CHAPTER-1

INTRODUCTION

1.1 Objectives:

The primary goal of this project is to develop a streamlined and user-friendly computerized system for managing pharmacy supplies. The specific objectives are:

- Designing an intuitive system that is easy for users to navigate and operate efficiently.
- Eliminating redundant data to ensure the accuracy and flexibility of the system.
- Conducting a comprehensive study of pharmacy supply management systems to inform the design process.
- Creating software with fast processing capabilities and an appealing user interface.
- Building a robust software solution that can be used for an extended period without frequent errors or the need for maintenance.
- Establishing synchronized and centralized databases to facilitate seamless communication between farmers and sellers.
- Leveraging computerization to optimize time and cost efficiency.
- Enhancing the user experience with a visually appealing Graphical User Interface (GUI).
- Implementing stringent data security measures, including login credentials, to safeguard sensitive information.
- Enabling quick and easy storage and retrieval of data to support efficient decision-making.
- Improving coordination in the distribution and management of medicines.
- Minimizing reliance on paper-based records to reduce administrative burden and environmental impact.

1.2 Limitations in the Current Market :

Despite the benefits, there are certain limitations to be mindful of:

- Manual data entry may be time-consuming, impacting overall efficiency.
- Traditional paper-based record-keeping methods can lead to a significant amount of paperwork.
- The use of physical files and registers may require additional storage space.
- Paper-based storage is susceptible to damage and loss, compromising data integrity.
- Integration with official databases, such as Aadhar, has not been implemented, limiting the system's functionality and data verification capabilities.

CHAPTER-2

STUDY OF EXISTING SYSTEM

2.1 CASE STUDY

Rising debt, cost-cutting, and layoffs in health care-delivery facilities, alluded to earlier. Models for the design and operation of supply chain networks may be steady state or dynamic and may be deterministic or deal with uncertainties (particularly in product demands). Research in this field started very early on, with location-allocation problems forming part of the earlier set of ‘‘classical’’ operations research problems. The gap between the growing demand and available supply of high-quality, cost effective, and timely health care continues to be a daunting challenge not only in developing and underdeveloped countries, but also in developed countries. Further, the issues involved with the supply chain design in developing countries are prevalent in developed countries, especially with the rising number of uninsured and jobless among the patient populations and with the budget deficits. Thus the project is a sincere effort in simplifying the tasks of administrators in an easily usable format.

2.2 SUMMARY OF WORK

While there has been no consensus on the definition of Pharmacy Supply Management in the literature, they have proposed that researchers adopt the below definition to allow for the coherent development of theory in the area. In order to have a successful supply management, we need to make many decisions related to the flow of information, product, and funds. Each decision should be made in a way to increase the whole supply chain profitability . Supply management is more complex in healthcare and other industries because of the impact on people’s health requiring adequate and accurate medical supply according to the patient’s need.

CHAPTER 3

SYSTEM SPECIFICATION

3.1 SOFTWARE REQUIREMENTS SPECIFICATION

3.1.1 SOFTWARE REQUIREMENTS:

- Frontend: HTML, CSS, Java Script, Bootstrap
- Backend: Python flask (Python 3.7) , SQLAlchemy,
- Operating System: Windows 10/ Mac OS
- Browser: Google Chrome/ Brave
- Server: XAMPPS (Version-3.7)
- Python main editor (user interface): PyCharm Community
- Workspace editor: Visual Studio Code

HARDWARE REQUIREMENTS:

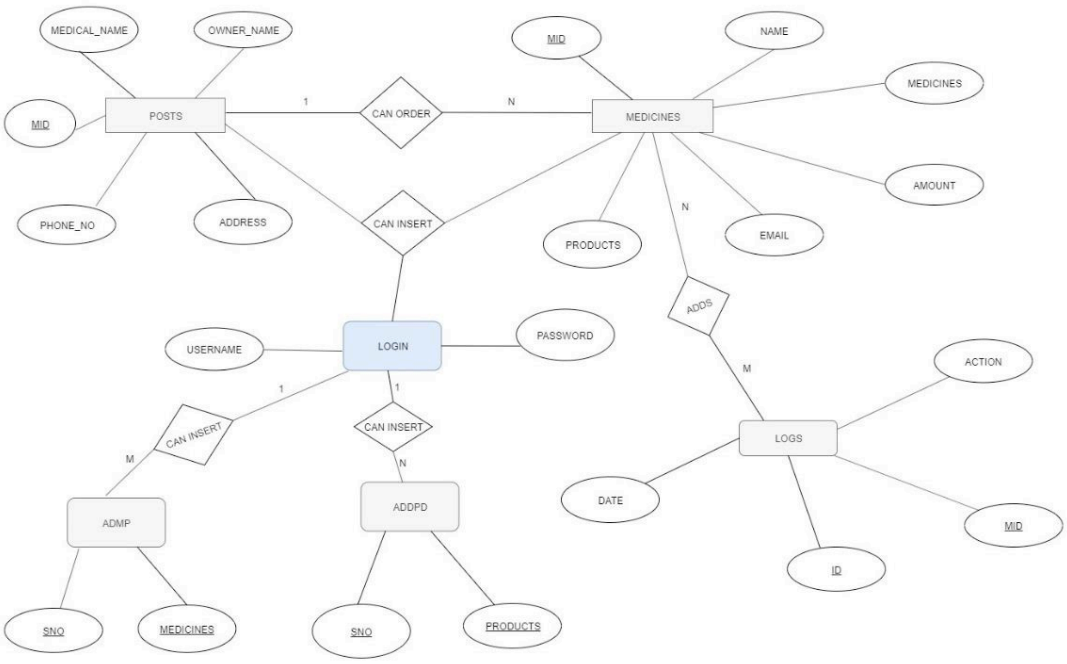
- Computer with a 1.1 GHz or faster processor
- Minimum 2GB of RAM or more
- 2.5 GB of available hard-disk space
- 5400 RPM hard drive
- 1366 × 768 or higher-resolution display
- DVD-ROM drive

