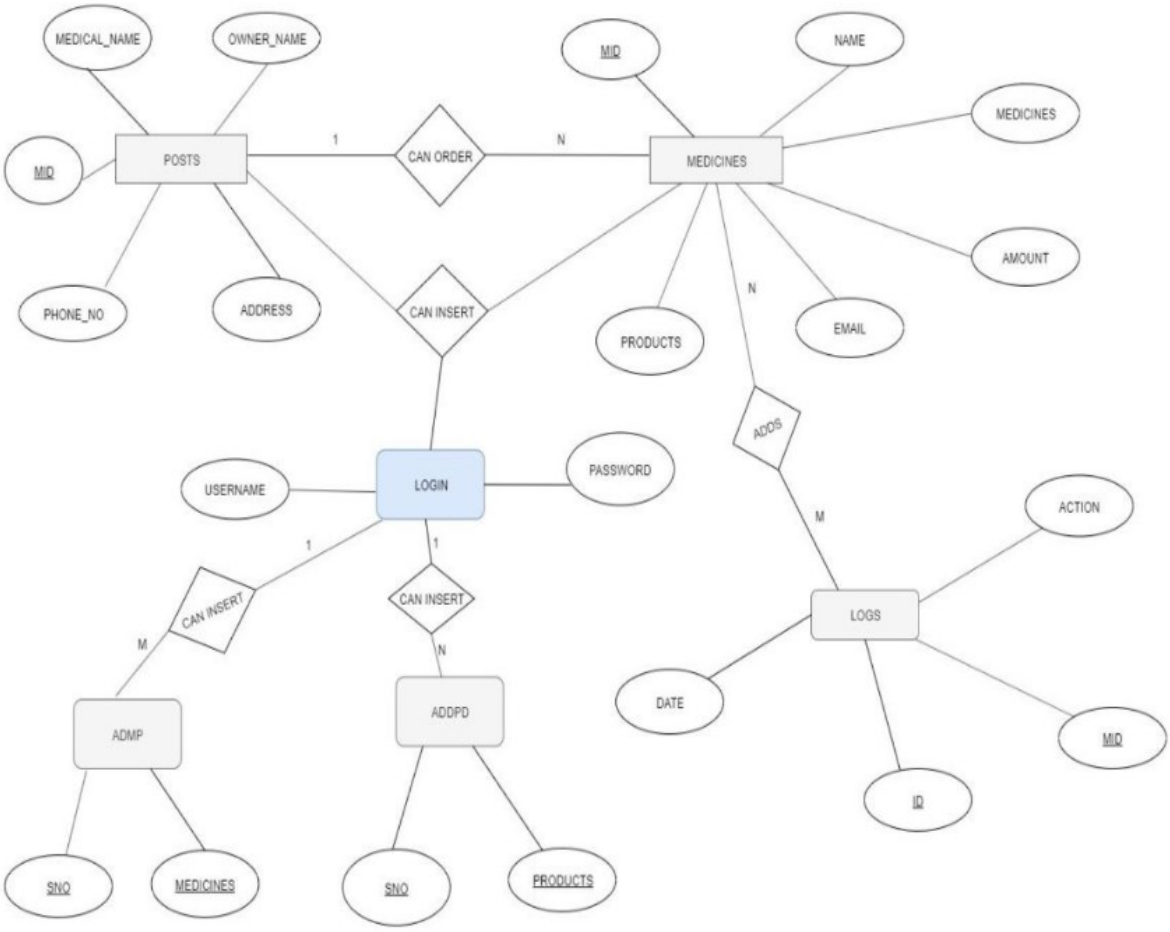
1.2 PROBLEM STATEMENT

In many small to mid-sized pharmacies and medical stores, inventory and supply chain management are often handled manually or through outdated, inefficient systems. This can lead to critical issues such as stock mismanagement, expired medicine sales, billing errors, and delayed restocking etc. Additionally, maintaining accurate records of suppliers, customers, and transactions becomes increasingly difficult without a centralized, automated solution. To address these challenges, there is a need for a user-friendly and efficient system that can manage medicine inventories, automate billing, track supplier and customer data, and streamline order processes. The Medical Supply Management System aims to fulfil this need by providing a database-driven software solution that enhances operational efficiency, minimizes human error, and ensures timely and accurate handling of medical supplies.

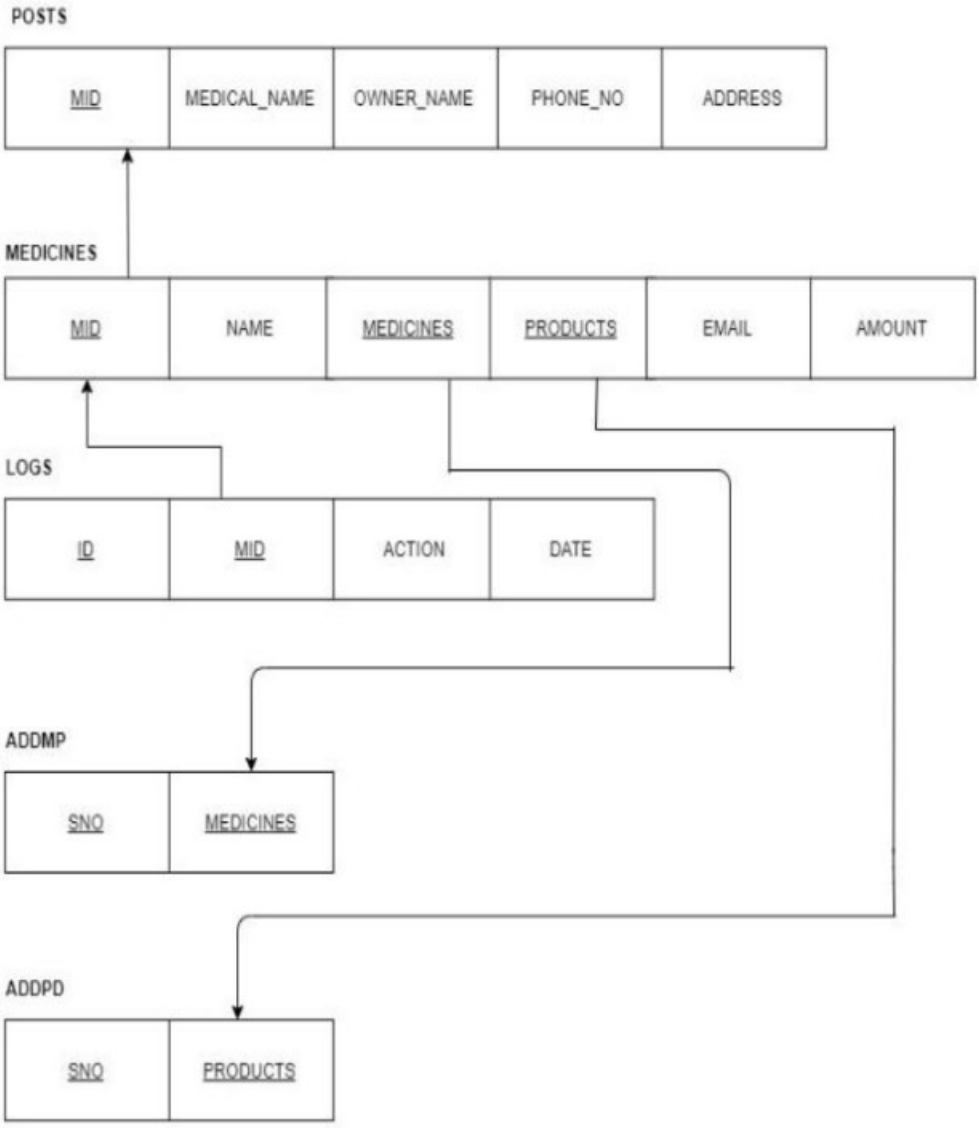
CHAPTER 2

BACKEND DESIGN

2.1 CONCEPTUAL DATABASE DESIGN (ER-DIAGRAM)



2.2 LOGICAL DATABASE DESIGN (ER-MAPPING)



2.3 NORMALIZATION UP TO 3NF

2.3.1 FIRST NORMAL FORM (1NF)

Eliminating multivalued attributes

Rule: Each column should contain atomic (indivisible) values, and each record should be unique.

Applied in the system:

- All tables have atomic values. For example, in the medicine table, fields like medicine_name, category, expiry_date, and quantity are stored in separate columns.
- No multi-valued attributes (e.g., no column contains multiple suppliers or medicine types in one field).
- Each row has a primary key, ensuring uniqueness (e.g., medicine_id in medicine, supplier_id in supplier).

Justification: Ensures that data is structured and easily searchable. Atomic values reduce ambiguity and support indexing**.**

2.3.2 SECOND NORMAL FORM (2NF)

Eliminate Partial Dependencies

Rule: All non-key attributes must be fully functionally dependent on the entire primary key**.**

Applied in the system:

- In tables where composite primary keys could exist (e.g., order_items with order_id and medicine_id), non-key attributes like quantity_ordered depend on the full composite key, not just part of it.
- Tables like supplier, medicine, and customer use single-column primary keys, and all other columns are dependent on those keys.

Justification: Prevents redundancy by ensuring that data not relevant to a part of a composite key is placed in its own table. For example, supplier contact info is stored only in the supplier table, not repeatedly in medicine.

2.3.3 THIRD NORMAL FORM (3NF)

Eliminate Transitive Dependencies

Rule: No non-key attribute should completely depend on another non-key attribute.

