Contents

1.PROJECT SYNOPSIS

1.1 Introduction To Domain

This project concerned about developing a pharmacy management system that will be used for retail pharmacies. The purpose of this project is to manage all data derived for a pharmacy to maintain their business through the system rather than recording their data manually which is more risk to the business to maintain and to avoid loss. Most of them they record their data manually through book of accounts, this type of recording data it makes them to incurred more loss and they are not able to determine if they incurred loss or not for those who having a large stock of medicine. Due to this challenges which cause to minimize a business profit I became with solution on how they can reduce risk and maximize their profit through pharmacy management system.

1.2 Objectives

The overall objective of this project is to establish a system for pharmacy shops so as to improve the performance and efficiency of pharmacy shops management . in order to achieve this goal effectively, there are some specific objectives should be implemented:

The following are specific objectives for this project:

- To provide easily accessibility of customers management
- To provide easily accessibility of sales reports
- To provide easily accessibility of stock reports
- To provide easily accessibility of employees records etc.
- To minimize human errors.

1.3 Module Description

These are the main modules of the project:

- **Products module**: we can create, read, update and delete products from this module.
- **Customers module**: All the operations related to customers operations, is managed by this module.
- **Sells module**: it has been developed for managing sells.
- **Payment module**: It manages the payment.

1.4 Application

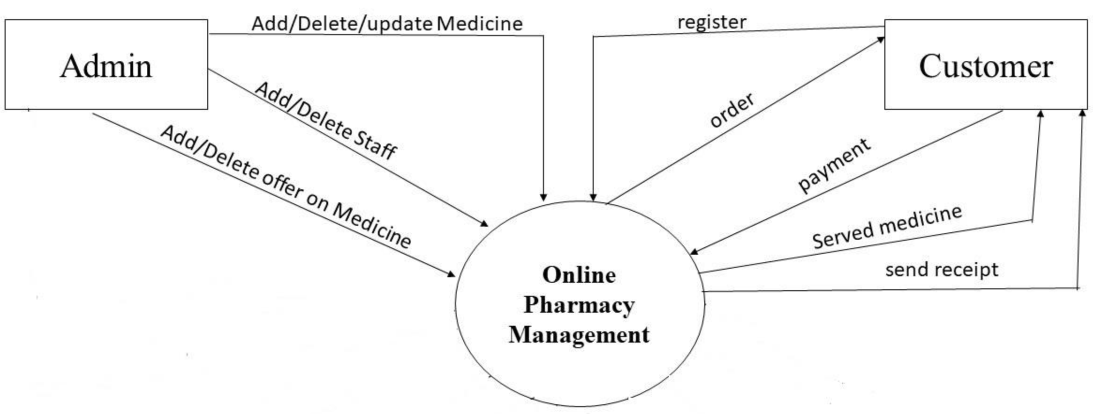
The user of this system is being able to manage all necessary activities of the pharmacy shop. The information management that provided by the system is a great advantage to reduce records errors associated with pharmacy shops.

Pharmacy management system covered the following areas:

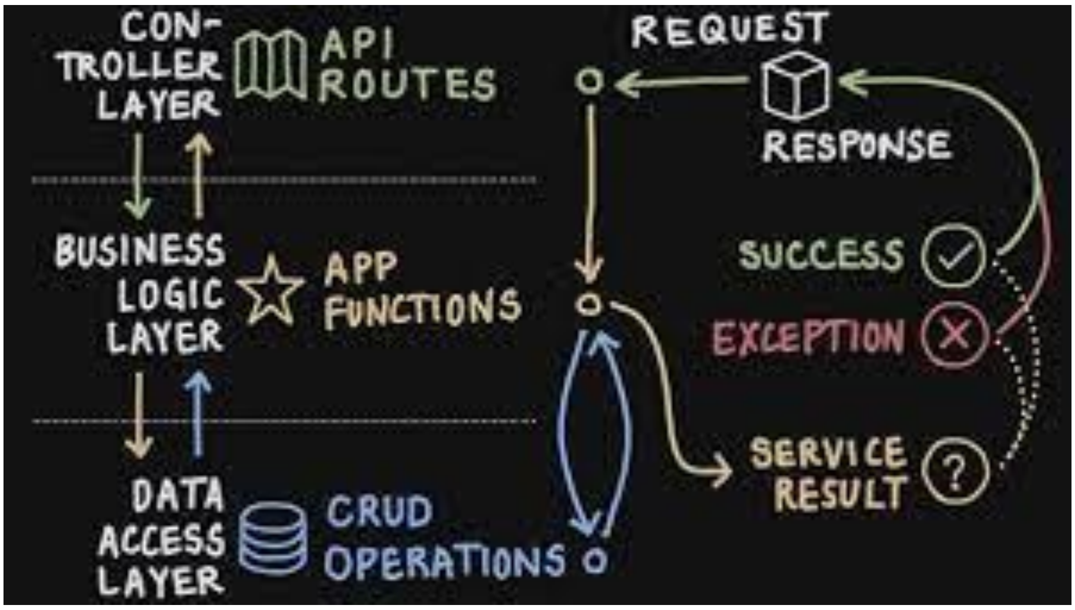
- Customers management
- Managers management
- Employees management
- Sales management
- Stock management

2. ARCHITECTURE DIAGRAM:

```
User Interface
├── Account Creation and Authentication
│     ├── User Registration
│     ├── Login
│     └── Account Management
│
├── Ordering Medicine
│     ├── View Medicine
│     ├── Add to cart
│     ├── Payment for the order
│     └── Ask for medicine through Prescription
│
├
│
└── Admin Control
     ├── User Management
     ├── Add Medicines
     ├── Edit Medicines
     └── View order
```


Admin
Customer
Online Pharmacy Management
Add/Delete/update Medicine
Add/Delete Staff
Add/Delete offer on Medicine
register
order
payment
Served medicine
send receipt

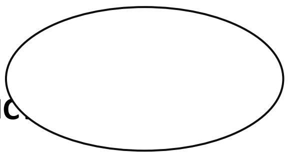
FAST-API FLOW :