CHAPTER 4

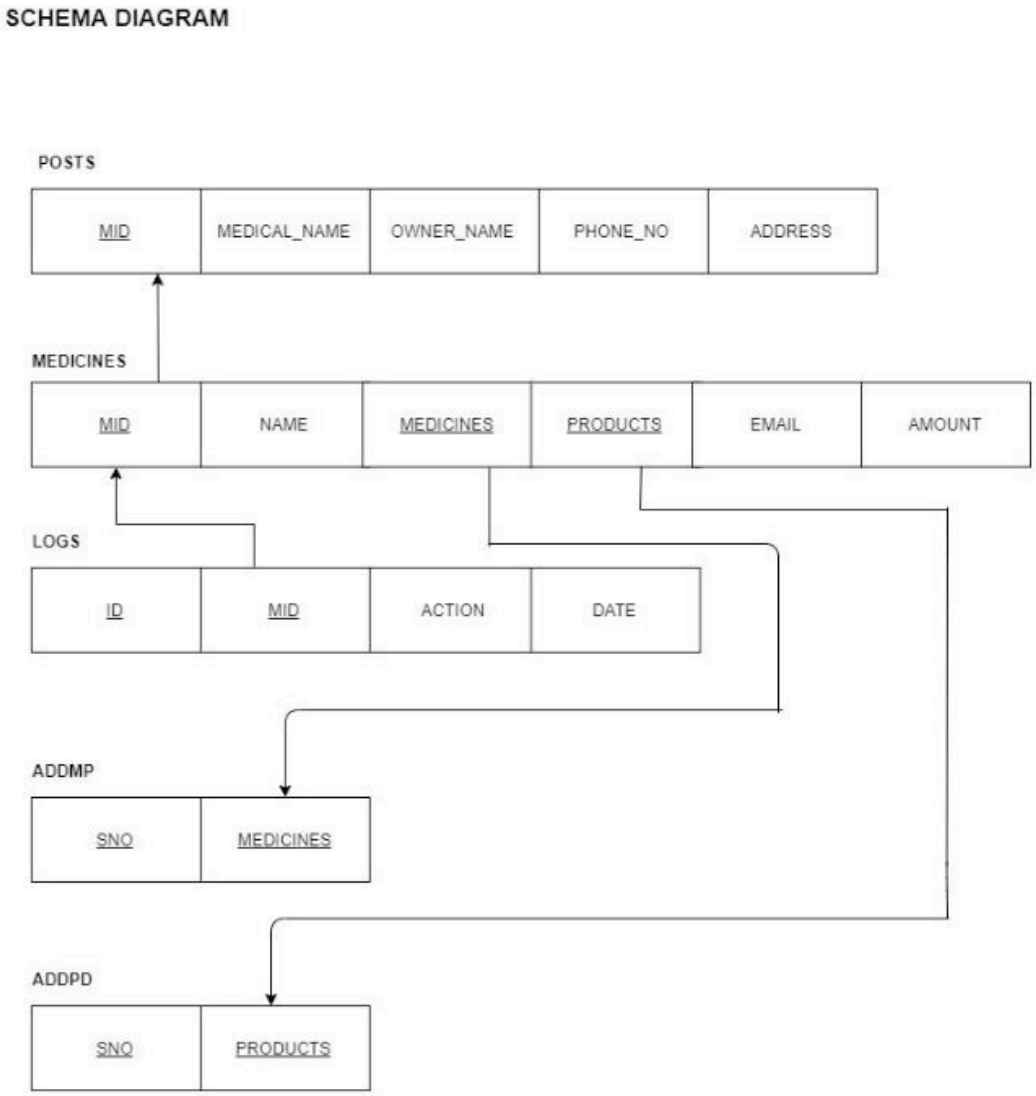
SYSTEM DESIGN

4.1 CONCEPTUAL DESIGN:

4.1.1 E-R DIAGRAM:



4.1.2 SCHEMA DIAGRAM:



4.2 IMPLEMENTATION:

An "implementation" of Python should be taken to mean a program or environment which provides support for the execution of programs written in the Python language, as represented by the CPython reference implementation.
There have been and are several distinct software packages providing of what we all recognize as Python, although some of those are more like distributions or variants of some existing implementation than a completely new implementation of the language.

Back End (MySQL)

Database:

A Database Management System (DBMS) is software designed to manage databases, large sets of structured data, and perform operations requested by users. Common examples include Oracle, DB2, Microsoft Access, SQL Server, Firebird, PostgreSQL, MySQL, SQLite, FileMaker, and Sybase Adaptive Server Enterprise. Initially prevalent in large companies with adequate hardware, DBMSs are now standard in various company back offices.

A DBMS is a complex software set controlling the organization, storage, management, and data retrieval in a database. Key components include:

- Modeling Language: Defines the schema for each database according to the DBMS data model. The dominant model, embedded in SQL, is practical despite objections from relational model purists. Many DBMSs support the Open Database Connectivity API for standardized programmer access.

- Data Structures: Optimized for handling large amounts of data stored on permanent storage, implying slower access compared to volatile main memory.

- Query Language and Report Writer: Allows users to interactively query the database, analyze data, and update it based on user privileges.

- Data Security: Prevents unauthorized access or modifications. Users can access the entire database or specific subsets (sub-schemas) based on privileges.

- Interactive Data Management: Enables interactive database entry, update, and interrogation, facilitating personal database management. However, this may lack audit

trails and necessary controls in multi-user environments, requiring customized application programs.

- Transaction Mechanism: Ideally guarantees ACID properties, ensuring data integrity despite concurrent user accesses and faults. Also maintains the database's integrity.

- Redundancy Avoidance: DBMS prevents duplicate records through unique index constraints (e.g., customer numbers).

- When using a DBMS, adapting information systems to changing requirements becomes easier. Organizations might use different DBMS types for daily transactions and analysis. Overall systems design decisions are made by data administrators and systems analysts, while detailed database design is handled by database administrators.

- SQL (Structured Query Language):

- SQL is the language for manipulating relational databases, closely tied to the relational model. In this model, data is stored in structures called relations or tables.

SQL statements serve various purposes:

- Data Definition (DDL): Defines tables and structures in the database, used for creating, altering, and dropping schema objects like tables and indexes.
- Data Manipulation (DML): Involves operations like SELECT, INSERT, UPDATE, and DELETE. These statements allow users to retrieve, add, modify, or delete data in the database.
- Data Control Language (DCL): Manages access to data within the database. Commands like GRANT and REVOKE control the user's privileges and permissions.
- Transaction Control Language (TCL): Manages transactions within the database. Commands like COMMIT and ROLLBACK ensure data integrity by finalizing or undoing transactions.

4.3: Stored Procedure

Routine name: proc 1,post proc
Type: procedure
Definition: Select from posts;
Select from medicines;

4.4: Triggers

It is the special kind of stored procedure that automatically executes when an event occurs in the database.
Triggers used :

1: Trigger name: on insert
Table: medicines
Time: after
Event: insert
Definition: INSERT INTO logs VALUES(null, new.mid, 'inserted', NOW());

2: Trigger name: on delete
Table: medicines
Time: after
Event: delete
Definition: INSERT INTO logs VALUES(null, old.mid, 'deleted', NOW());

CHAPTER 5 CONCLUSION

PHARMACY MANAGEMENT SYSTEM successfully implemented offline medicines supply management database which helps us in administrating the data user for managing the tasks performed in medicines supply. The project successfully used various functionalities of Ampps and python flask and also create the fully functional database management system for offline pharmacy.