Applied in the system:

- In the customer table, fields like email and phone depend only on customer_id, not on other fields like customer_name.
- In the medicine table, details like price, expiry_date, and category depend directly on medicine_id, not on each other.
- Any attributes that were indirectly dependent have been separated. For example, supplier_name is not stored in the medicine table—only supplier_id is, and the name is fetched via a foreign key.

Justification: This avoids duplication and inconsistencies. For instance, if a supplier changes their name, updating it in the supplier table alone is sufficient.

2.4 LANGUAGES USED FOR BACKEND DESIGN

Frontend- HTML, CSS, Java Script, Bootstrap

Backend-Python flask (Python 3.7), SQLAlchemy

Database Connectivity- MySQL connector for Python

2.5 MODULES & CONFIGURATION

Tkinter: Used to create the graphical user interface (GUI) for user interaction. It provides buttons, forms, and input fields for all system operations.

MySQL Connector (mysql.connector): Facilitates the connection between the Python frontend and the MySQL database backend, enabling data retrieval and updates.

MySQL: Serves as the backend relational database to store and manage data related to medicines, suppliers, customers, and transactions.

Datetime: Used to handle date-related operations such as tracking expiry dates and billing timestamps.

2.6 DATABASE MODELS

Medicine: Stores information about each medicine.

Supplier: Holds details about medicine suppliers.

Customer: Contains basic customer information for billing and tracking.

Bill: Records billing transactions for customer purchases.

Bill_items: Tracks items associated with each bill.

Orders: Manages stock restocking and orders from suppliers.

2.7 ROUTES & THEIR FUNCTION

1. Home/Main Menu – Navigation hub to access all modules.
2. Medicine Management – Add, update, delete, and search medicines.
3. Supplier Management – Manage supplier details and link them to medicines.
4. Customer Management – Add or view customer info for billing.
5. Billing – Select medicines, calculate totals, and generate bills.
6. Management – Place and track stock orders from suppliers.
7. Search – Built-in search in each module for quick record access.

CHAPTER 3
FRONT-END DESIGN

3.1 MEDICAL MANAGEMENT SUPPLY SYSTEM – WEB PAGE & FORM LAYOUT DESIGN

3.1.1 MAIN WINDOW

A simple dashboard with buttons to navigate to:

- Medicines
- Suppliers
- Customers
- Billing
- Orders

3.1.2 FORMS LAYOUT

- Medicine/Supplier/Customer Forms:
  - Input fields (Entry widgets) arranged vertically or in a grid for attributes like name, ID, quantity, etc.
  - Buttons: Add, Update, Delete, Search, Clear
  - Table/Listbox below or beside to display existing records.

3.1.3 BILLING INTERFACE

- Drop-downs or entry fields to select customer and medicines.
- Quantity input and auto-price calculation.
- Final total display with a Generate Bill button.

3.1.4 ORDER FORM

- Select supplier and medicine.
- Enter quantity and order date.

- Submit button to place order and update stock.

3.2 OVERVIEW

The Medical Supply Management System is a DBMS-based mini project developed using Python (Tkinter) for the GUI and MySQL for backend data storage. It is designed to help pharmacies and medical stores manage their day-to-day operations like inventory tracking, supplier coordination, customer handling, billing, and order processing efficiently. The system reduces manual work, minimizes errors, and ensures accurate and organized record-keeping.

Key modules include Medicine Management, Supplier & Customer Management, Billing, and Order Tracking. The user interface is simple and intuitive, enabling even non-technical users to navigate and operate the system with ease. This project serves as a practical application of database concepts in a real-world domain.

3.2.1 TECHNOLOGY STACK AND CONNECTIVITY

FRONTEND (Tkinter GUI):

- **User Interaction**: Provides forms and buttons for users to input medicine details, supplier information, customer data, and orders.
- **Data Submission**: Captures the user's input and validates it before sending SQL commands to the backend.
- **Displaying Information**: Shows search results, current inventory, billing details, and order statuses in tables or list views.
- **Workflow Navigation**: Manages navigation across different modules (Medicine, Supplier, Customer, Billing, Orders) through menus or buttons.

BACKEND (MySQL Database):

- **Data Storage**: Maintains persistent storage of all medical supplies, supplier info, customer records, billing transactions, and orders.

- **Data Integrity**: Enforces rules like primary keys, foreign keys, and constraints to maintain accurate and consistent data.
- **Query Processing**: Executes queries sent from the frontend — whether for adding new records, updating stock levels, retrieving billing history, or tracking orders.
- **Transaction Management**: Handles complex operations like billing calculations and stock updates atomically to avoid data inconsistencies.
- Data is structured in tables with relationships

CHAPTER 4
IMPLEMENTATION

4.1 FRONTEND TOOLS

1. **Tkinter**
    - Provides the framework for creating windows, forms, buttons, and other GUI elements.
    - Manages user interactions and event-driven programming for smooth app navigation.
2. **Tkinter Entry Widget**
    - Styles the HTML elements to improve visual design and responsiveness.
    - Used to create layouts, align elements, and ensure mobile-friendly pages.
3. **Tkinter Button Widget**
    - Triggers actions such as adding, updating, deleting records, or searching data.
    - Links GUI events to backend functions and database operations.
4. **Tkinter Treeview Widget**
    - Displays data in tabular format, like lists of medicines or billing records.
    - Supports sorting, selection, and dynamic updates based on database queries.

4.2 BACKEND TOOLS

5. **MySQL**
    - Stores all project data including medicines, suppliers, customers, bills, and orders.
    - Supports complex queries, indexing, and relationships to maintain data integrity.

6. **MySQL Workbench**

   - Provides a graphical interface to design database schemas and manage data.
   - Helps in writing and testing SQL queries during development and debugging.

4.3 DATABASE

7. **SQL**
   - Language used to insert, update, delete, and retrieve data from the database.
   - Enables defining constraints, keys, and relationships for data consistency.