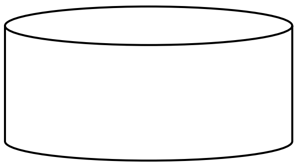


DATA FLOW DIAGRAM

DATA FLOW DIAGRAM NOTATION

FUNC



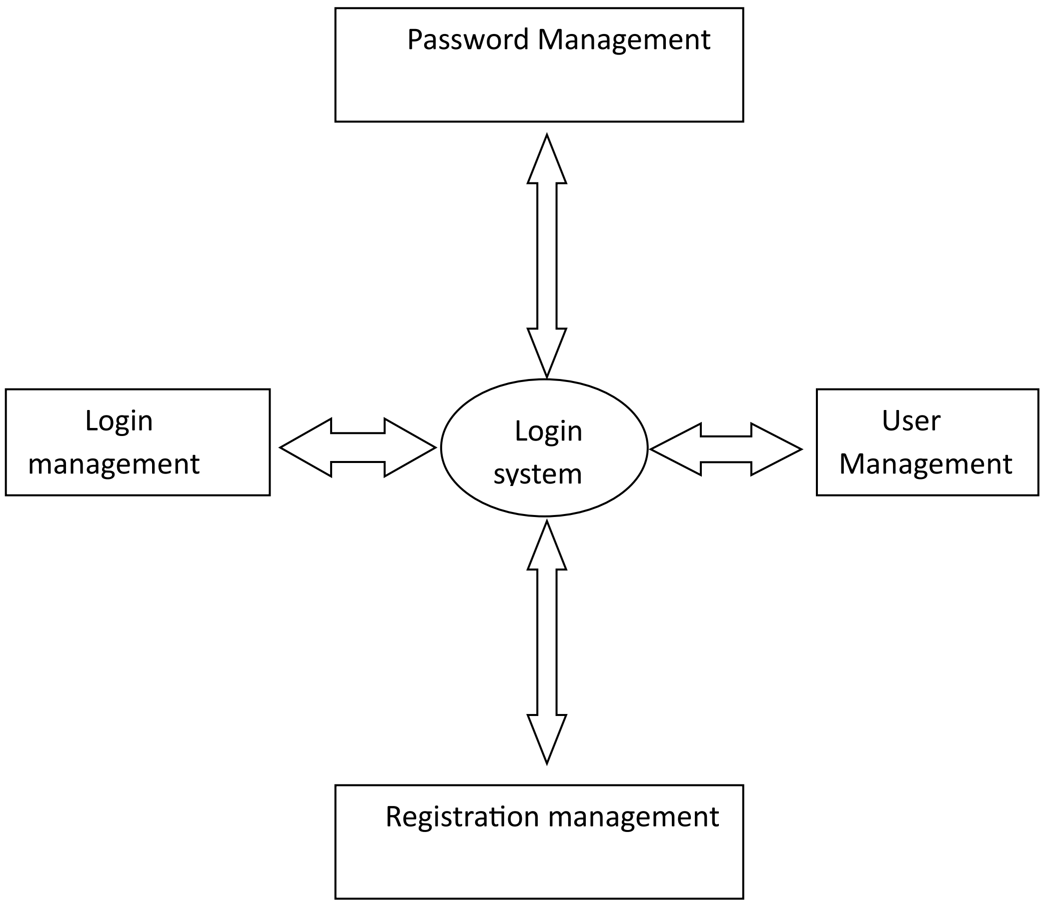


DATABASE

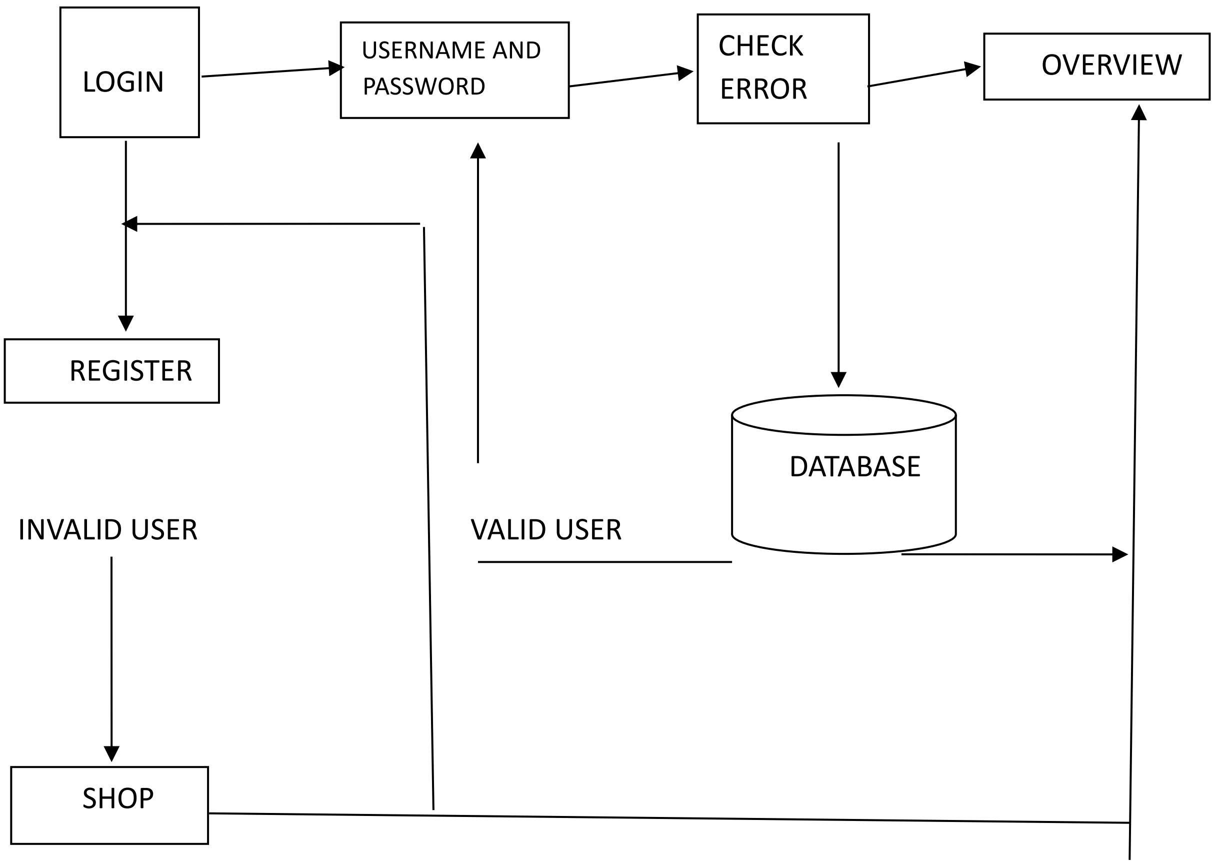
I

FLOW

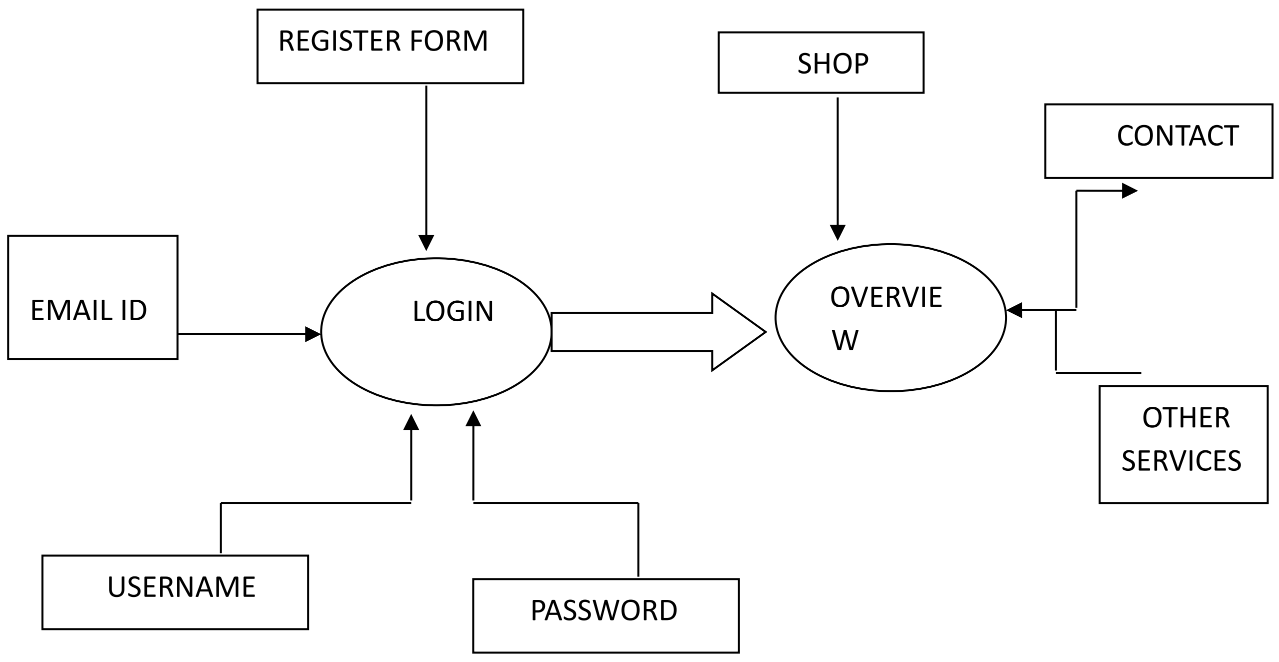
DFD LEVEL 0


Password Management
Login management
Login system
User Management
Registration management

DFD LEVEL 1


LOGIN
USERNAME AND PASSWORD
CHECK ERROR
OVERVIEW
REGISTER
DATABASE
INVALID USER
VALID USER
SHOP

DFD LEVEL0 1


REGISTER FORM
SHOP
CONTACT
EMAIL ID
LOGIN
OVERVIE W
OTHER SERVICES
USERNAME
PASSWORD

3. MODULAR DESCRIPTION IN DETAIL

The Pharmacy Store management project aims to develop a comprehensive system that facilitates the efficient management of medicine. This system consists of two main user roles: Admin and User. The Admin is responsible for managing medicines and orders. Users can view the medicines online and add to cart based on their needs.This detailed explanation will cover the functionalities and interactions of each user role in the system.