Using MySQL as the database is highly beneficial as it is free to download, popular and can be easily customized. The data stored in the MySQL database can easily be retrieved and manipulated according to the requirements with basic knowledge of SQL.
With the theoretical inclination of our syllabus it becomes very essential to take the atmost advantage of any opportunity of gaining practical experience that comes along. The building blocks of this Major Project "Pharmacy Supply Management System" was one of these opportunities. It gave us the requisite practical knowledge to supplement the already taught theoretical concepts thus making us more competent as a computer engineer.

The project from a personal point of view also helped us in understanding the following aspects of project development:
• The planning that goes into implementing a project.
• The importance of proper planning and an organized methodology.
• The key element of team spirit and co-ordination in a successful project.

FUTURE ENHANCEMENT

• Enhanced database storage facility
• Enhanced user friendly GUI
• More advanced transportation of medicines
• Online Bill payments

REFERENCES

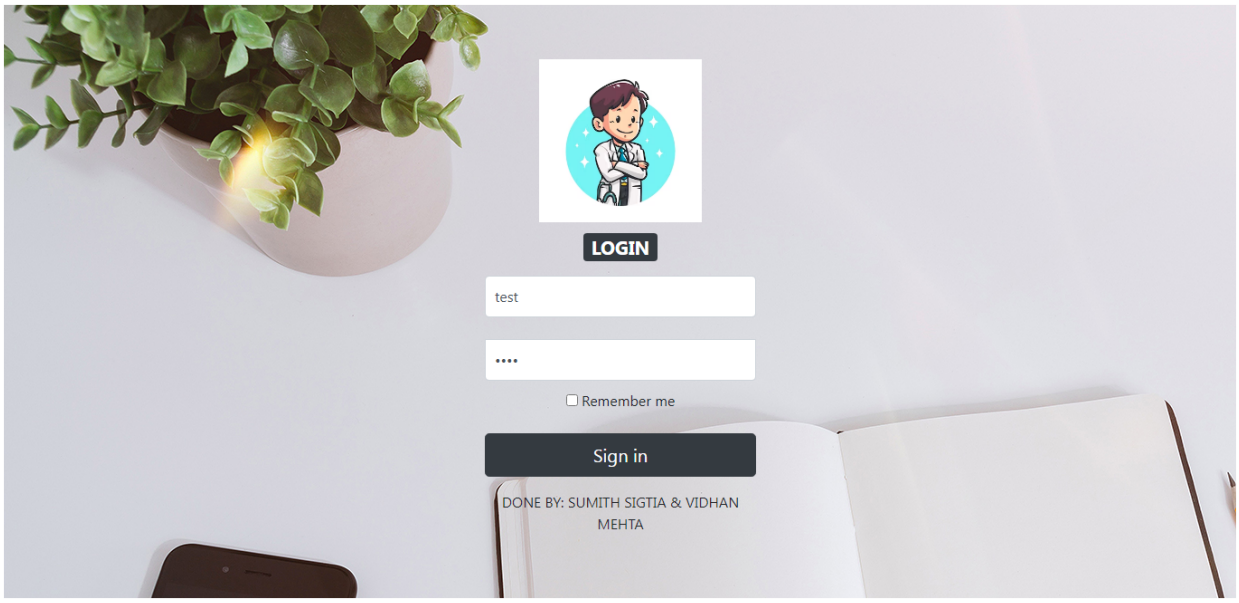
• https://www.youtube.com/
• https://www.google.com/
• http://www.getbootstrap.com/
• https://bootstrapmade.com/
• https://tailwindcss.com/
• https://pypi.org/project/Flask/

APPENDIX A

USER INTERFACES

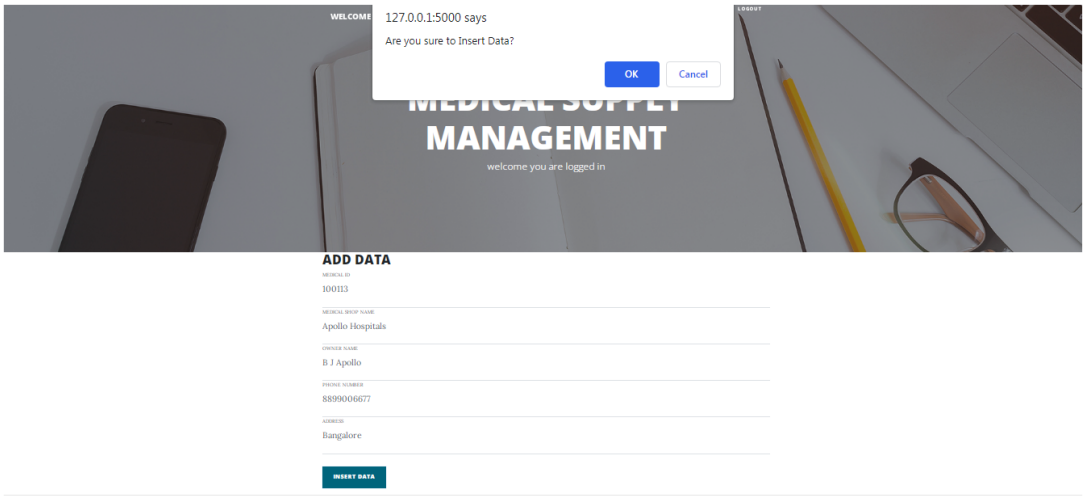
SCREEN SHOTS

Login Page:

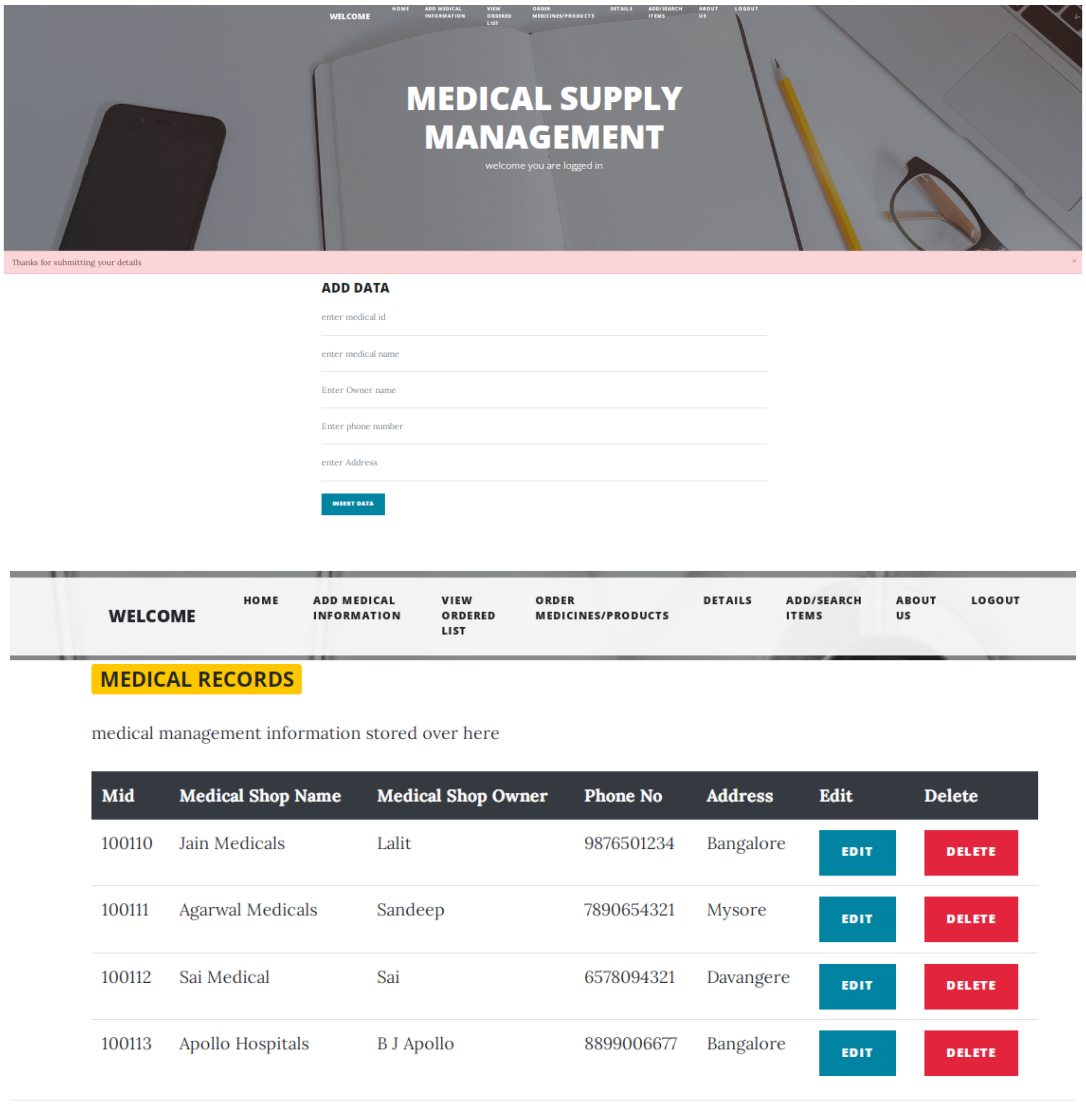


Home Page:

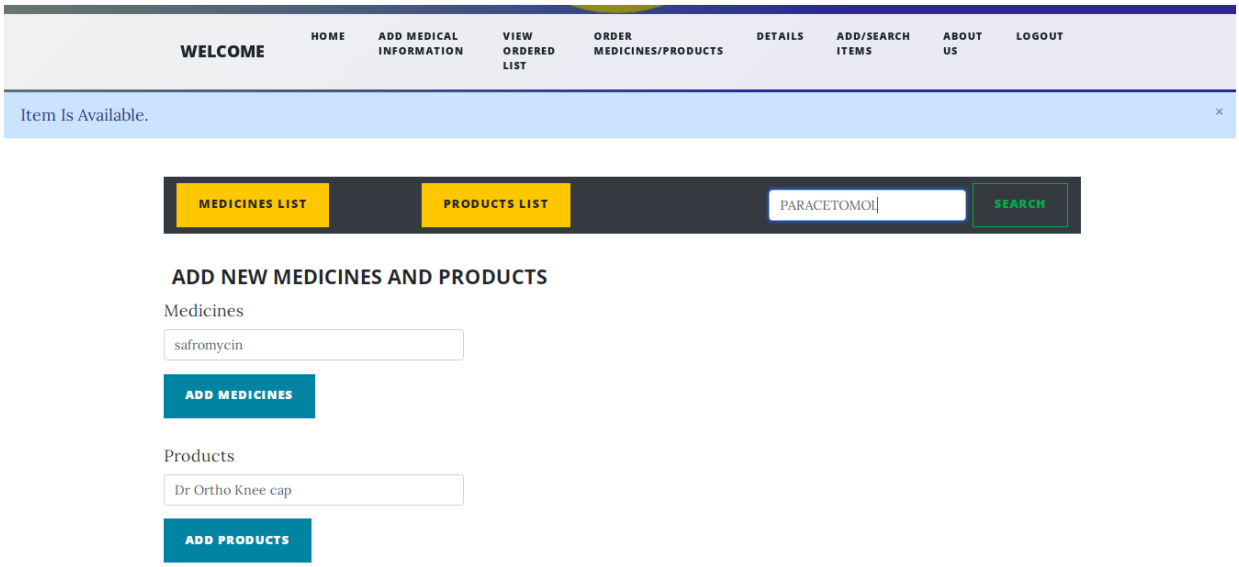
Add Medical Store Information page:



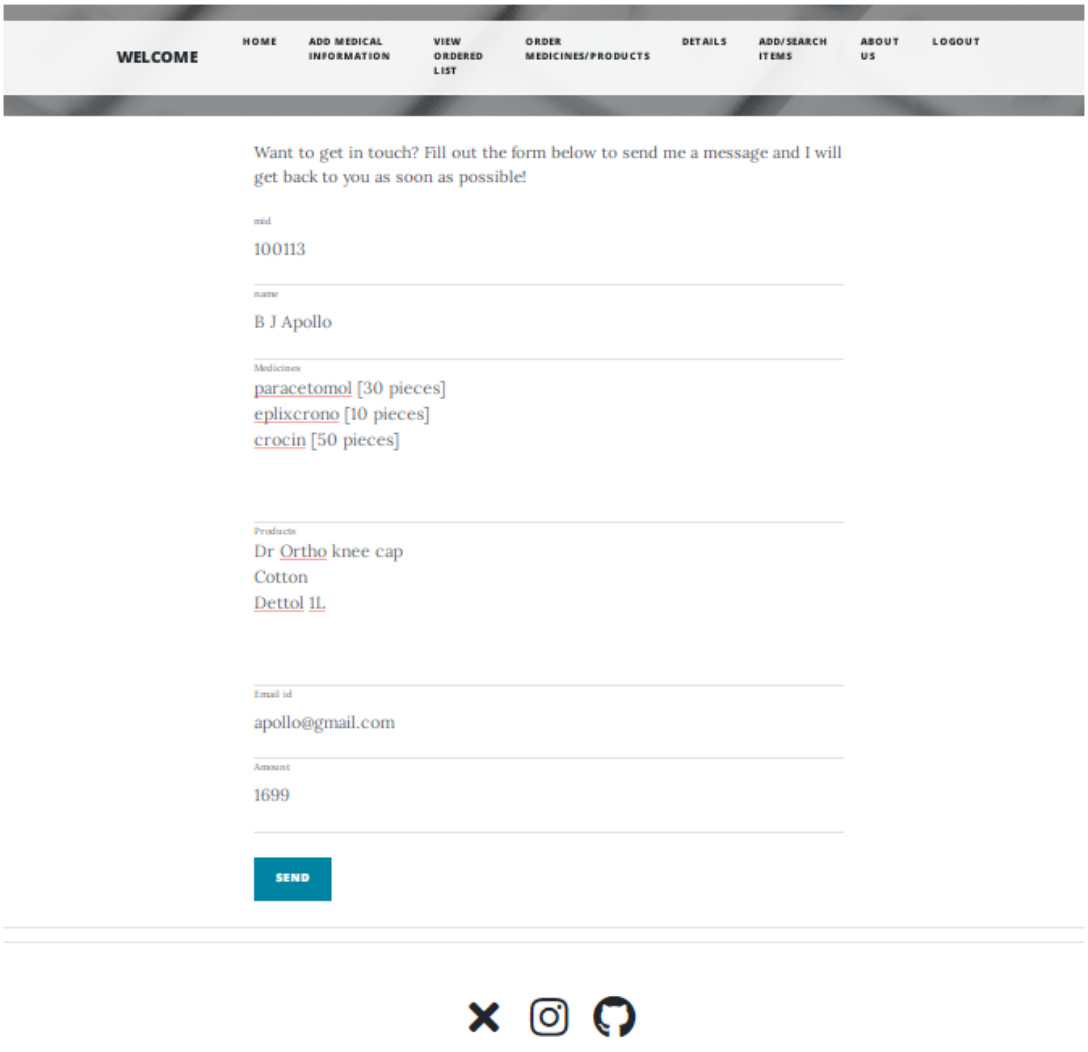
Medical Store Information Submitted:



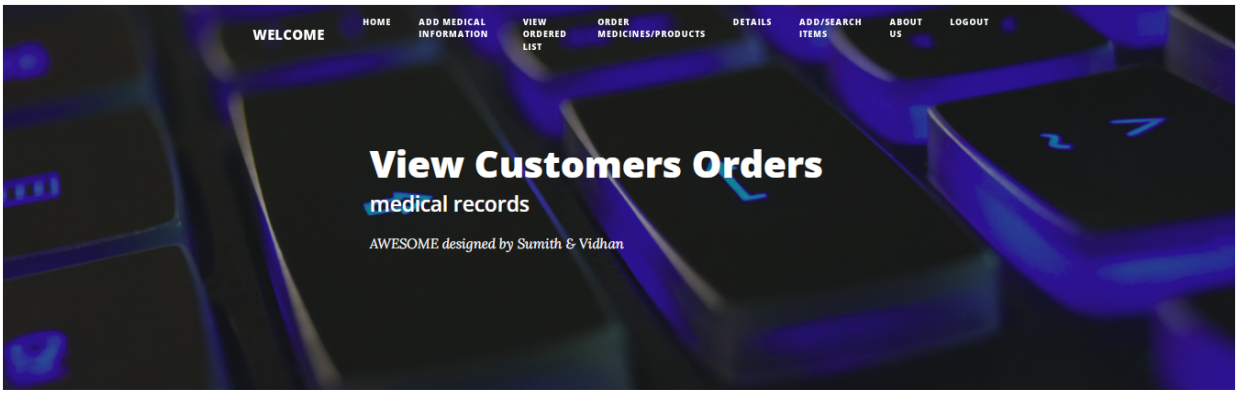
Add Medicines and Products Page:



Order Medicines & Products page:



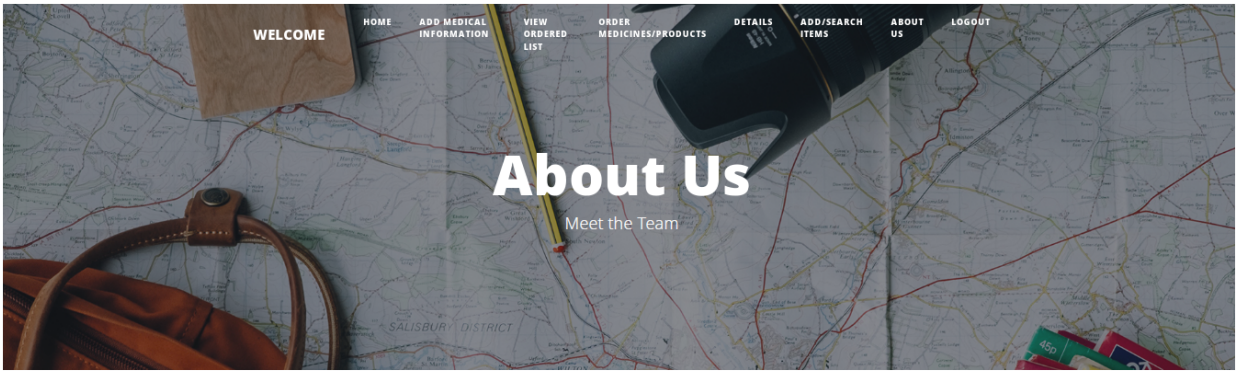
Order Summary Page:



medical management information stored over here

Mid	Medicines	Products	Amount	Delete
100112	dolo 650 carpel 250mg	cotton	250	DELETE
100113	paracetomol [30 pieces] eplixcrono [10 pieces] crocin [50 pieces]	Dr Ortho knee cap Cotton Dettol 1L	1699	DELETE