4.3 CODE

8. **Main.py**

```
from flask import Flask, render_template, request, session, redirect, flash
from flask_sqlalchemy import SQLAlchemy
import json


with open('config.json','r') as c:
    params = json.load(c)["params"]


local_server = True
app = Flask(__name__)
app.secret_key = 'super-secret-key'


if(local_server):
    app.config['SQLALCHEMY_DATABASE_URI'] = params['local_uri']
```

```
else:
    app.config['SQLALCHEMY_DATABASE_URI'] = params['proud_uri']


db = SQLAlchemy(app)


class Medicines(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    amount = db.Column(db.Integer, nullable=False)
    name = db.Column(db.String(500), nullable=False)
    medicines= db.Column(db.String(500), nullable=False)
    products = db.Column(db.String(500), nullable=False)
    email = db.Column(db.String(120), nullable=False)
    mid = db.Column(db.String(120), nullable=False)


class Posts(db.Model):
    mid = db.Column(db.Integer, primary_key=True)
    medical_name = db.Column(db.String(80), nullable=False)
    owner_name = db.Column(db.String(200), nullable=False)
    phone_no = db.Column(db.String(200), nullable=False)
    address = db.Column(db.String(120), nullable=False)


class Addmp(db.Model):
    sno = db.Column(db.Integer,  primary_key=True)
    medicine = db.Column(db.String, nullable=False)
```

```
class Addpd(db.Model):
    sno = db.Column(db.Integer,  primary_key=True)
    product = db.Column(db.String, nullable=False)

class Logs(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    mid = db.Column(db.String, nullable=True)
    action = db.Column(db.String(30), nullable=False)
    date = db.Column(db.String(100), nullable=False)

@app.route("/")
def hello():
    return render_template('index.html', params=params)

@app.route("/index")
def home():
    return render_template('dashbord.html', params=params)

@app.route("/search",methods=['GET','POST'])
def search():
    if request.method == 'POST':
        name = request.form.get('search')
        post = Addmp.query.filter_by(medicine=name).first()
```

```
        pro = Addpd.query.filter_by(product=name).first()
        if (post or pro):
            flash("Item Is Available.", "primary")
        else:
            flash("Item is not Available.", "danger")
    return render_template('search.html', params=params)


@app.route("/details", methods=['GET','POST'])
def details():
    if ('user' in session and session['user'] == params['user']):
        posts =Logs.query.all()
        return render_template('details.html', params=params, posts=posts)


@app.route("/aboutus")
def aboutus():
    return render_template('aboutus.html', params=params)


@app.route("/insert", methods = ['GET','POST'])
def insert():
    if (request.method == 'POST'):
        '''ADD ENTRY TO THE DATABASE'''
        mid=request.form.get('mid')
        medical_name = request.form.get('medical_name')
        owner_name = request.form.get('owner_name')
```

```
        phone_no = request.form.get('phone_no')

        address = request.form.get('address')

        push = Posts(mid=mid,medical_name=medical_name, owner_name=owner_name,
phone_no=phone_no, address=address)

        db.session.add(push)

        db.session.commit()

        flash("Thanks for submitting your details","danger")

    return render_template('insert.html',params=params)


@app.route("/addmp", methods = ['GET','POST'])

def addmp():

    if (request.method == 'POST'):

        '"ADD ENTRY TO THE DATABASE"'

        newmedicine = request.form.get('medicine')

        push=Addmp(medicine=newmedicine,)

        db.session.add(push)

        db.session.commit()

        flash("Thanks for adding new items", "primary")

    return render_template('search.html', params=params)


@app.route("/addpd", methods = ['GET','POST'])

def addpd():

    if (request.method == 'POST'):

        '"ADD ENTRY TO THE DATABASE"'

        newproduct = request.form.get('product')
```

```
        push=Addpd(product=newproduct,)
        db.session.add(push)
        db.session.commit()
        flash("Thanks for adding new items", "primary")
    return render_template('search.html', params=params)


@app.route("/list",methods=['GET','POST'])
def post():
    if ('user' in session and session['user'] == params['user']):
        posts=Medicines.query.all()
        return render_template('post.html', params=params, posts=posts)


@app.route("/items",methods=['GET','POST'])
def items():
    if ('user' in session and session['user'] == params['user']):
        posts=Addmp.query.all()
        return render_template('items.html', params=params,posts=posts)


@app.route("/items2", methods=['GET','POST'])
def items2():
    if ('user' in session and session['user'] == params['user']):
        posts=Addpd.query.all()
        return render_template('items2.html',params=params,posts=posts)
```

```
@app.route("/sp",methods=['GET','POST'])
def sp():
    if ('user' in session and session['user'] == params['user']):
        posts=Medicines.query.all()
        return render_template('store.html', params=params,posts=posts)


@app.route("/logout")
def logout():
    session.pop('user')
    flash("You are logout", "primary")
    return redirect('/login')


@app.route("/login",methods=['GET','POST'])
def login():
    if ('user' in session and session['user'] == params['user']):
        posts = Posts.query.all()
        return render_template('dashbord.html',params=params,posts=posts)
    if request.method=='POST':
        username=request.form.get('uname')
        userpass=request.form.get('password')
        if(username==params['user'] and userpass==params['password']):
            session['user']=username
            posts=Posts.query.all()
            flash("You are Logged in", "primary")
```

```
            return render_template('index.html',params=params,posts=posts)
        else:
            flash("wrong password", "danger")
    return render_template('login.html', params=params)


@app.route("/edit/<string:mid>",methods=['GET','POST'])
def edit(mid):
    if('user' in session and session['user']==params['user']):
        if request.method =='POST':
            medical_name=request.form.get('medical_name')
            owner_name=request.form.get('owner_name')
            phone_no=request.form.get('phone_no')
            address=request.form.get('address')
            if mid==0:

posts=Posts(medical_name=medical_name,owner_name=owner_name,phone_no=phone_no,address=add
ress)
                db.session.add(posts)
                db.session.commit()
            else:
                post=Posts.query.filter_by(mid=mid).first()
                post.medical_name=medical_name
                post.owner_name=owner_name
                post.phone_no=phone_no
                post.address=address
```

```
            db.session.commit()
            flash("data updated ", "success")
            return redirect('/edit/'+mid)
        post = Posts.query.filter_by(mid=mid).first()
        return render_template('edit.html',params=params,post=post)


@app.route("/delete/<string:mid>", methods=['GET', 'POST'])
def delete(mid):
    if ('user' in session and session['user']==params['user']):
        post=Posts.query.filter_by(mid=mid).first()
        db.session.delete(post)
        db.session.commit()
        flash("Deleted Successfully", "warning")
    return redirect('/login')


@app.route("/deletemp/<string:id>", methods=['GET', 'POST'])
def deletemp(id):
    if ('user' in session and session['user']==params['user']):
        post=Medicines.query.filter_by(id=id).first()
        db.session.delete(post)
        db.session.commit()
        flash("Deleted Successfully", "primary")
    return redirect('/list')
```

```
@app.route("/medicines", methods = ['GET','POST'])

def medicine():

    if(request.method=='POST'):

        '''ADD ENTRY TO THE DATABASE'''

        mid=request.form.get('mid')

        name=request.form.get('name')

        medicines=request.form.get('medicines')

        products=request.form.get('products')

        email=request.form.get('email')

        amount=request.form.get('amount')


entry=Medicines(mid=mid,name=name,medicines=medicines,products=products,email=email,amount=a
mount)

        db.session.add(entry)

        db.session.commit()

        flash("Data Added Successfully","primary")

    return render_template('medicine.html',params=params)



app.run(debug=True)
```

9. **SQL**

```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";

START TRANSACTION;

SET time_zone = "+00:00";



-- Character set setup
```

```
/*!40101 SET @OLD_CHARACTER_SET_CLIENT = @@CHARACTER_SET_CLIENT */;
 /*!40101 SET @OLD_CHARACTER_SET_RESULTS = @@CHARACTER_SET_RESULTS */;
 /*!40101 SET @OLD_COLLATION_CONNECTION = @@COLLATION_CONNECTION */;
 /*!40101 SET NAMES utf8mb4 */;

-- Table: addmp
CREATE TABLE `addmp` (
  `sno` int(11) NOT NULL,
  `medicine` varchar(500) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

INSERT INTO `addmp` (`sno`, `medicine`) VALUES
(1, 'Dolo 650'),
(2, 'Carpel 250 mg'),
(3, 'Azythromycin 500'),
(4, 'Azythromycin 250'),
(5, 'Rantac 300'),
(6, 'Omez'),
(7, 'Okacet'),
(8, 'Paracetomol');

-- Table: addpd
CREATE TABLE `addpd` (
  `sno` int(11) NOT NULL,
```

```
  `product` varchar(200) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

INSERT INTO `addpd` (`sno`, `product`) VALUES
(1, 'colgate'),
(2, 'perfume'),
(3, 'garnier face wash'),
(4, 'garnier face wash');

-- Table: logs
CREATE TABLE `logs` (
  `id` int(11) NOT NULL,
  `mid` int(11) NOT NULL,
  `action` varchar(500) NOT NULL,
  `date` varchar(500) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

INSERT INTO `logs` (`id`, `mid`, `action`, `date`) VALUES
(1, 1001, ' INSERTED', '2023-01-24 12:54:41'),
(2, 1001, ' DELETED', '2023-01-24 12:57:48');

-- Table: medicines
CREATE TABLE `medicines` (
  `id` int(11) NOT NULL,
```

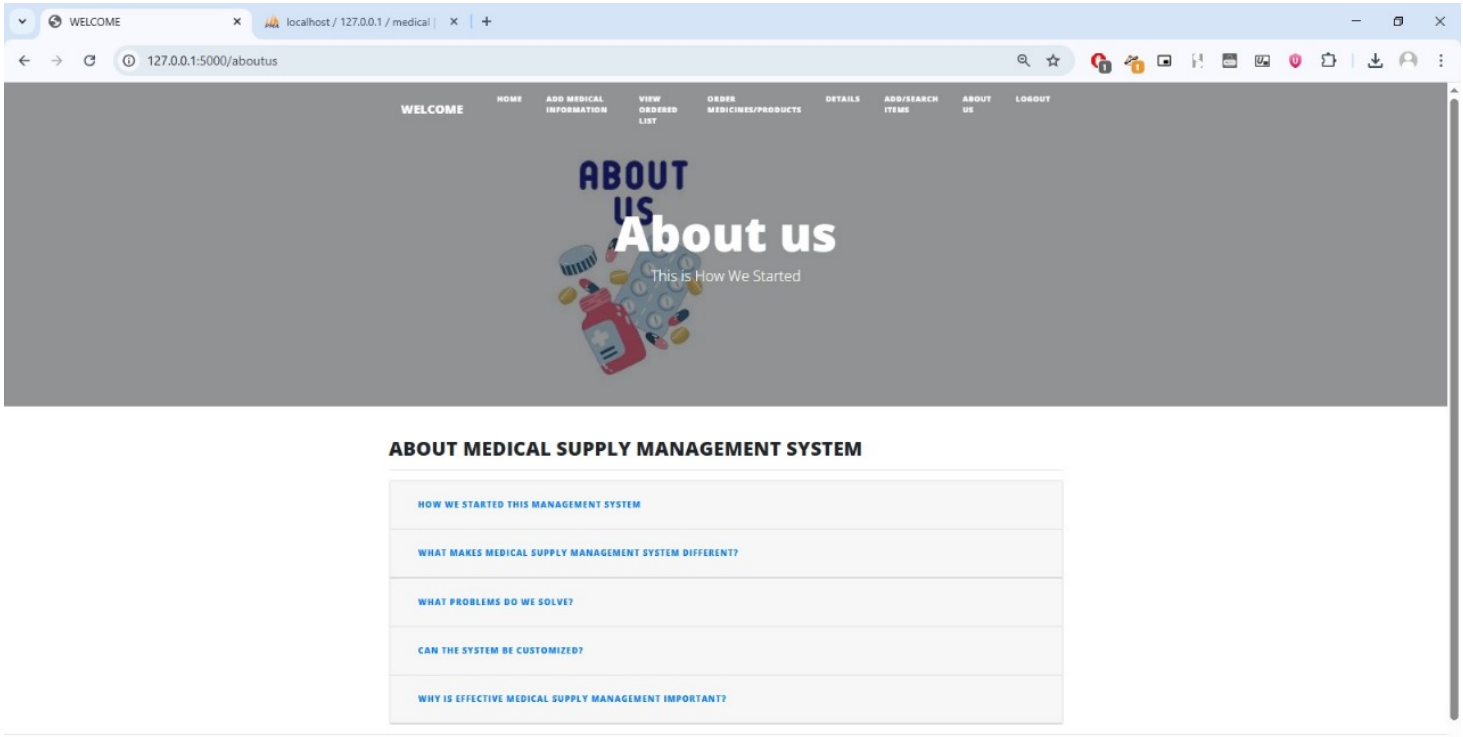
```
  `amount` int(11) NOT NULL,
  `name` varchar(100) NOT NULL,
  `medicines` varchar(500) NOT NULL,
  `products` varchar(500) NOT NULL,
  `email` varchar(50) NOT NULL,
  `mid` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Triggers
DELIMITER $$

CREATE TRIGGER `Delete` BEFORE DELETE ON `medicines`
FOR EACH ROW
INSERT INTO Logs VALUES (NULL, OLD.mid, ' DELETED', NOW());
$$

CREATE TRIGGER `Insert` AFTER INSERT ON `medicines`
FOR EACH ROW
INSERT INTO Logs VALUES (NULL, NEW.mid, ' INSERTED', NOW());
$$

CREATE TRIGGER `Update` AFTER UPDATE ON `medicines`
FOR EACH ROW
INSERT INTO Logs VALUES (NULL, NEW.mid, ' UPDATED', NOW());
```