The online pharmacy management system consists of the following modular components for the admin:

1. User Management:
- Admin can manage user accounts, including registration, login, and profile management.
- Admin can view and update user information, such as contact details and delivery addresses.

2. Medicine Management:
- Admin can add new medicines to the system, including details like name, description, dosage, and price.
- Admin can update existing medicine information, such as stock availability and expiration dates.
- Admin can categorize medicines based on their type or medical condition.
- Admin can search and filter medicines based on various criteria, making it easier to find specific products.

3. Order Management:
- Admin can view and manage customer orders, including order status and payment details.
- Admin can update the order status, such as processing, shipped, or delivered.
- Admin can generate invoices and receipts for completed orders.
- Admin can handle order cancellations or modifications requested by customers.

4. Inventory Management:
- Admin can monitor and track the stock levels of medicines.
- Admin can receive notifications or alerts when stock levels are low or when certain medicines are about to expire.

4. TECHNICAL DESCRIPTION

The technical description of the Pharmacy Store Management project provides adetailed explanation of the underlying technology and architecture used to develop and implement the system. This section focuses on the technical aspects of the system, including the software, hardware, databases, and frameworks involved. The following points outline the key components of the technical description:

System Architecture

1. Client-Server Architecture: The Pharmacy Store Management followsa client-server architecture, where clients (web browsers or mobile applications) interact with the central server to request and process data. This architecture enables a distributed and scalable system.
2. Three-Tier Architecture: The system employs a three-tier architectureconsisting of the presentation layer, application layer, and data layer. The presentation layer handles user interfaces, the application layer contains the business logic, and the data layer manages the storage and retrieval of data.

Programming Languages and Frameworks:-

1. Front-End Development: The front-end of the system is developed using HTML, CSS. Additionally, frameworks such as React or Angular are used to enhance the user interface and provide a dynamic and responsive experience.

2. Back-End Development: The back-end of the system utilizes server-side programming languages such as Python, Node.js. These languages provide the necessary functionality to handle requests, process data, and interact with the database.

- Database Management:-

1. Database System: The Pharmacy Store Management employs a relational database management system (RDBMS) such as MySQL. The RDBMS stores and manages the data related to users, faculty members, visitor entries, appointments, and visitor history.

2. Database Design: The database is designed using appropriate entity-relationship modeling techniques, ensuring data integrity, normalization, and efficient query processing. Tables are created to store user information, faculty details, visitor entries, appointments, and other relevant data.

- Security Measures:-

1. User Authentication: The system incorporates secure user authentication mechanisms to verify the identity of administrators, faculty members, and visitors. This includes password hashing, encryption, and the use of secure protocols.

2. Access Control: Role-based access control (RBAC) is implemented to manage user permissions and restrict access to sensitive functionalities and data. This ensures that only authorized users can perform specific actions within the system.

3. Data Protection: Medicine data and sensitive information are securely stored and protected through encryption and access control mechanisms. Data encryption is employed during data transmission and storage to prevent unauthorized access and maintain data confidentiality.

Integration and APIs:-

1. Integration with External Systems: The Pharmacy Store Management may integrate with other existing systems within the organization, such as LDAP for user authentication or CRM systems to synchronizevisitor data.

2. APIs and Web Services: The system exposes APIs or web services to facilitate data exchange and integration with external systems or third-party applications. These APIs allow authorized systems to interact with the Pharmacy Store Management and retrieve or update relevant data.

Scalability and Performance:-

1. Caching Mechanisms: To improve performance, caching mechanisms like in-memory caching or content delivery networks (CDNs) can be utilized to store frequently accessed data closer to the users, reducing response times.

2. Load Balancing: To distribute incoming traffic and ensure high availability, load balancing techniques can be implemented. This involves distributing requests across multiple servers or using load balancing software or hardware.

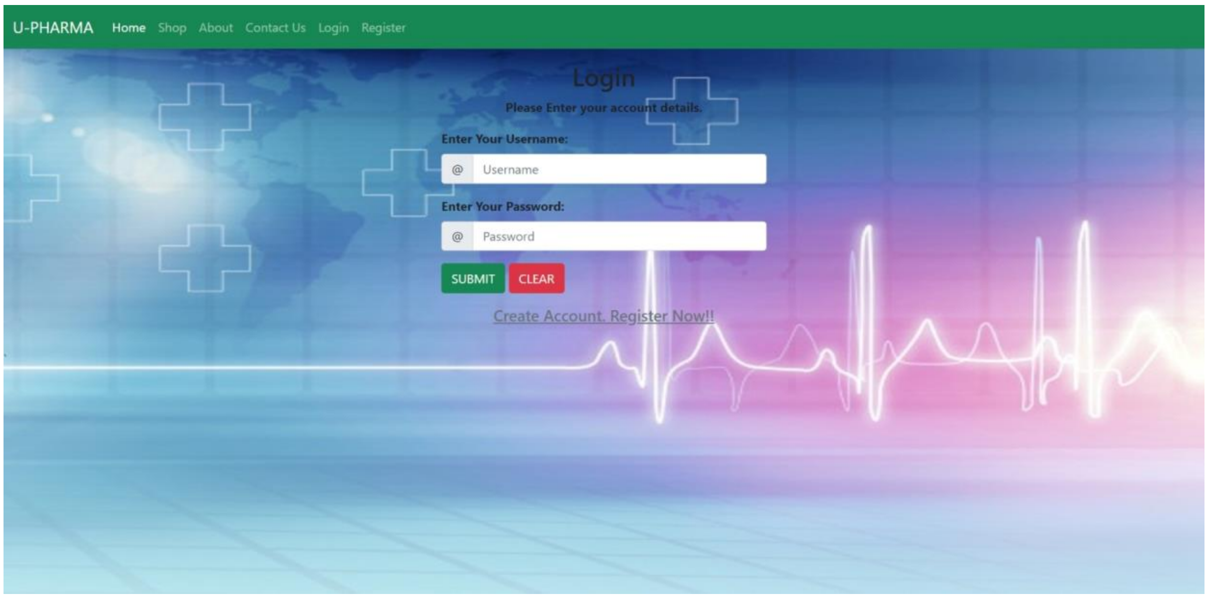
3. By implementing caching mechanisms and load balancing techniques, the Phamacy Store Managment can achieve improved performance and scalability. Caching frequently accessed data reduces database load and response times, while load balancing evenly distributes user requests to maintain high availability and efficient resource utilization. These measures contribute to a seamless user experience, increased system responsiveness, and the ability to handle growing user loads.