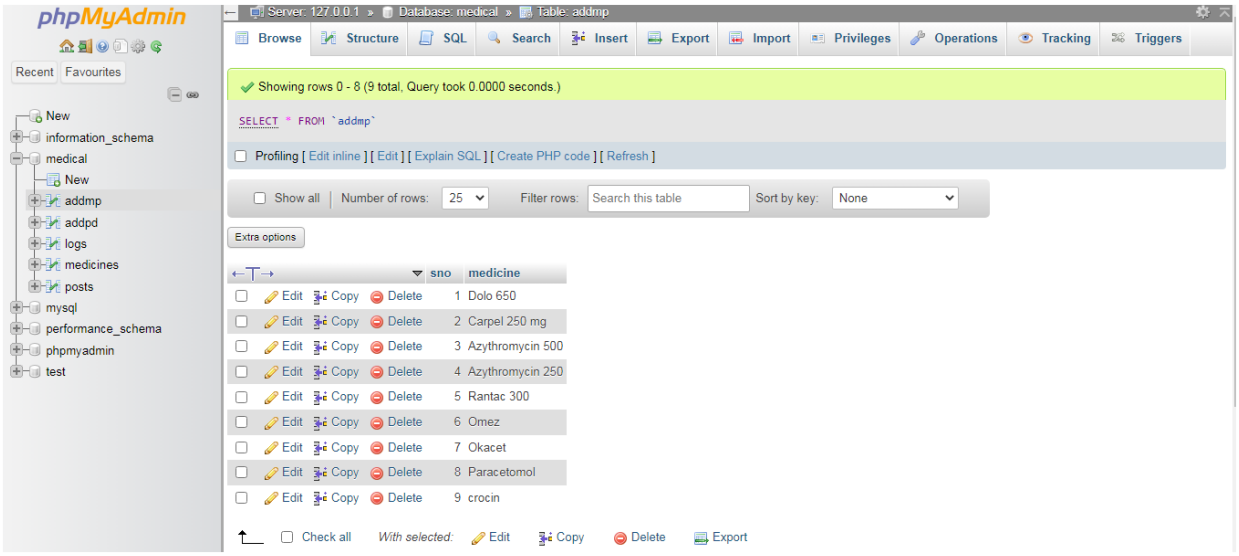
About us page:

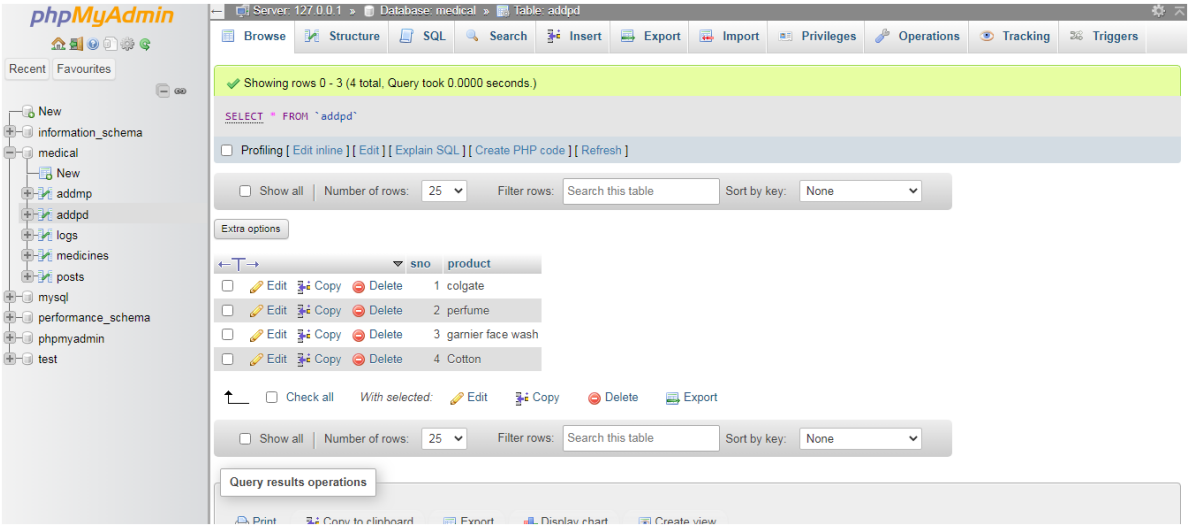


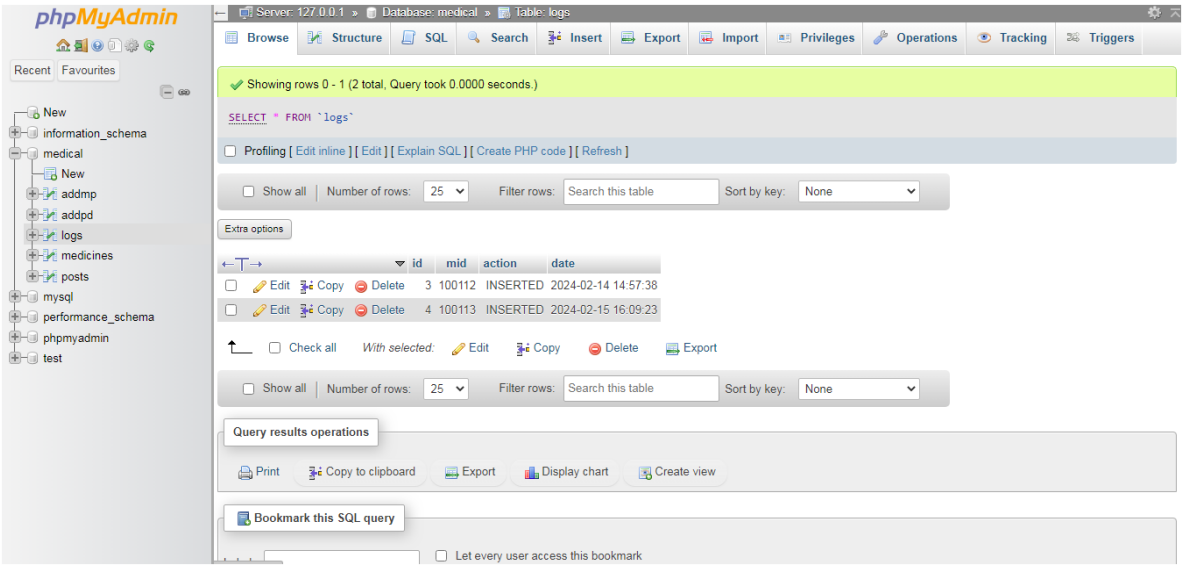
We are Sumith Sigtia and Vidhan Mehta, students of The Oxford College of Engineering at Bommanahalli, Bangalore. As part of our DBMS Mini project for the 5th semester in the AIML Department, B.E., we designed and implemented this Medical Supply Management System.