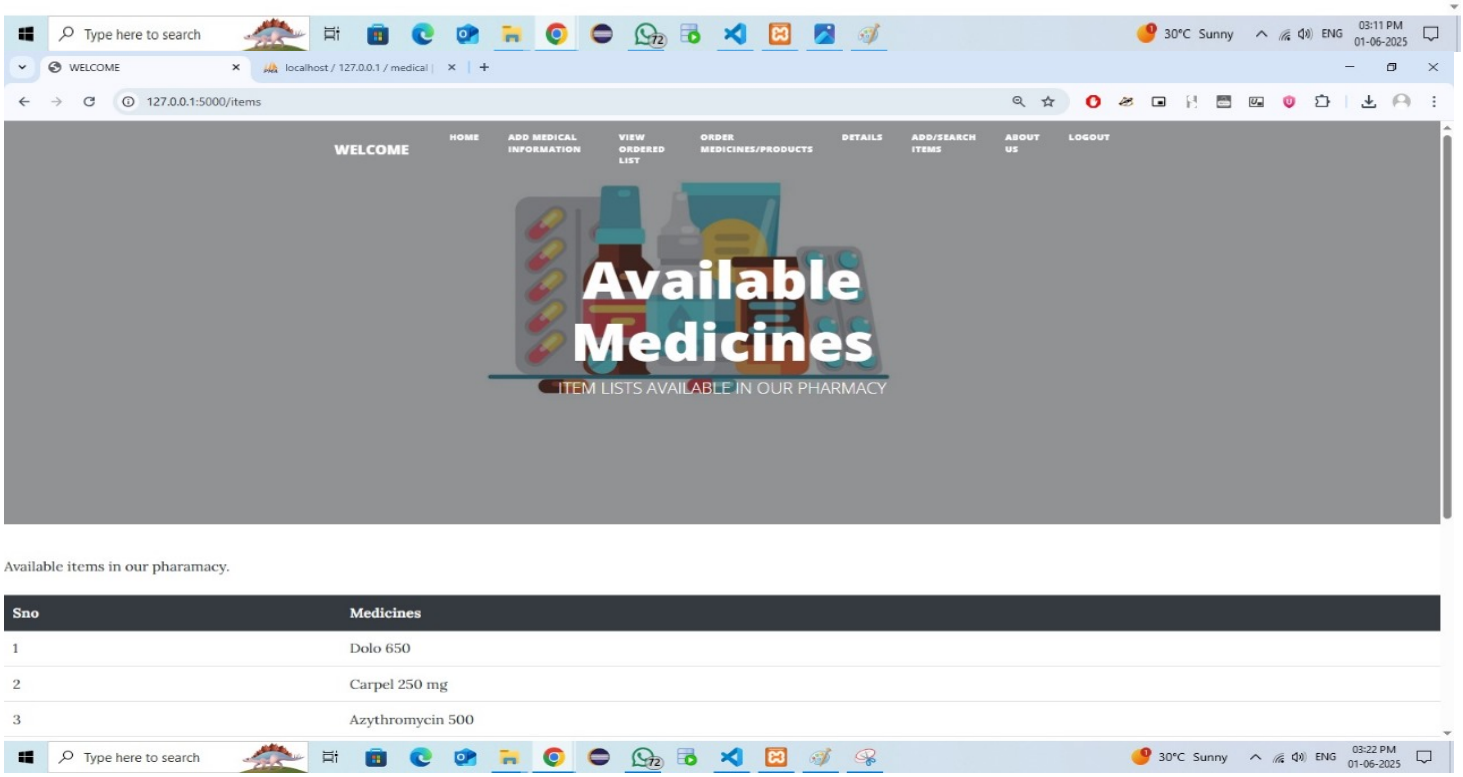
```
$$

DELIMITER ;

-- Table: posts
CREATE TABLE `posts` (
  `mid` int(11) NOT NULL,
  `medical_name` varchar(100) NOT NULL,
  `owner_name` varchar(100) NOT NULL,
  `phone_no` varchar(20) NOT NULL,
  `address` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

INSERT INTO `posts` (`mid`, `medical_name`, `owner_name`, `phone_no`, `address`) VALUES
(1001, 'ARK PROCODER MEDICAL', 'ANEES', '7896541230', 'Bangalore');

-- Primary Keys
ALTER TABLE `addmp` ADD PRIMARY KEY (`sno`);
ALTER TABLE `addpd` ADD PRIMARY KEY (`sno`);
ALTER TABLE `logs` ADD PRIMARY KEY (`id`);
ALTER TABLE `medicines` ADD PRIMARY KEY (`id`);
ALTER TABLE `posts` ADD PRIMARY KEY (`mid`);

-- Auto-Increment
```

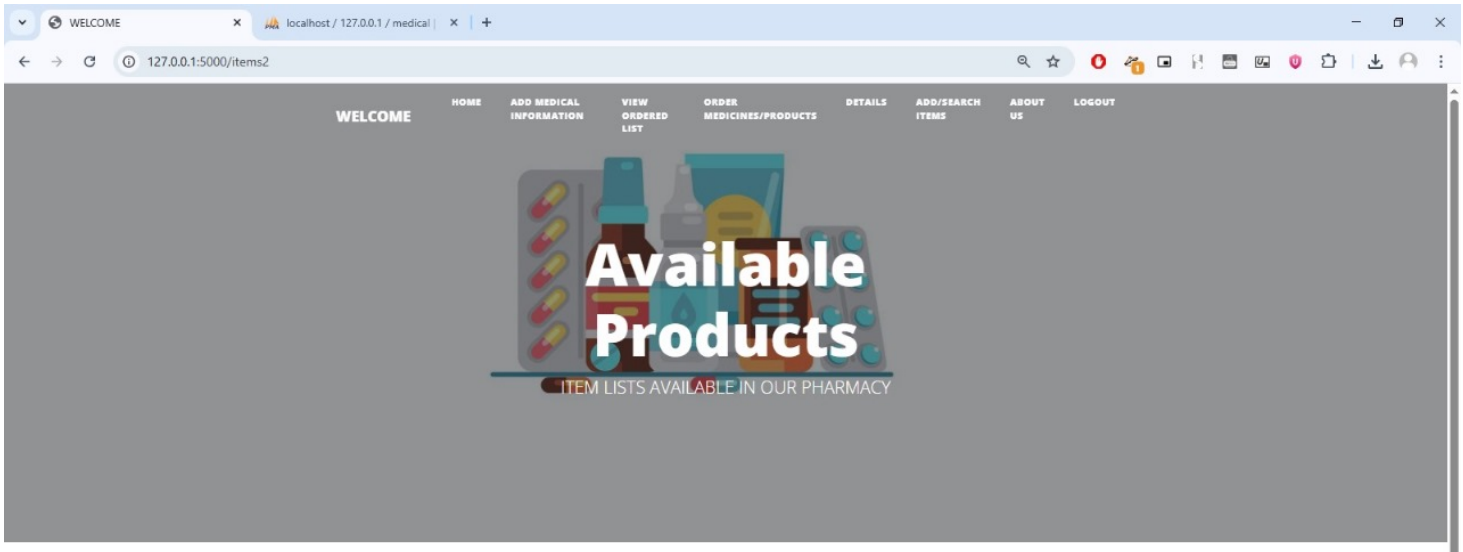
```
ALTER TABLE `addmp` MODIFY `sno` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT = 9;

ALTER TABLE `addpd` MODIFY `sno` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT = 5;

ALTER TABLE `logs` MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT = 3;

ALTER TABLE `medicines` MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT = 3;

ALTER TABLE `posts` MODIFY `mid` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT = 1002;


COMMIT;


/*!40101 SET CHARACTER_SET_CLIENT = @OLD_CHARACTER_SET_CLIENT */;

/*!40101 SET CHARACTER_SET_RESULTS = @OLD_CHARACTER_SET_RESULTS */;

/*!40101 SET COLLATION_CONNECTION = @OLD_COLLATION_CONNECTION */;
```