5.CODE SNIPPETS

Login Page

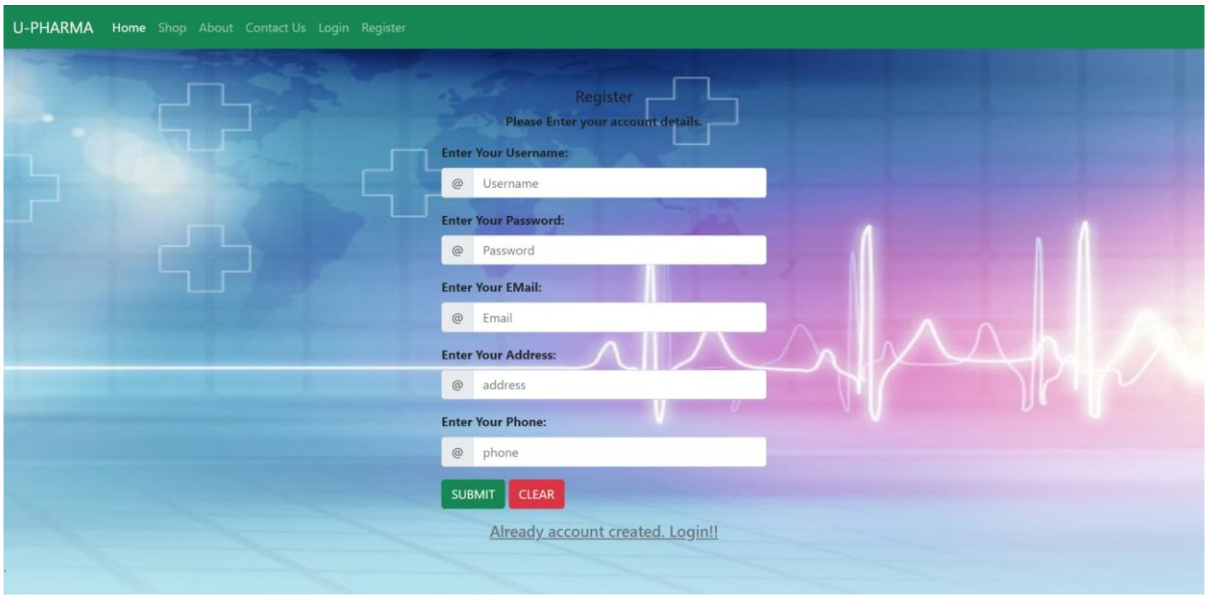
```
<form class="my-3" method="POST" action="/login">

          <div class="row">
                <div class="col-md-4"></div>

                <div class="col-md-4">
<b>


                            <h2 class="card-title text-center">Login</h2>
                            <p class="card-text text-center">Please Enter your
account details.</p>

                           <label for="username" class="form-label">Enter Your
Username:</label>
                            <div class="input-group mb-3">
                                               <span  class="input-group-text"
id="spusername">@</span>
                              <input type="text" id="username" name="username"
class="form-control" placeholder="Username"
                                aria-label="Username" aria-describedby="basic-
addon1" required>
                            </div>

                           <label for="password" class="form-label">Enter Your
Password:</label>
                            <div class="input-group mb-3">
                                               <span  class="input-group-text"
id="sppassword">@</span>
                                             <input  type="password"  id="password"
name="password" class="form-control" placeholder="Password"
                                aria-label="Password" aria-describedby="basic-
addon1" required>
                            </div>

                            <h5 class="text-danger">{{msg}}</h5>
                                             <button   class="btn   btn-success"
type="submit">SUBMIT</button>
```

```
                                              <button  class="btn  btn-danger"
type="reset">CLEAR</button>
                                <h5 class="text-center my-3"><a href="/register"
class="link-secondary">Create Account. Register Now!!</a> </h5>

                          </div>
                      </div>

                  </div>
              </b>


      </form>
```


Login
Please Enter your account details.
Enter Your Username:
Enter Your Password:

Connecting to Database

```
@app.post("/review", response_class=HTMLResponse)
def review_page(request: Request, mreviews:str = Form(...)):
    with sqlite3.connect("app.db") as con:
        cur = con.cursor()
                    review_query    =    "INSERT    into    reviews(review)
values('{}')".format(mreviews)
        cur.execute (review_query)
        con.commit()
        return RedirectResponse("/", status_code=status.HTTP_302_FOUND)
app.post("/register", response_class=HTMLResponse)
def do_register(request: Request, username: str = Form(...), password: str =
Form(...), email: str = Form(...),
                address: str = Form(...), phone: str = Form(...)):
    with sqlite3.connect(DATABASE_NAME) as con:
        cur = con.cursor()
          cur.execute("INSERT into users(username, password, email, address,
phone) values(?,?,?,?,?)",
                    (username, password, email, address, phone))
        con.commit()
    return RedirectResponse("/login", status_code=status.HTTP_302_FOUND)
```



Register

Please Enter your account details.

Enter Your Username: Username

Enter Your Password: Password

Enter Your EMail: Email

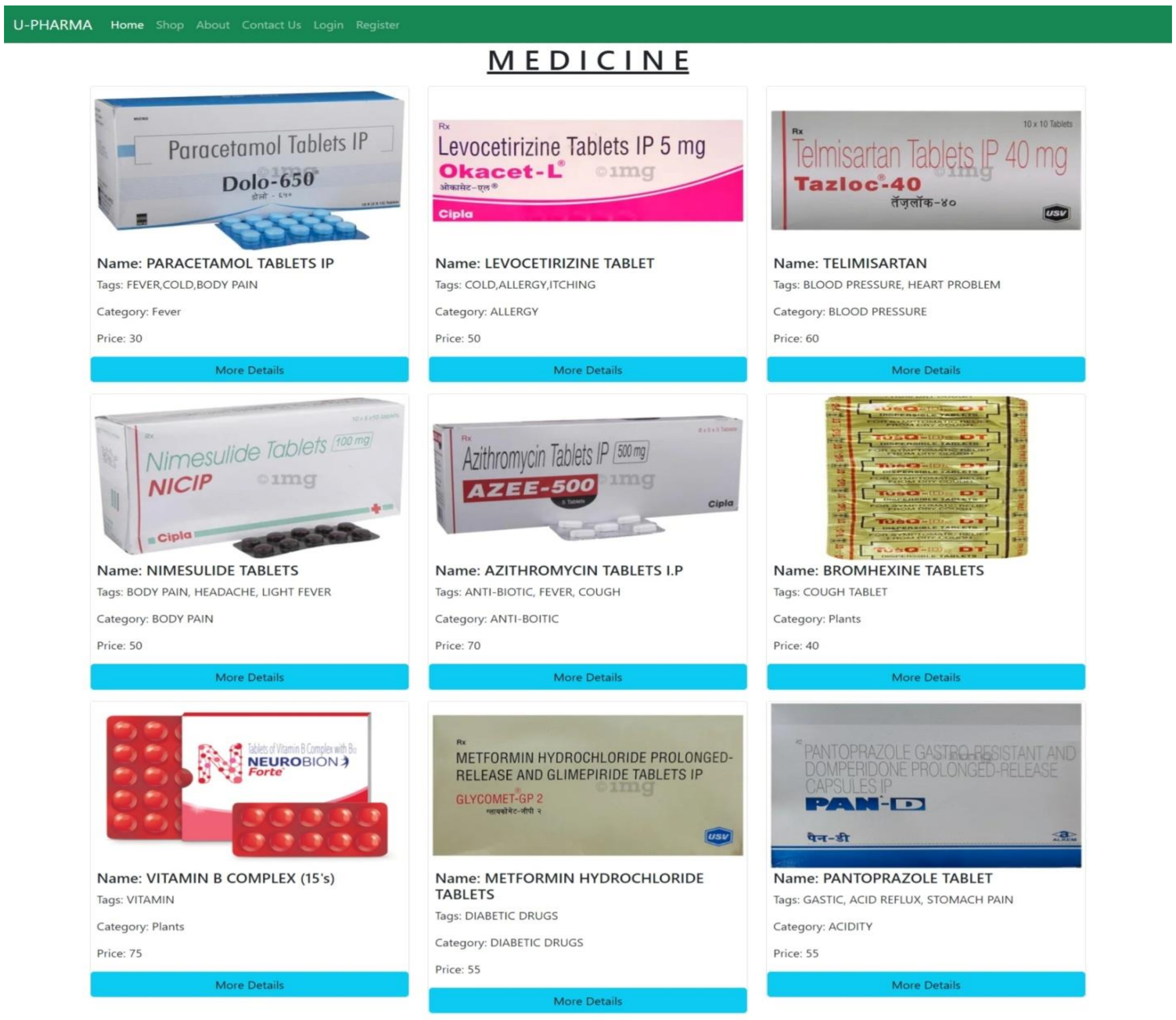
Enter Your Address: address

Enter Your Phone: phone

SUBMIT CLEAR

Already account created. Login!!

```
@app.get("/shop", response_class=HTMLResponse)
def shop(request: Request):
    con = sqlite3.connect(DATABASE_NAME)
    con.row_factory = sqlite3.Row
    cur = con.cursor()
    cur.execute("select * from products")
    plants = cur.fetchall()
    con.close
      return  templates.TemplateResponse("shop.html",  {"request":  request,
"plants": plants})
```