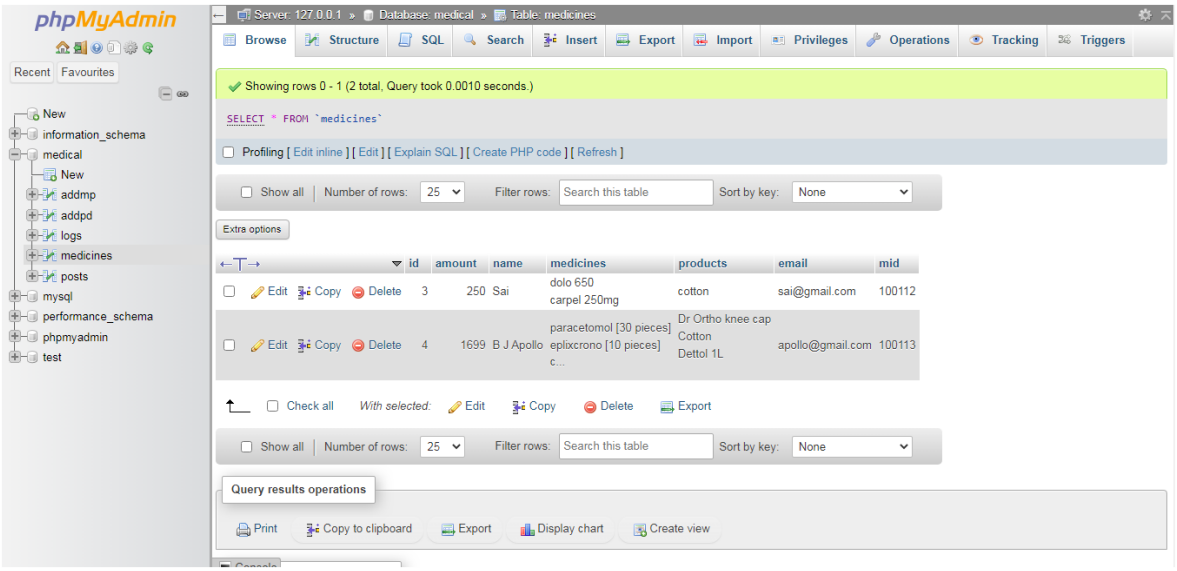
Our aim is to provide an efficient and user-friendly platform for managing medical records, medicines, and products. Feel free to explore and make use of the features we've implemented.

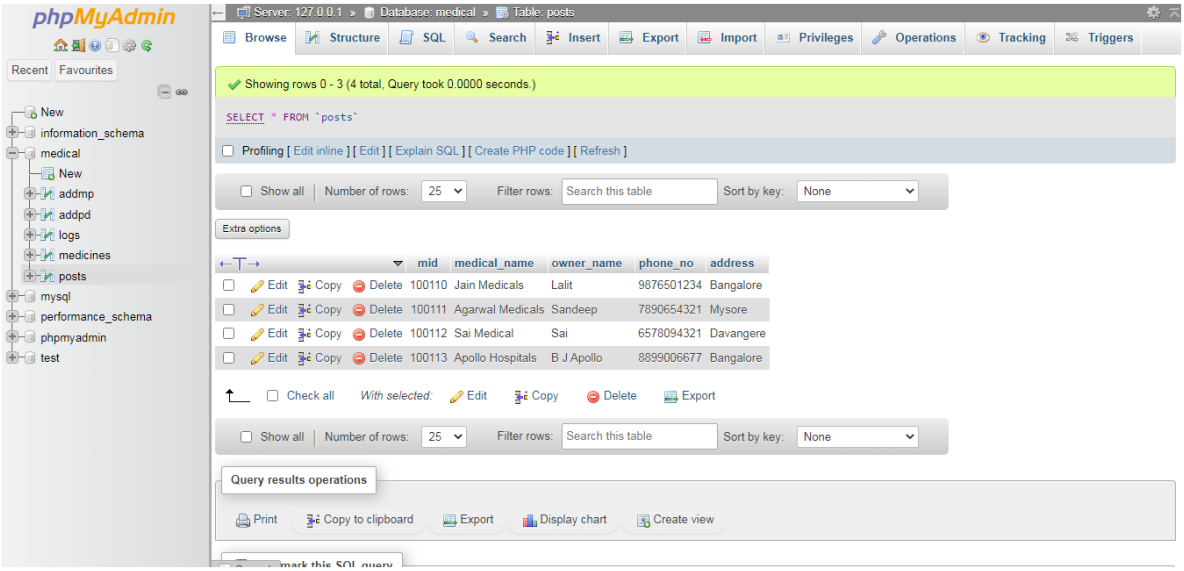
MYSQL Database in XAMPP Server:








phpMyAdmin
Recent Favourites
New
information_schema
medical
New
addmp
addpd
logs
medicines
posts
mysql
performance_schema
phpmyadmin
test
Server: 127.0.0.1 » Database: medical » Table: medicines
Browse Structure SQL Search Insert Export Import Privileges Operations Tracking Triggers
Showing rows 0 - 1 (2 total, Query took 0.0010 seconds.)
SELECT * FROM `medicines`
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
Extra options
id amount name medicines products email mid
Edit Copy Delete 3 250 Sai dolo 650 carpel 250mg cotton sai@gmail.com 100112
Edit Copy Delete 4 1699 B J Apollo paracetomol [30 pieces] eplixcrono [10 pieces] c... Dr Ortho knee cap Cotton Dettol 1L apollo@gmail.com 100113
Check all With selected: Edit Copy Delete Export
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
Query results operations
Print Copy to clipboard Export Display chart Create view


phpMyAdmin
Recent Favourites
New
information_schema
medical
New
addmp
addpd
logs
medicines
posts
mysql
performance_schema
phpmyadmin
test
Server: 127.0.0.1 » Database: medical » Table: posts
Browse Structure SQL Search Insert Export Import Privileges Operations Tracking Triggers
Showing rows 0 - 3 (4 total, Query took 0.0000 seconds.)
SELECT * FROM `posts`
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
Extra options
mid medical_name owner_name phone_no address
Edit Copy Delete 100110 Jain Medicals Lalit 9876501234 Bangalore
Edit Copy Delete 100111 Agarwal Medicals Sandeep 7890654321 Mysore
Edit Copy Delete 100112 Sai Medical Sai 6578094321 Davangere
Edit Copy Delete 100113 Apollo Hospitals B J Apollo 8899006677 Bangalore
Check all With selected: Edit Copy Delete Export
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
Query results operations
Print Copy to clipboard Export Display chart Create view

APPENDIX B

BACKEND PYHTON WITH MYSQL CODE

```
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=medic
               al_name,owner_name=owner_name,phone_no=phone_no,address=address)

                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=amount)
        db.session.add(entry)
        db.session.commit()
        flash("Data Added Successfully","primary")


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


app.run(debug=True)
```