CHAPTER 5

SNAPSHOTS

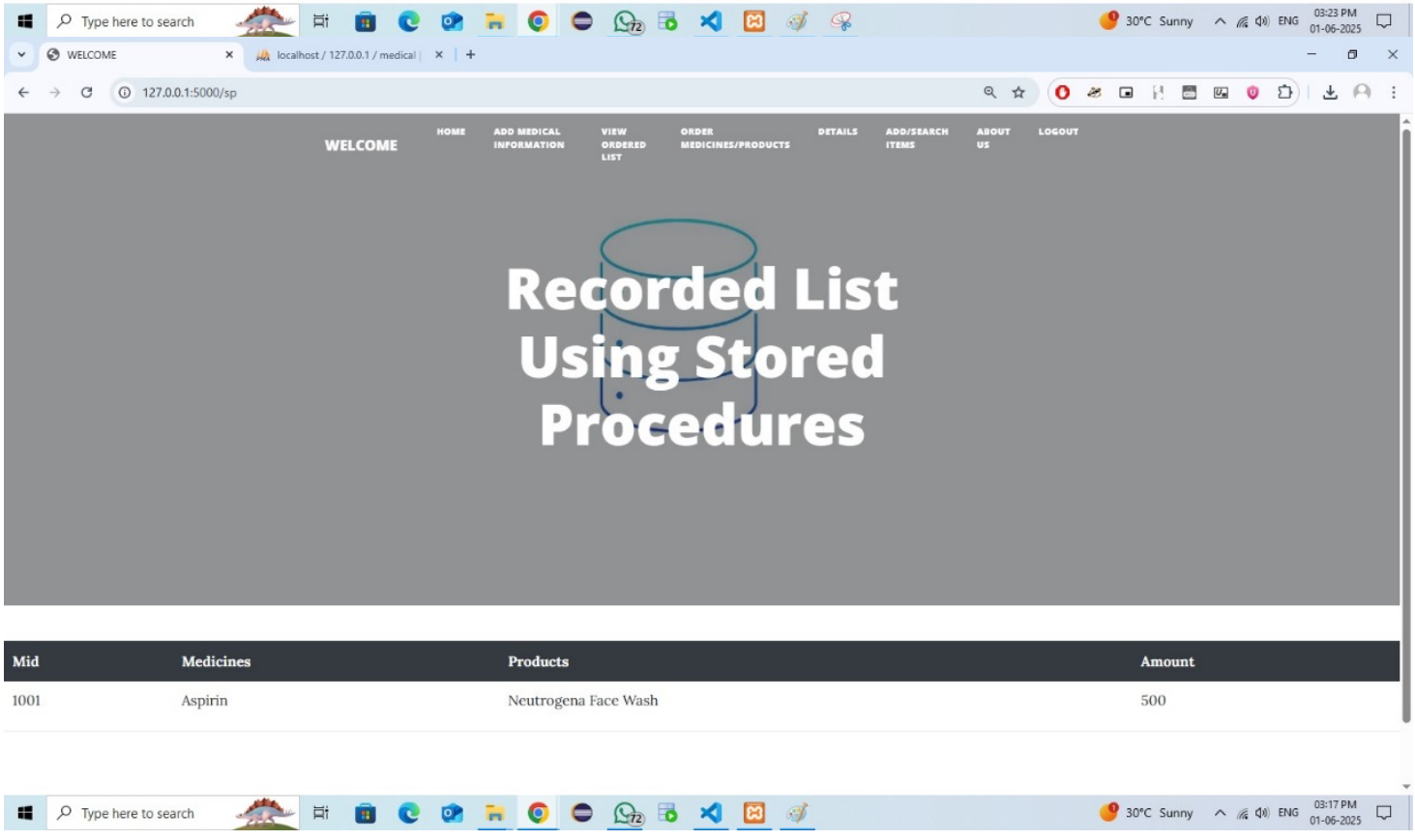




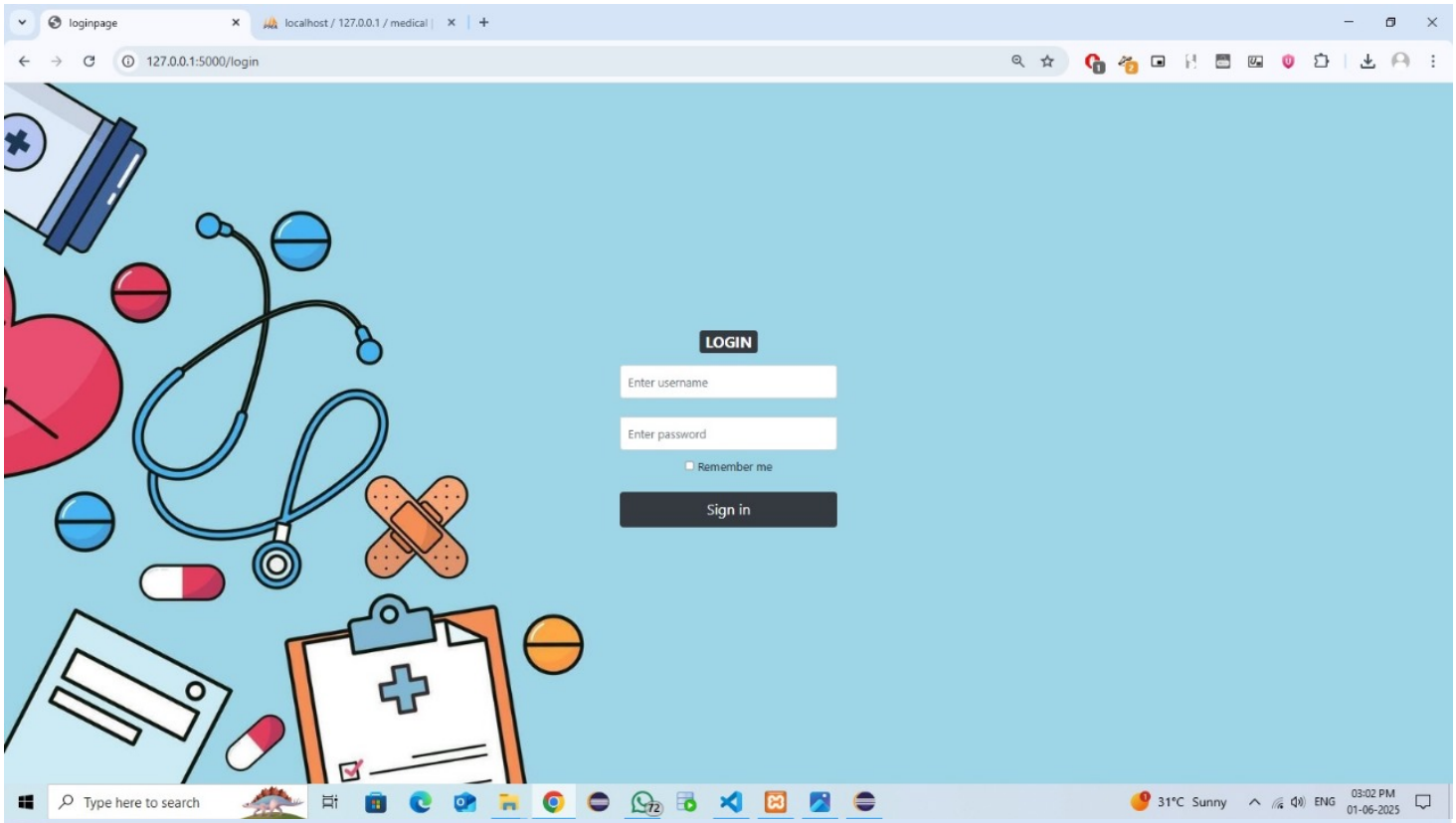


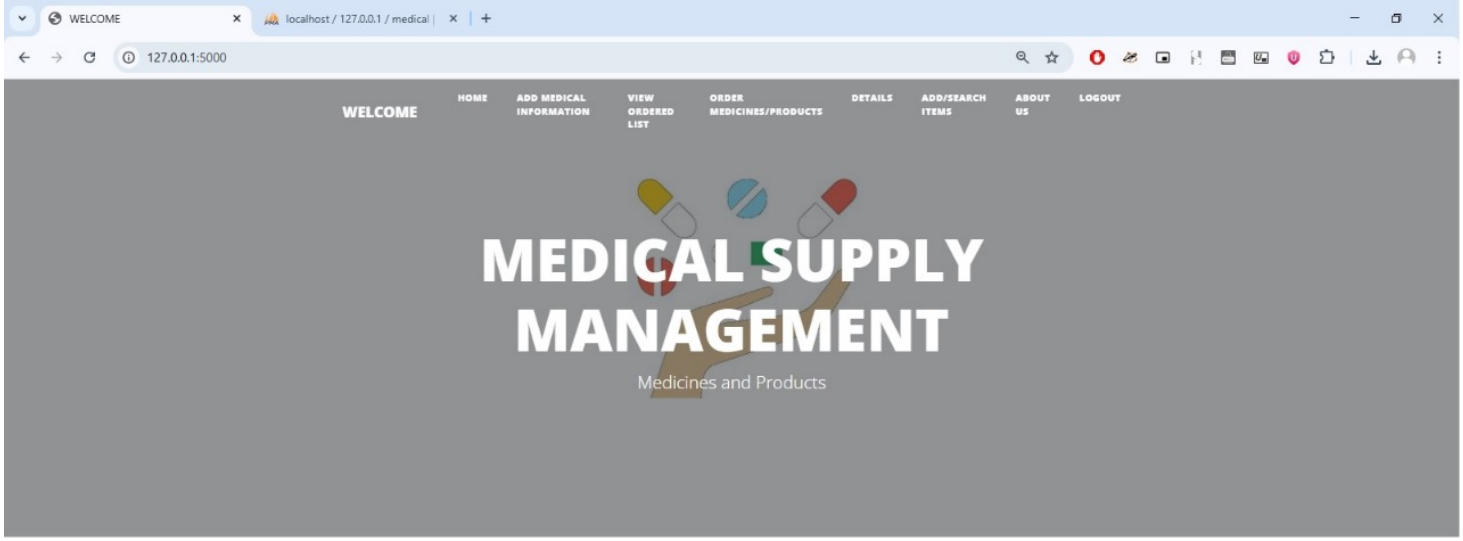
Avalaible items in our pharamacy

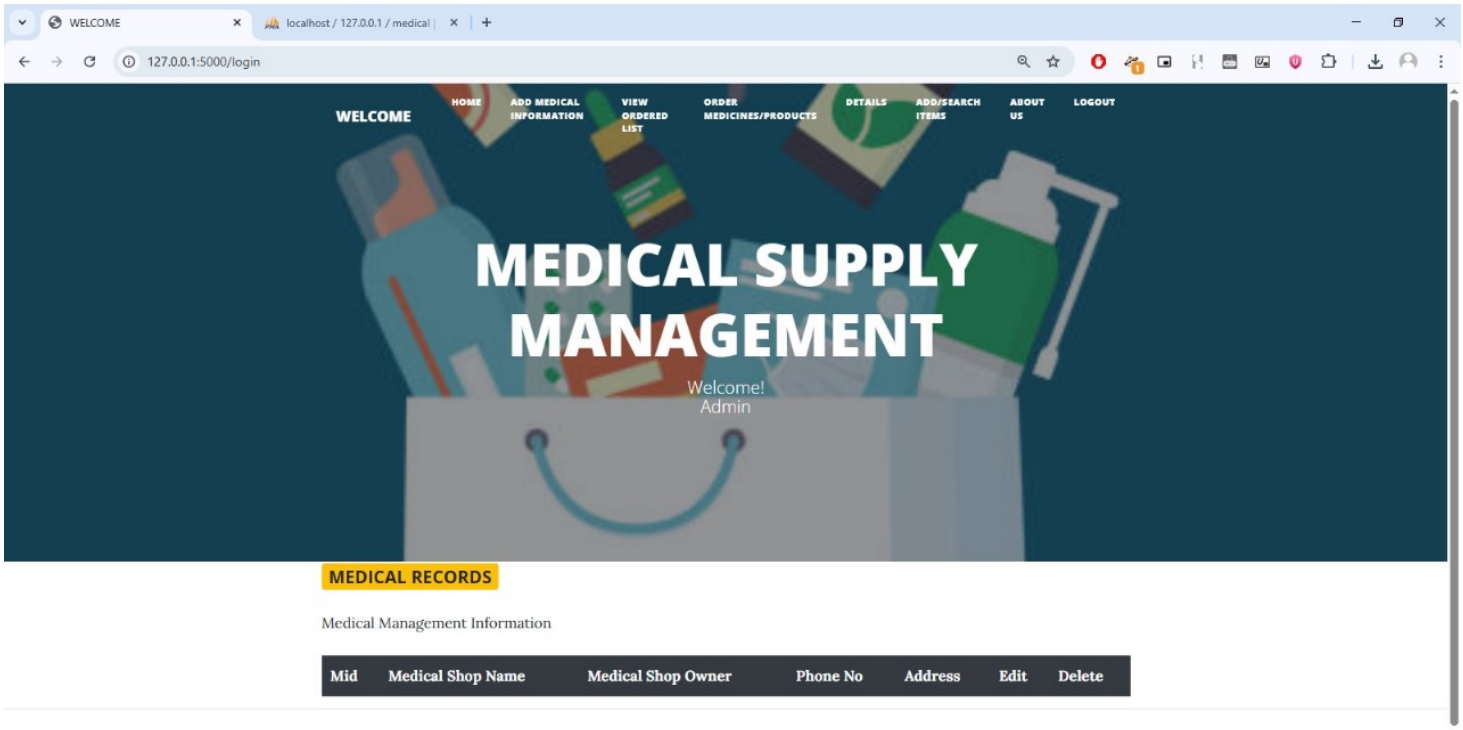
Sno	Products
1	colgate
2	perfume
3	garnier face wash