MEDICINE

Name: PARACETAMOL TABLETS IP
Tags: FEVER,COLD,BODY PAIN
Category: Fever
Price: 30
More Details

Name: LEVOCETIRIZINE TABLET
Tags: COLD,ALLERGY,ITCHING
Category: ALLERGY
Price: 50
More Details

Name: TELIMISARTAN
Tags: BLOOD PRESSURE, HEART PROBLEM
Category: BLOOD PRESSURE
Price: 60
More Details

Name: NIMESULIDE TABLETS
Tags: BODY PAIN, HEADACHE, LIGHT FEVER
Category: BODY PAIN
Price: 50
More Details

Name: AZITHROMYCIN TABLETS I.P
Tags: ANTI-BIOTIC, FEVER, COUGH
Category: ANTI-BOITIC
Price: 70
More Details

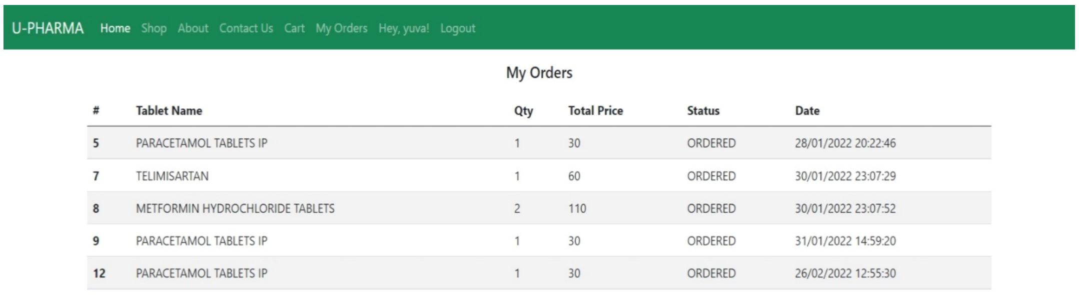
Name: BROMHEXINE TABLETS
Tags: COUGH TABLET
Category: Plants
Price: 40
More Details

Name: VITAMIN B COMPLEX (15's)
Tags: VITAMIN
Category: Plants
Price: 75
More Details

Name: METFORMIN HYDROCHLORIDE TABLETS
Tags: DIABETIC DRUGS
Category: DIABETIC DRUGS
Price: 55
More Details

Name: PANTOPRAZOLE TABLET
Tags: GASTIC, ACID REFLUX, STOMACH PAIN
Category: ACIDITY
Price: 55
More Details

```
@app.get("/addtocart", response_class=HTMLResponse ()
        cur.execute("INSERT into carts(pid, qty, uid) values(?,?,?)",
                    (pid, qty, uid))
        con.commit()
    return RedirectResponse("/cart", status_code=status.HTTP_302_FOUND)
@app.get("/cart", response_class=HTMLResponse)
def cart(request: Request):
    if not request.session.get('isLogin'):
        return RedirectResponse('/login', status_code=status.HTTP_302_FOUND)

    uid = request.session.get('uid')

    con = sqlite3.connect(DATABASE_NAME)
    con.row_factory = sqlite3.Row
    cur = con.cursor()
    cur.execute("SELECT *,c.id as cid from USERS u,carts c, products p where
u.id=c.uid and c.pid=p.id and c.uid =?", [uid])
    items = cur.fetchall()
    total = sum(map(lambda item: int(item['qty']) * int(item['price']), items))
    con.close
      return  templates.TemplateResponse("/cart.html",  {"request":  request,
"items": items, "total" :total})
```



My Orders

#	Tablet Name	Qty	Total Price	Status	Date
5	PARACETAMOL TABLETS IP	1	30	ORDERED	28/01/2022 20:22:46
7	TELIMISARTAN	1	60	ORDERED	30/01/2022 23:07:29
8	METFORMIN HYDROCHLORIDE TABLETS	2	110	ORDERED	30/01/2022 23:07:52
9	PARACETAMOL TABLETS IP	1	30	ORDERED	31/01/2022 14:59:20
12	PARACETAMOL TABLETS IP	1	30	ORDERED	26/02/2022 12:55:30

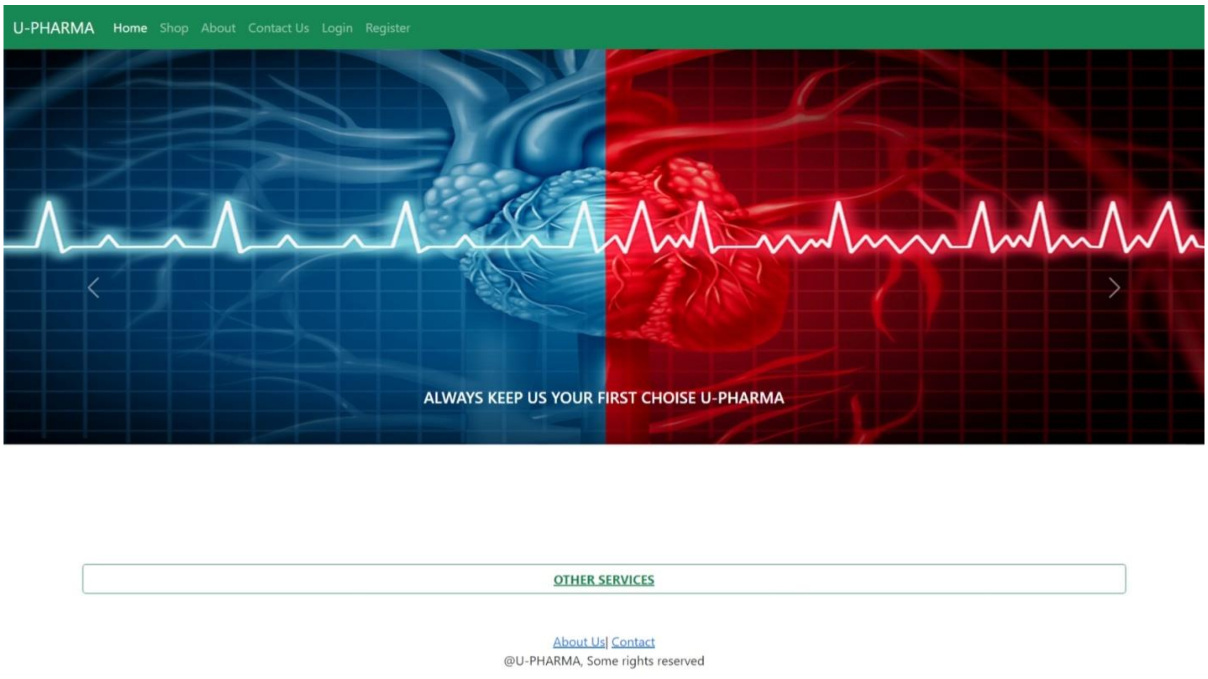
Importing Modules

```
from ast import For
from datetime import datetime
import re
from turtle import st
from fastapi import APIRouter
from fastapi import FastAPI, Request, Cookie
from fastapi.params import Form
from fastapi.responses import HTMLResponse
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
import sqlite3
import starlette.status as status
from starlette.middleware.sessions import SessionMiddleware
from starlette.responses import RedirectResponse, Response
from fastapi.security import HTTPBasic, HTTPBasicCredentials
import products
import user
import cart
```