FRONT END CODE

layout.html

```
<!DOCTYPE html>
<html lang="en">

<head>

  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">
  <meta name="description" content="">
  <meta name="author" content="">

  <title>{{params['blog_name']}}</title>

  <!-- Bootstrap core CSS -->
  <link href="{{url_for('static',filename='vendor/bootstrap/css/bootstrap.min.css')}}"
rel="stylesheet">

  <!-- Custom fonts for this template -->
  <link href="{{url_for('static',filename='vendor/fontawesome-
free/css/all.min.css')}}" rel="stylesheet" type="text/css">
  <link
href='https://fonts.googleapis.com/css?family=Lora:400,700,400italic,700italic'
rel='stylesheet' type='text/css'>
  <link
href='https://fonts.googleapis.com/css?family=Open+Sans:300italic,400italic,600italic,
700italic,800italic,400,300,600,700,800' rel='stylesheet' type='text/css'>

  <!-- Custom styles for this template -->
  <link href="{{url_for('static',filename='css/clean-blog.min.css')}}"
rel="stylesheet">

</head>

<body>

  <!-- Navigation -->
  <nav class="navbar navbar-expand-lg navbar-light fixed-top" id="mainNav">
    <div class="container">
      <a class="navbar-brand" href="#">{{params['blog_name']}}</a>
      <div class="collapse navbar-collapse" id="navbarResponsive">
        <ul class="navbar-nav ml-auto">
          <li class="nav-item">
            <a class="nav-link" href="/login">home</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="/insert">add medical information</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="/list">view ordered list</a>
```

```
          </li>
          <li class="nav-item">
            <a class="nav-link" href="/medicines ">Order medicines/products</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="/details ">Details</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="/search ">add/search items</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="/aboutus">about us</a>
          </li>
          <li class="nav-item">
            <a onclick="return confirm('Are you sure to logout?');" class="nav-link"
href="/logout">logout</a>
          </li>
        </ul>
      </div>
    </div>
  </nav>

  {% block body %}  {% endblock %}
  <hr>

  <!-- Footer -->
  <footer>
    <div class="container">
      <div class="row">
        <div class="col-lg-8 col-md-10 mx-auto">
          <ul class="list-inline text-center">
            <li class="list-inline-item">
              <a href="{{params['tw_url']}}" target="_blank">
                <span class="fa-stack fa-lg">
                  <i class="fas fa-times fa-stack-2x"></i> <!-- "X" icon -->
                </span>
              </a>
            </li>
            <li class="list-inline-item">
              <a href="{{params['fb_url']}}" target="_blank">
                <span class="fa-stack fa-lg">
                  <i class="fab fa-instagram fa-stack-2x"></i> <!-- Instagram icon -->
                </span>
              </a>
            </li>
            <li class="list-inline-item">
              <a href="{{params['gh_url']}}" target="_blank">
                <span class="fa-stack fa-lg">
                  <i class="fab fa-github fa-stack-2x "></i> <!-- GitHub icon -->
                </span>
              </a>
            </li>

        </div>
```

```
      </div>
    </div>
  </footer>

  <!-- Bootstrap core JavaScript -->
  <script src="{{url_for('static',filename='vendor/jquery/jquery.min.js')}}"></script>
  <script
src="{{url_for('static',filename='vendor/bootstrap/js/bootstrap.bundle.min.js')}}"></s
cript>

  <!-- Custom scripts for this template -->
  <script src="{{url_for('static',filename='js/clean-blog.min.js')}}"></script>

</body>

</html>
```

Medicine.html

```
{% extends "layout.html" %}
{% block body %}

  <!-- Page Header -->
  <header class="masthead mb-0" style="background-image:
url('{{url_for('static',filename='img/gmail.jpg')}}">
    <div class="overlay"></div>
    <div class="container">
      <div class="row">
        <div class="col-lg-8 col-md-10 mx-auto">
          <div class="page-heading">
            <h1> MEDICINES AND PRODUCTS </h1>
            <span class="subheading">Do you Have questions? I have answers</span>
          </div>
        </div>
      </div>
    </div>
  </header>
   {% with messages=get_flashed_messages(with_categories=true) %}
  {% if messages %}
  {% for category, message in messages %}

  <div class="alert alert-{{category}} alert-dismissible fade show" role="alert">
    {{message}}

  <button type="button" class="close" data-dismiss="alert" aria-label="Close">
    <span aria-hidden="true">&times;</span>
  </button>
</div>
  {% endfor %}
  {% endif %}
```

```
  {% endwith %}

  <!-- Main Content -->
  <div class="container">
    <div class="row">
      <div class="col-lg-8 col-md-10 mx-auto">
        <p>Want to get in touch? Fill out the form below to send me a message and I
will get back to you as soon as possible!</p>
        <form name="sentMessage" id="contactForm" novalidate action="/medicines"
method="post">
          <div class="control-group">
            <div class="form-group floating-label-form-group controls">
              <label>mid</label>
              <input type="number" name='mid' class="form-control" placeholder="enter
id" id="mid" required data-validation-required-message="Please enter id" >
              <p class="help-block text-danger"></p>
            </div>
          </div>
          <div class="control-group">
            <div class="form-group floating-label-form-group controls">
              <label>name</label>
              <input type="text" name="name" class="form-control" placeholder="Enter
name" id="name" required data-validation-required-message="Please enter name">
              <p class="help-block text-danger"></p>
            </div>
          </div>

            <div class="control-group">

            <div class="form-group floating-label-form-group controls">
              <label>Medicines</label>
              <textarea rows="5" class="form-control" placeholder="Enter medicines
names" id="medicines" name="medicines" required data-validation-required-
message="Please enter a medicines/products/syrups" ></textarea>
              <p class="help-block text-danger"></p>
            </div>
          </div>
          <div class="control-group">
            <div class="form-group floating-label-form-group controls">
              <label>Products</label>
              <textarea rows="5" class="form-control" placeholder="Enter products
names" id="products" name="products" required data-validation-required-message="Please
enter a medicines/products/syrups" ></textarea>
              <p class="help-block text-danger"></p>
            </div>
          </div>
          <div class="control-group">
            <div class="form-group floating-label-form-group controls">
              <label>Email id</label>
              <input type="email" name="email" class="form-control" placeholder="Enter
your mail id" id="email" required data-validation-required-message="Please enter
email">
              <p class="help-block text-danger"></p>
            </div>
```

```
          </div>
            <div class="control-group">
            <div class="form-group floating-label-form-group controls">
              <label>Amount</label>
              <input type="email" name="amount" class="form-control"
placeholder="Enter Amount" id="amount" required data-validation-required-
message="Please enter amount">
              <p class="help-block text-danger"></p>
            </div>
          </div>
          <br>
          <div id="success"></div>
          <div class="form-group">
            <button onclick="return confirm('Are you sure to SaveData?');"
type="submit" class="btn btn-primary" id="sendMessageButton" >Send</button>
          </div>
        </form>
      </div>
    </div>
  </div>

  <hr>

  <!-- Bootstrap core JavaScript -->
  <script src="vendor/jquery/jquery.min.js"></script>
  <script src="vendor/bootstrap/js/bootstrap.bundle.min.js"></script>

  <!-- Contact Form JavaScript -->
  <script src="js/jqBootstrapValidation.js"></script>
  <script src="js/contact_me.js"></script>

  <!-- Custom scripts for this template -->
  <script src="js/clean-blog.min.js"></script>

</body>

</html>
{% endblock %}
```