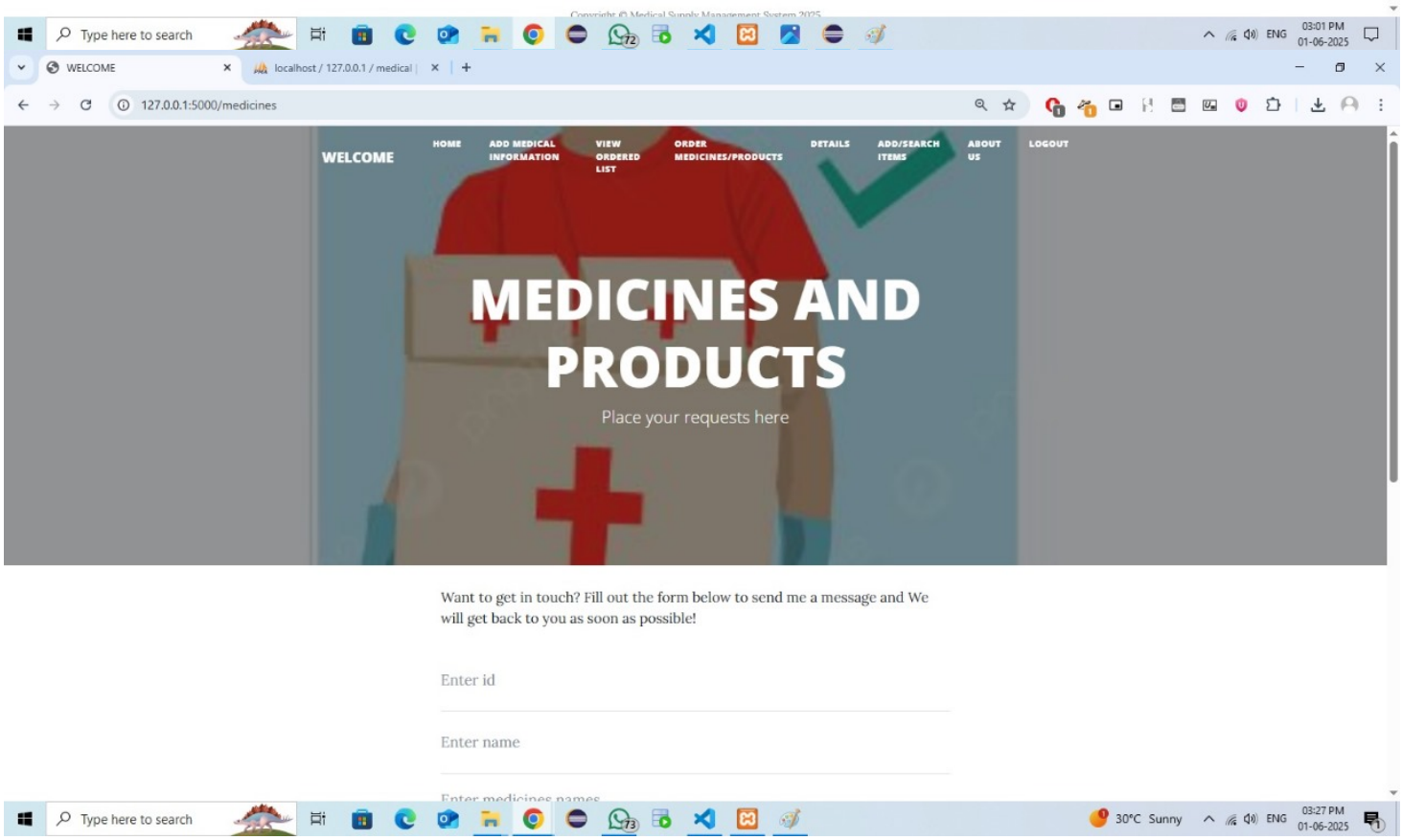


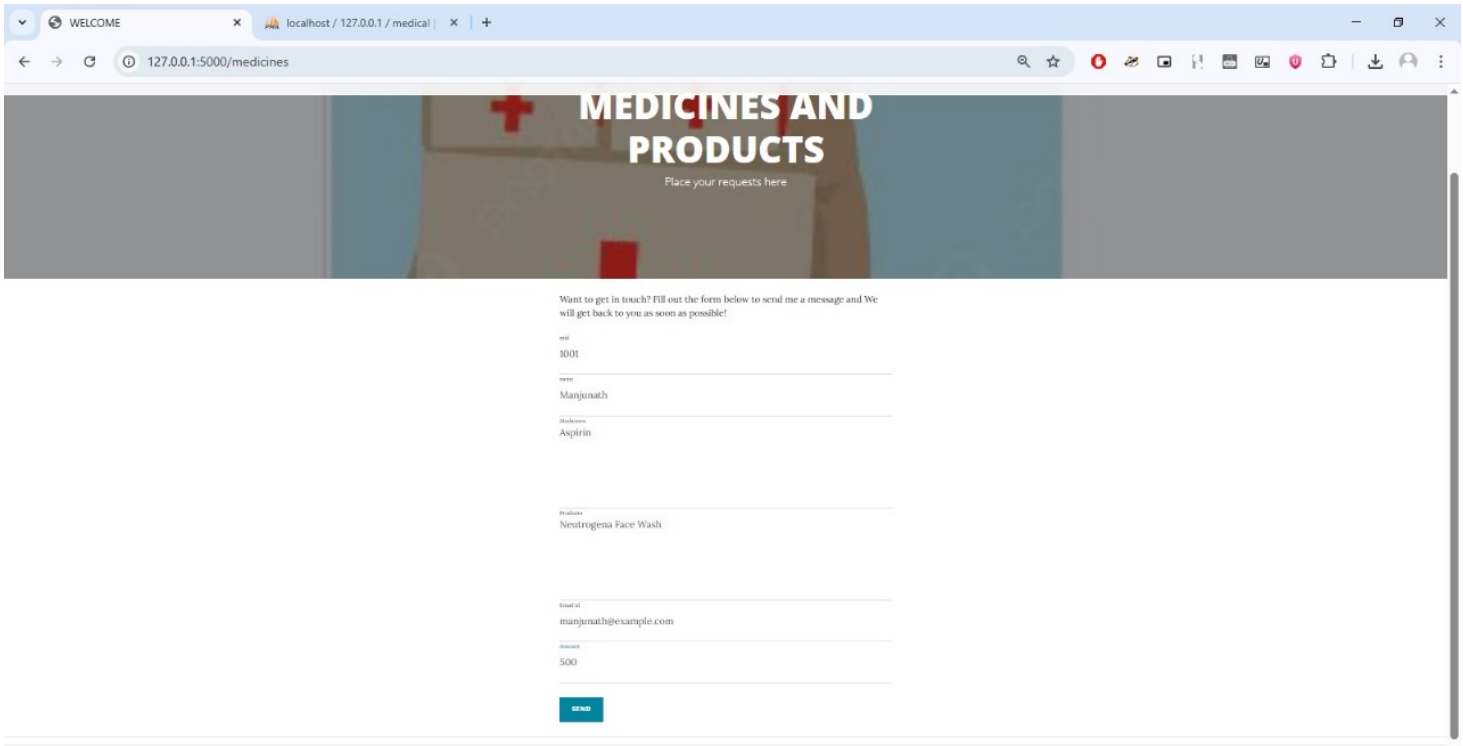
Mid	Medicines	Products	Amount
1001	Aspirin	Neutrogena Face Wash	500