Error handling

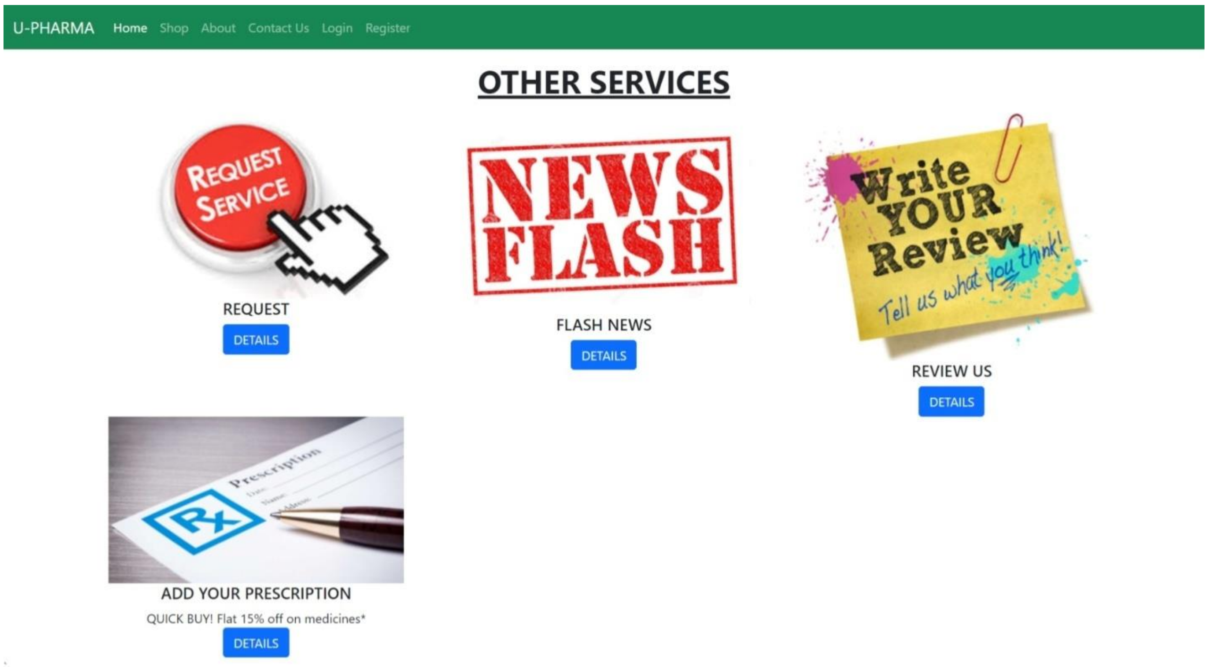
```
 @router.post("/login", response_class=HTMLResponse)
     def  do_login(request:  Request,  response:  Response,  username:  str  =
Form(...), password: str = Form(...)):
        con = sqlite3.connect(DATABASE_NAME)
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        user = cur.fetchone()
        if not user:
              raise  HTTPException(status_code=404,  detail="Wrong  Login  id  or
password.Contact Admin")
                return  templates.TemplateResponse("/login.html",  {"request":
request, "msg": "Invalid Username or Password"})
        else:
            request.session.setdefault("isLogin", True)
            request.session.setdefault('username', user['username'])
            request.session.setdefault('uid', user['id'])
            return RedirectResponse("/", status_code=status.HTTP_302_FOUND)
```

Home Page



Main Module

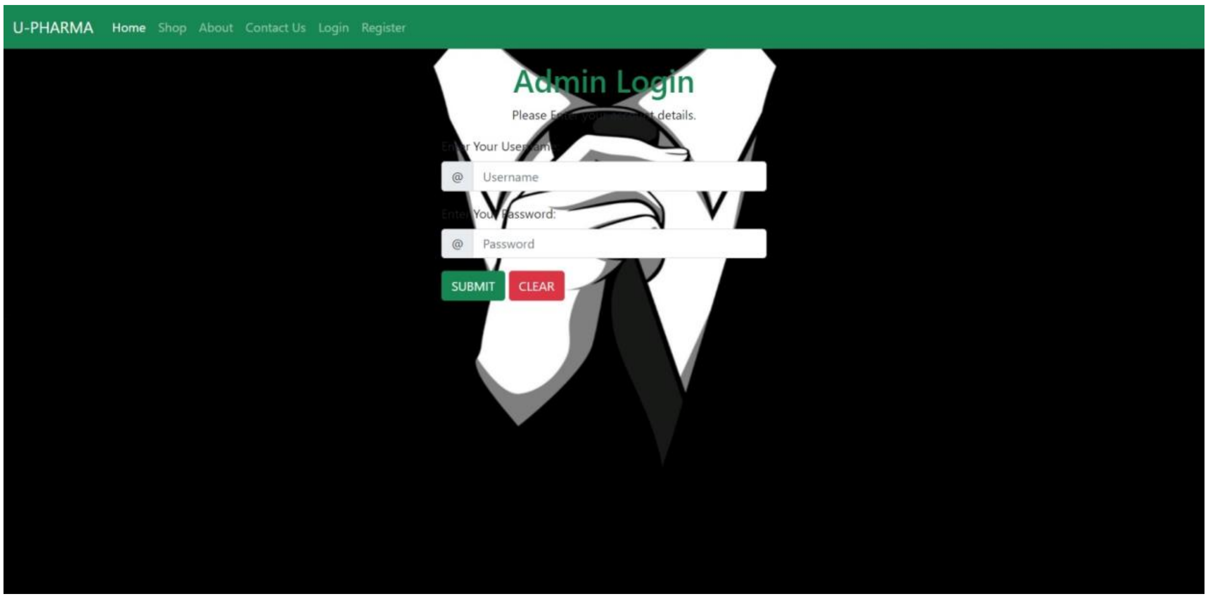
```
router = APIRouter()
security = HTTPBasic()

# configuring the static, which serve static
router.mount("/static", StaticFiles(directory="static"), name="static")


# adding the Session Middleware
#router.add_middleware(SessionMiddleware, secret_key='MyApp')

# configuring the HTML pages
templates = Jinja2Templates(directory="templates")

#constant name for DATABASE_NAME
DATABASE_NAME = "app.db"

class user:

    @router.get("/register",response_class=HTMLResponse)
    def register(request: Request):
             return  templates.TemplateResponse("register.html",  {"request":
request})


    @router.post("/register", response_class=HTMLResponse)
    def do_register(request: Request, username: str = Form(...), password: str
= Form(...), email: str = Form(...),
                    address: str = Form(...), phone: str = Form(...)):
        with sqlite3.connect(DATABASE_NAME) as con:
            cur = con.cursor()
            cur.execute("INSERT into users(username, password, email, address,
phone) values(?,?,?,?,?)",
                        (username, password, email, address, phone))
            con.commit()
        return RedirectResponse("/login", status_code=status.HTTP_302_FOUND)


    @router.get("/login",response_class=HTMLResponse)
    def login(request: Request):
        return templates.TemplateResponse("login.html", {"request": request})


    @router.post("/login", response_class=HTMLResponse)
      def  do_login(request:  Request,  response:  Response,  username:  str  =
Form(...), password: str = Form(...)):
        con = sqlite3.connect(DATABASE_NAME)
        con.row_factory = sqlite3.Row
        cur = con.cursor()
         cur.execute("select  *  from  users  where  username  =?  and  password=?",
[username, password])
        user = cur.fetchone()
        if not user:
              raise  HTTPException(status_code=404,  detail="Wrong  Login  id  or
password.Contact Admin")
                return  templates.TemplateResponse("/login.html",  {"request":
request, "msg": "Invalid Username or Password"})
        else:
            request.session.setdefault("isLogin", True)
            request.session.setdefault('username', user['username'])
            request.session.setdefault('uid', user['id'])
            return RedirectResponse("/", status_code=status.HTTP_302_FOUND)
```