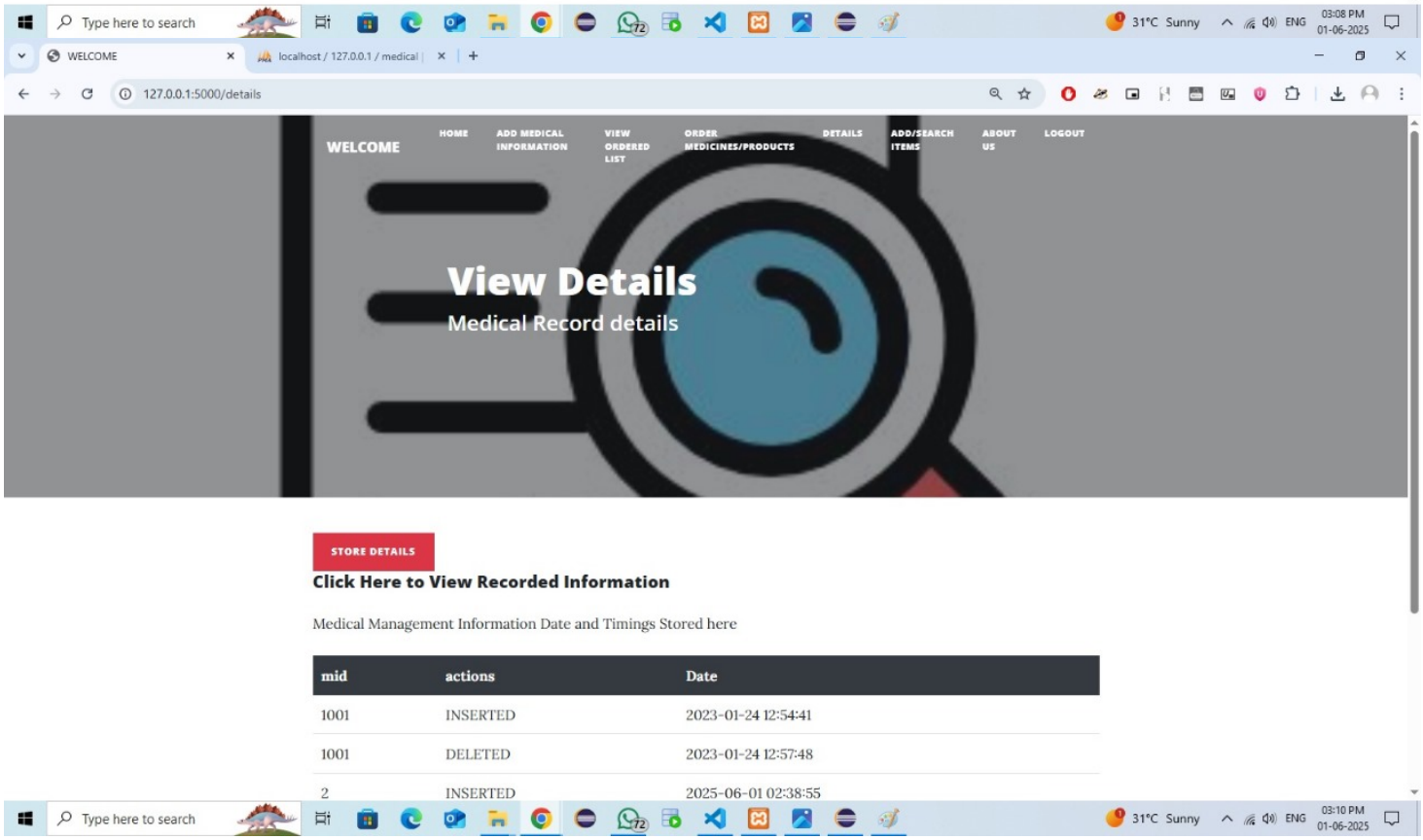


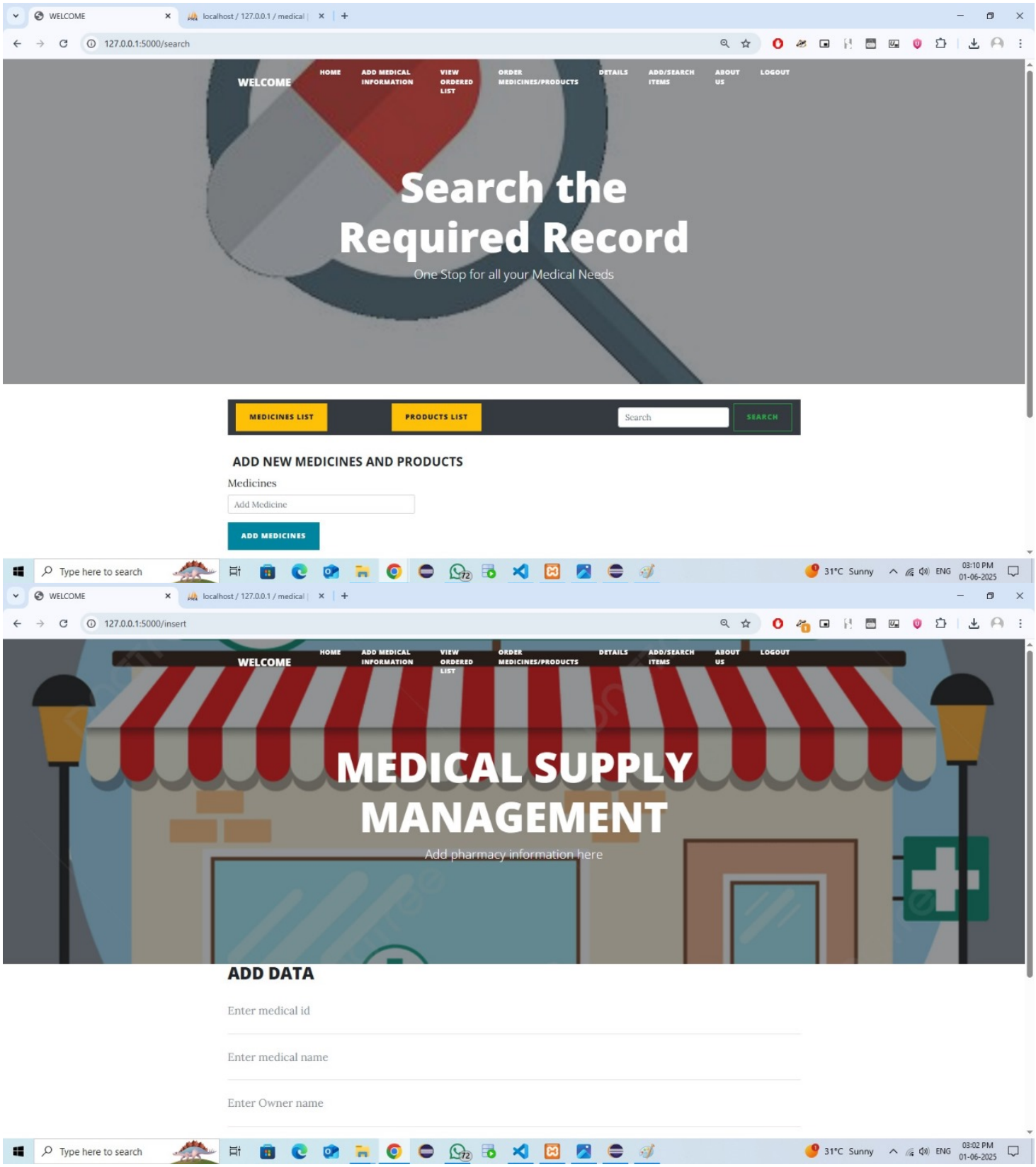


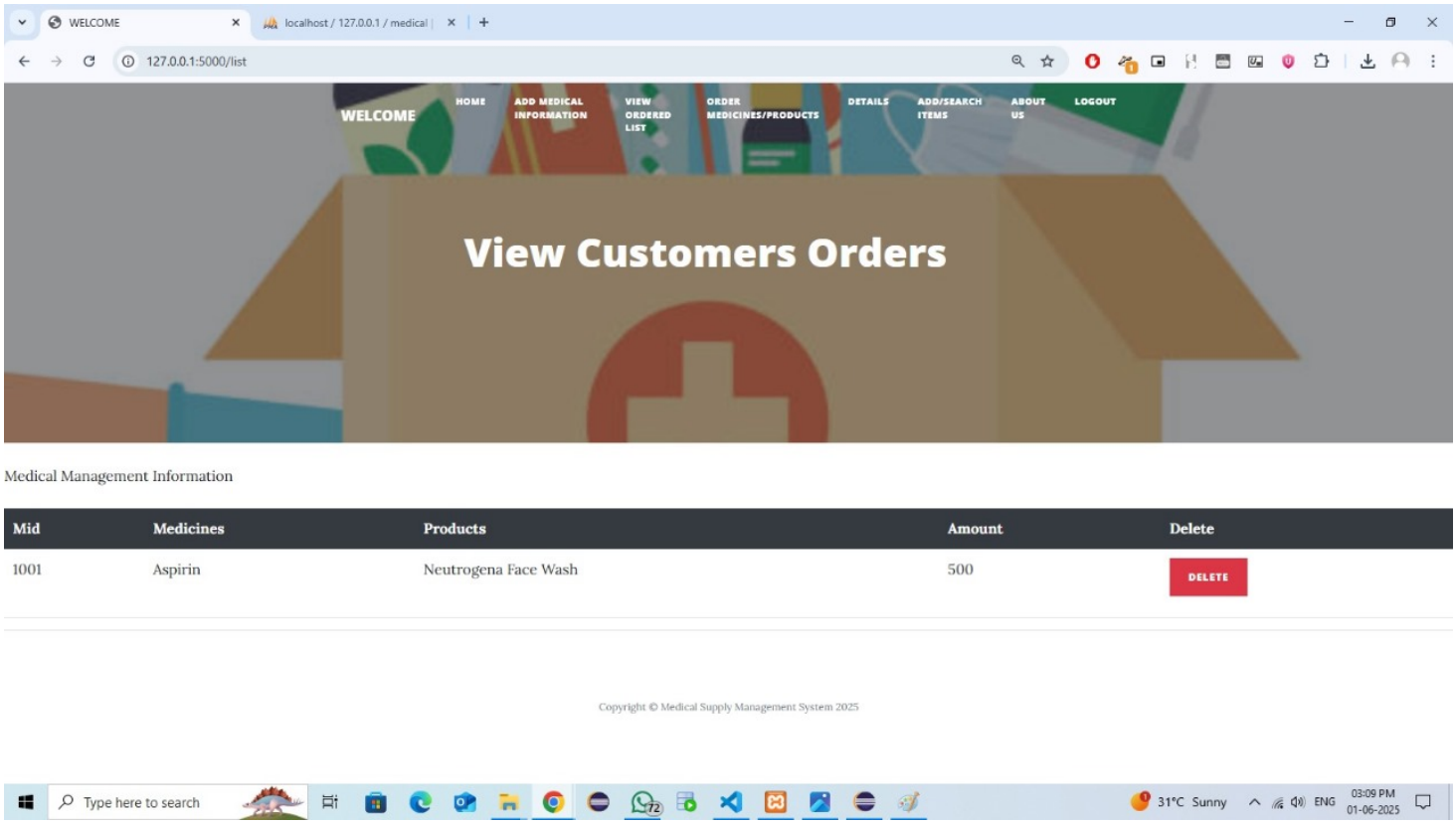

WELCOME
HOME
ADD MEDICAL INFORMATION
VIEW ORDERED LIST
ORDER MEDICINES/PRODUCTS
DETAILS
ADD/SEARCH ITEMS
ABOUT US
LOGOUT
MEDICAL SUPPLY MANAGEMENT
Welcome!
Admin
MEDICAL RECORDS
Medical Management Information
Mid
Medical Shop Name
Medical Shop Owner
Phone No
Address
Edit
Delete


WELCOME
HOME
ADD MEDICAL INFORMATION
VIEW ORDERED LIST
ORDER MEDICINES/PRODUCTS
DETAILS
ADD/SEARCH ITEMS
ABOUT US
LOGOUT
MEDICINES AND PRODUCTS
Place your requests here
Want to get in touch? Fill out the form below to send me a message and We will get back to you as soon as possible!
Enter id
Enter name


WELCOME
localhost / 127.0.0.1 / medical
127.0.0.1:5000/medicines
MEDICINES AND PRODUCTS
Place your requests here
Want to get in touch? Fill out the form below to send me a message and We will get back to you as soon as possible!
1001
Manjunath
Aspirin
Neutrogena Face Wash
manjunath@example.com
500
SEND


Type here to search
31°C Sunny
ENG
03:08 PM
01-06-2025
WELCOME
localhost / 127.0.0.1 / medical
127.0.0.1:5000/details
WELCOME
HOME
ADD MEDICAL INFORMATION
VIEW ORDERED LIST
ORDER MEDICINES/PRODUCTS
DETAILS
ADD/SEARCH ITEMS
ABOUT US
LOGOUT
View Details
Medical Record details
STORE DETAILS
Click Here to View Recorded Information
Medical Management Information Date and Timings Stored here
mid
actions
Date
1001
INSERTED
2023-01-24 12:54:41
1001
DELETED
2023-01-24 12:57:48
2
INSERTED
2025-06-01 02:38:55
Type here to search
31°C Sunny
ENG
03:10 PM
01-06-2025


WELCOME
localhost / 127.0.0.1 / medical
127.0.0.1:5000/search
WELCOME
HOME
ADD MEDICAL INFORMATION
VIEW ORDERED LIST
ORDER MEDICINES/PRODUCTS
DETAILS
ADD/SEARCH ITEMS
ABOUT US
LOGOUT
Search the Required Record
One Stop for all your Medical Needs
MEDICINES LIST
PRODUCTS LIST
Search
SEARCH
ADD NEW MEDICINES AND PRODUCTS
Medicines
Add Medicine
ADD MEDICINES
Type here to search
31°C Sunny
ENG
03:10 PM
01-06-2025
WELCOME
localhost / 127.0.0.1 / medical
127.0.0.1:5000/insert
WELCOME
HOME
ADD MEDICAL INFORMATION
VIEW ORDERED LIST
ORDER MEDICINES/PRODUCTS
DETAILS
ADD/SEARCH ITEMS
ABOUT US
LOGOUT
MEDICAL SUPPLY MANAGEMENT
Add pharmacy information here
ADD DATA
Enter medical id
Enter medical name
Enter Owner name
Type here to search
31°C Sunny
ENG
03:02 PM
01-06-2025


WELCOME
localhost / 127.0.0.1 / medical
127.0.0.1:5000/list
WELCOME
HOME
ADD MEDICAL INFORMATION
VIEW ORDERED LIST
ORDER MEDICINES/PRODUCTS
DETAILS
ADD/SEARCH ITEMS
ABOUT US
LOGOUT
View Customers Orders
Medical Management Information
Mid
Medicines
Products
Amount
Delete
1001
Aspirin
Neutrogena Face Wash
500
DELETE
Copyright © Medical Supply Management System 2025
Type here to search
31°C Sunny
ENG
03:09 PM
01-06-2025

CHAPTER 6
CONCLUSION

The **Medical Supply Management System** was created to simplify and automate the daily operations of a pharmacy or medical store. It addresses key challenges such as managing inventory, supplier and customer records, billing processes, and order tracking. By replacing manual paperwork with a digital solution, the system improves efficiency, accuracy, and accessibility of information.

The frontend of the application is built using **Python's Tkinter library**, which provides a simple yet effective GUI. Through its clean interface, users can perform tasks like adding medicines, generating bills, and searching customer records with ease. The design focuses on usability, allowing even non-technical staff to operate the system comfortably and reliably.

The backend is powered by a **MySQL database**, which stores all essential data in a structured format. Tables for medicines, suppliers, customers, orders, and billing are linked using primary and foreign keys to maintain data integrity. SQL queries handle data retrieval and manipulation, ensuring that operations like stock updates and bill generation are handled efficiently and correctly.

Connectivity between the frontend and backend is established using the **MySQL Connector/Python** library. This enables real-time data exchange as users interact with the interface, with the backend processing queries and returning results instantly. Secure connection handling and parameterized queries add a layer of safety and ensure reliable performance.

In conclusion, this project successfully integrates a graphical user interface with a relational database to solve real-world business needs. It demonstrates practical knowledge of GUI design, SQL database management, and backend integration. The system not only improves pharmacy workflow but also serves as a strong foundation for future upgrades such as multi-user support, advanced reporting, or migration to a web-based platform.