Admin module

```
router = APIRouter()
security = HTTPBasic()

# configuring the static, which serve static
router.mount("/static", StaticFiles(directory="static"), name="static")


# adding the Session Middleware
#router.add_middleware(SessionMiddleware, secret_key='MyApp')

# configuring the HTML pages
templates = Jinja2Templates(directory="templates")

#constant name for DATABASE_NAME
DATABASE_NAME = "app.db"

class admins:

    @router.get("/admin/", response_class=HTMLResponse)
    def admin_index(request: Request):
          return  templates.TemplateResponse("/admin/index.html",  {"request":
request})


    @router.post("/admin/", response_class=HTMLResponse)
    def admin_index(request: Request, username: str = Form(...), password: str
= Form(...)):
        con = sqlite3.connect(DATABASE_NAME)
        con.row_factory = sqlite3.Row
        cur = con.cursor()
         cur.execute("select * from admin where username =? and password=?",
[username, password])
        admin = cur.fetchone()
        if not admin:
            return templates.TemplateResponse("/admin/index.html", {"request":
request, "msg": "Invalid Username or Password"})
        else:
            request.session.setdefault("isLogin", True)
            request.session.setdefault('username', admin['username'])
            request.session.setdefault('uid', admin['id'])
            request.session.setdefault('role', admin['role'])
                                return    RedirectResponse("/admin/dashboard",
status_code=status.HTTP_302_FOUND)


    @router.get("/admin/dashboard", response_class=HTMLResponse)
    def dashboard(request: Request):
        con = sqlite3.connect(DATABASE_NAME)
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        cur.execute("select * from products")
        products = cur.fetchall()
        con.close
        return templates.TemplateResponse("/admin/dashboard.html", {"request":
request, "items": products})

    @router.get("/promo", response_class=HTMLResponse)
    def promo(request: Request):
          return  templates.TemplateResponse("/admin/promo.html",  {"request":
request})
```

```
    @router.get("/admin/products", response_class=HTMLResponse)
    def admin_products(request: Request):
        con = sqlite3.connect(DATABASE_NAME)
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        cur.execute("select * from products")
        products = cur.fetchall()
        con.close
        return templates.TemplateResponse("/admin/products.html", {"request": request, "products": products})


    @router.get("/admin/products/create", response_class=HTMLResponse)
    def admin_products_create(request: Request):
            return  templates.TemplateResponse("/admin/products_create.html", {"request": request})


    @router.post("/admin/products/create", response_class=HTMLResponse)
    def admin_products_create(request: Request, pname:str = Form(...), price: str = Form(...), image: str = Form(...), details: str = Form(...), tags: str = Form(...), category:str = Form(...)):
        with sqlite3.connect(DATABASE_NAME) as con:
            cur = con.cursor()
           cur.execute("INSERT into products(name, price, details, image, tags, category) values(?, ?, ?, ?, ?, ?)",
                        (pname, price, details, image, tags, category))
            con.commit()
        return RedirectResponse("/admin/products",status_code=status.HTTP_302_FOUND)


    @router.get("/admin/orders", response_class=HTMLResponse)
    def admin_orders(request: Request):
        con = sqlite3.connect(DATABASE_NAME)
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        cur.execute("SELECT *, o.id as oid from users u, products p, orders o where o.uid = u.id and o.pid = p.id")
        orders = cur.fetchall()
        con.close
          return  templates.TemplateResponse("/admin/orders.html",  {"request":
request, "orders": orders})


    @router.get("/admin/logout", response_class=HTMLResponse)
    def admin_logout(request: Request):
              return   templates.TemplateResponse("/admin/logout",   {"request":
request})


    @router.get("/admin/products_edit/{pid}", response_class=HTMLResponse)
    def admin_product_edit(request: Request, pid: int = 0):
               return   templates.TemplateResponse("/admin/products_edit.html",
{"request": request})


    @router.get("/admin/products_delete/{pid}", response_class=HTMLResponse)
    def admin_product_delete(request: Request, pid: int = 0):
                               return       RedirectResponse("/admin/products",
status_code=status.HTTP_302_FOUND)


    @router.get("/admin/orders", response_class=HTMLResponse)
    def admin_orders(request: Request):
          return  templates.TemplateResponse("/admin/orders.html",  {"request":
request})


    @router.get("/admin/orders_view/{oid}", response_class=HTMLResponse)
    def admin_order_view(request: Request, oid: int = 0):
                return   templates.TemplateResponse("/admin/orders_view.html",
{"request": request})
```



Admin Login

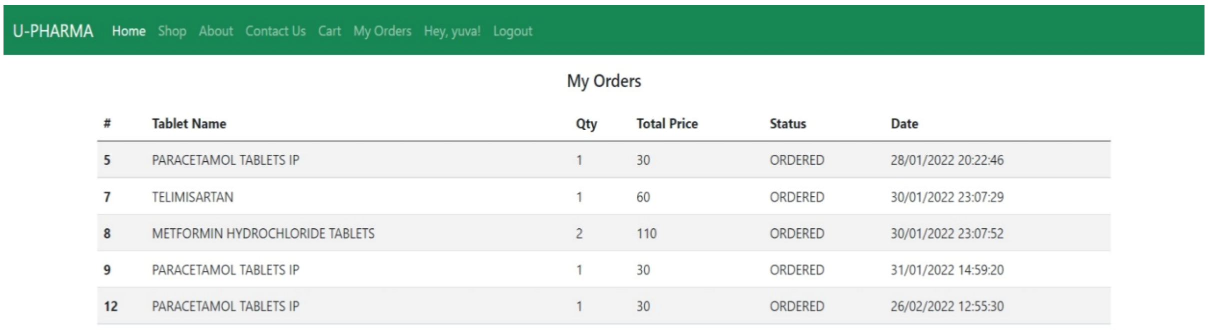
Please Enter your account details.

Enter Your Username:

@ Username

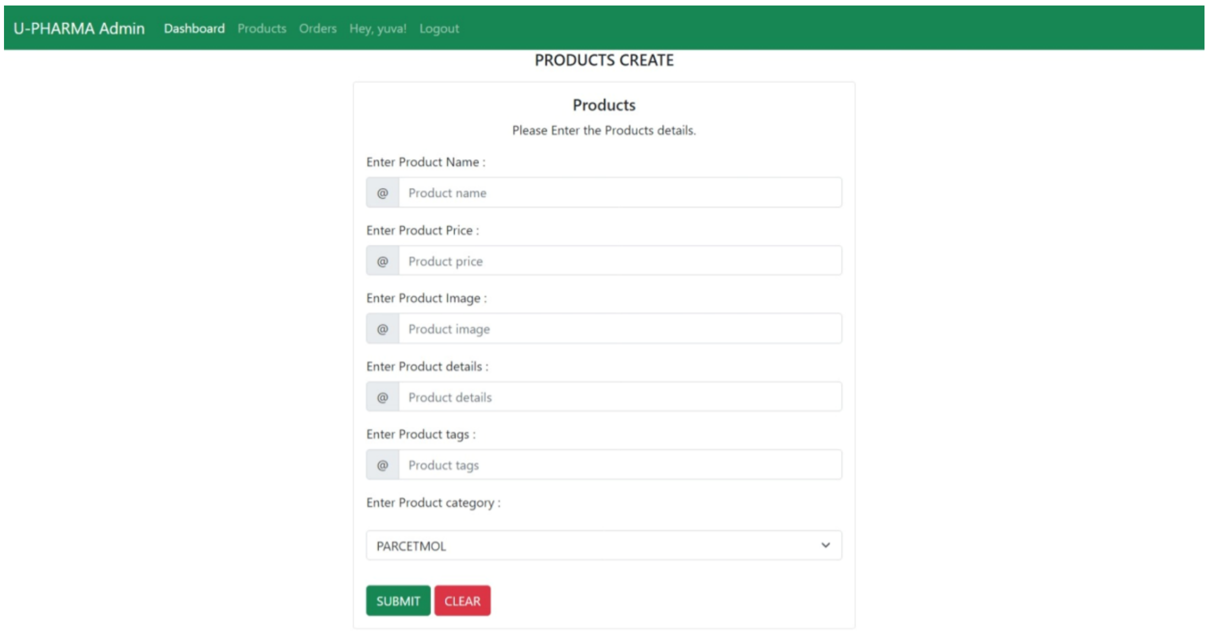
Enter Your Password:

@ Password



My Orders

#	Tablet Name	Qty	Total Price	Status	Date
5	PARACETAMOL TABLETS IP	1	30	ORDERED	28/01/2022 20:22:46
7	TELIMISARTAN	1	60	ORDERED	30/01/2022 23:07:29
8	METFORMIN HYDROCHLORIDE TABLETS	2	110	ORDERED	30/01/2022 23:07:52
9	PARACETAMOL TABLETS IP	1	30	ORDERED	31/01/2022 14:59:20
12	PARACETAMOL TABLETS IP	1	30	ORDERED	26/02/2022 12:55:30

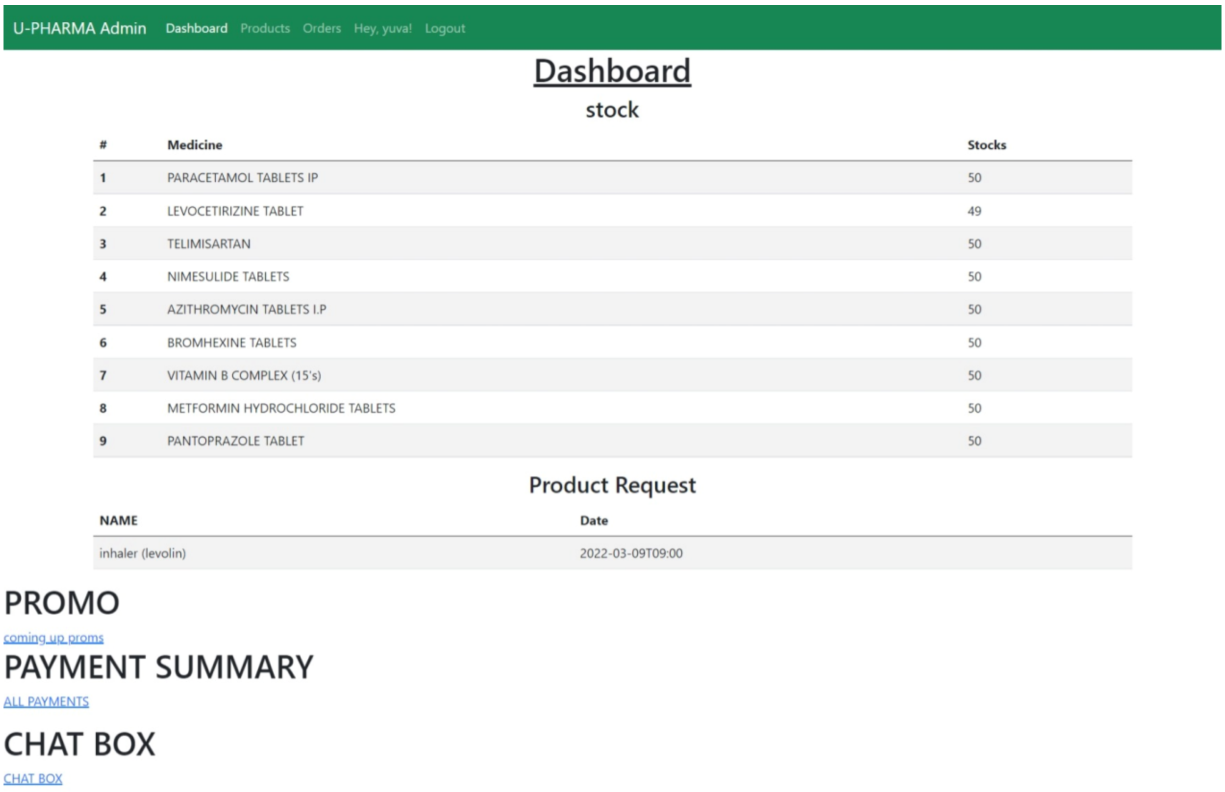


PRODUCTS CREATE

Products

Please Enter the Products details.

Enter Product Name :	@ Product name
Enter Product Price :	@ Product price
Enter Product Image :	@ Product image
Enter Product details :	@ Product details
Enter Product tags :	@ Product tags
Enter Product category :	PARCETMOL



Dashboard

stock

#	Medicine	Stocks
1	PARACETAMOL TABLETS IP	50
2	LEVOCETIRIZINE TABLET	49
3	TELIMISARTAN	50
4	NIMESULIDE TABLETS	50
5	AZITHROMYCIN TABLETS I.P	50
6	BROMHEXINE TABLETS	50
7	VITAMIN B COMPLEX (15's)	50
8	METFORMIN HYDROCHLORIDE TABLETS	50
9	PANTOPRAZOLE TABLET	50

Product Request

NAME	Date
inhaler (levolin)	2022-03-09T09:00

PROMO

coming up proms

PAYMENT SUMMARY

ALL PAYMENTS

CHAT BOX

CHAT BOX

6. CONCLUSION

UPharmacy management system is actually a software which handles the essential data of medicines and actually about the database of a pharmacy and its management. It provides the statistics about medicines or drugs which are in stocks which data can also be updated and edited. It works as per the requirements of the user and have options accordingly. This software also has ability to provide offers on medicines etc. there is other function available too. The main purpose is effectively and easily handling of pharmacy data and its management. We've learnt a lot through this project. This project has sharpened our concept of software-hardware interface. We've learnt a lot about different documentation. This project has not only tested our technical skills but has also tested our temperament. There were times that we lost hope but we recovered through constant concentration and hard work. If you have any kind of suggestion, improvements, more efficient development idea please feel free to communicate with us.

FUTU RE ENHANCEMENT

- DEALING WITH PRESCRIPTIONS
- CASH ON DELIVERY MODE

REFERENCES

Books:

1. “Learning Python” by Mark Lutz
2. “Python: Python Programming For Beginners” by Adam Stark

Reference Links:

1.Python Weekly is a curated newsletter that delivers the latest news, articles, tutorials, and releases related to Python. It's a great resource to stay up to date with Python community.

https://www.pythonweekly.com/

2.PyPi is the official repository for Python packages. It hosts a vast collection of open-source Python libraries and modules. You can search for specific packages and find their documentation, usage examples, and installation instructions -- https://pypi.org/

3."Python Django Web Framework - Full Course for Beginners" by freeCodeCamp.org
– [https://www.youtube.com/watch?v=E5mRWOjo-U4]
-- [https://www.youtube.com/watch?y=E5mRWOjo-U4]

4."Django Tutorials" by thenewboston :
-- [https://www.youtube.com/playlist?list=PL6gx4Cw19DGBImzz.FcL.gDhKTTINLIX11K]
-- (https:// www.youtube.com/playlist?list-PL.6gx4Cw|9DGBImzzFcLgDhKTTINLEXIIK)