

K L UNIVERSITY
FRESHMAN ENGINEERING DEPARTMENT
A Project Based Lab Report
On
PHARMA STORE MANAGEMENT SYSTEM

SUBMITTED BY:

I.D NUMBER	NAME
2200080080	G. Jashwanth sai
2200080177	A. Harshitha
2200080079	P. Sai Charan
2200080107	k. Manogna
2200080128	Y. Pooja

UNDER THE ESTEEMED GUIDANCE OF

Dr. R. MEGANATHAN
ASSOCIATE PROFESSOR



KL UNIVERSITY
Green fields, Vaddeswaram – 522
502Guntur Dt., AP, India.

DEPARTMENT OF BASIC ENGINEERING SCIENCES



CERTIFICATE

This is to certify that the project based laboratory report entitled “**PHARMA STORE MANAGEMENT SYSTEM**” submitted by bearing 2200080080 (G. Jashwanth sai), 2200080177 (A. Harshitha), 2200080079 (P. Sai Charan), 2200080107 (k. Manogna),2200080128 (Y.Pooja)to the **Department of Basic Engineering Sciences, KL University** in partial fulfillment of the requirements for the completion of a project in OBJECT ORIENTED PROGRAMMING SYSTEM – 22AD1202 course in I B Tech II Semester, is a bonafide record of the work carried out by him/her under my supervision during the academic year 2022-23.

PROJECT SUPERVISOR

HEAD OF THE DEPARTMENT

Dr. R. MEGANATHAN

Dr. D. HARITHA

ACKNOWLEDGEMENTS

It is great pleasure for me to express my gratitude to our honourable President **Sri. Koneru Satyanarayana**, for giving the opportunity and platform with facilities in accomplishing the project-based laboratory report.

I express the sincere gratitude to our director **Dr. A Jagadeesh** for his administration towards our academic growth.

I express sincere gratitude to our Coordinators and HOD-BES **Dr. D. Haritha** for her leadership and constant motivation provided in successful completion of our academic semester. I record it as my privilege to deeply thank for providing us the efficient faculty and facilities to make our ideas into reality.

I express my sincere thanks to our project supervisor **Dr. R. Meganathan** for his novel association of ideas, encouragement, appreciation, and intellectual zeal which motivated us to venture this project successfully.

Finally, it is pleased to acknowledge the indebtedness to all those who devoted themselves directly or indirectly to make this project report success.

I.D NUMBER	NAME
2200080080	G. Jashwanth sai
2200080177	A. Harshitha
2200080079	P. Sai Charan
2200080107	k. Manogna
2200080128	Y. Pooja

ABSTRACT

A Pharma Store Management System project that utilizes object-oriented programming concepts in Python to simplify and automate the management of medical stores. The project comprises four classes: Product, Customer, Order, and Main, which work together to provide a user-friendly interface for managing product information, customer details, orders, and invoices. The system is designed to reduce time consumption and human effort, allowing pharmacy store managers to easily record and manage all activities of the store, including stock inventory and documentation. The project is useful for small to medium-sized pharmacy stores and demonstrates the application of object-oriented programming concepts to solve real-world problems. Overall, the Pharma Store Management System project is a great example of how technology can improve business processes in the healthcare sector.

INDEX

S.NO	TITLE	Pg.no
1	Introduction	1
2	Requirements	2
3	Features	3
4	Class Diagram	4
5	Classes	5
6	Implementation	8
7	Result	27
8	Conclusion	30

INTRODUCTION

Aim of the project:

The aim is to develop an application that simplifies the inventory management process and reduces manual effort and time consumption in medical stores.

Objectives:

1. To provide a user-friendly interface for managing the inventory of medical stores.
2. To reduce manual effort and time consumption in maintaining the inventory and documentation of medical stores.
3. To keep track of product information, including stock levels, expiry dates, and prices.
4. To store customer information, including contact details and purchase history.
5. To enable the placement of orders and generate invoices for customers.
6. To provide reporting and analytics features to help store managers make informed decisions about inventory management.

Scope:

The Pharma Store Management System project includes the development of an application that can be used by small to medium-sized medical stores to manage their inventory and customer information. The application will provide a user-friendly interface for adding, updating, and deleting product information, placing orders, and generating invoices. The system will keep track of stock levels, expiry dates, and prices, and allow users to search and filter products by different attributes. The application will also store customer information, including contact details and purchase history, and provide reporting and analytics features to help store managers make informed decisions about inventory management. The system will ensure the security of the data through user authentication and database integration. The project can be extended to include additional features and integrations with other systems, such as a barcode scanner or an online store platform.

REQUIREMENTS

Hardware requirements:

1. Processor: Intel Core i3 or equivalent
2. RAM: 4 GB or more
3. Storage: 128 GB SSD or more
4. Display: 14-inch or larger with a resolution of 1366x768 or higher
5. Input devices: keyboard and mouse

Software requirements:

1. Operating System: Windows 10, macOS, or Linux
2. Python 3.x: The project will be developed using the Python programming language, so it is necessary to have Python installed on the system.
3. Integrated Development Environment (IDE): An IDE such as PyCharm, Visual Studio Code, or Spyder can be used to develop the project.

Other requirements:

1. User interface design: Make sure the GUI is user-friendly and intuitive. It should be easy for the user to navigate and perform the necessary tasks. Consider using icons, colours, and clear labelling to make the interface more visually appealing and easier to use.
2. Input validation: Validate all user input to ensure that only valid data is stored in the database. This will help prevent errors and ensure that the system runs smoothly.
3. Error handling: Implement error handling for all possible errors that could occur during the program execution. This includes database connection errors, file saving errors, and any other unexpected errors.

FEATURES

1. **Product Management:** Add, update, and delete product information, view available products with expiry date, price, and quantity.
2. **Customer Management:** Add new customers, update existing customer details, and view the list of all customers.
3. **Order Management:** Place orders, view order history, and generate invoices for customers. View order details such as product details, customer details, and order date.
4. **User-Friendly Interface:** A single dashboard to access all features of the project, providing an easy-to-use interface.
5. **Efficient Record Keeping:** Helps keep track of stock, expiry date, and customer information efficiently.
6. **Time and Effort Saving:** Reduces time consumption and human effort involved in manual record keeping and inventory management.

CLASS DIAGRAM

Main window
+master: tkinter object +toplevel: tkinter object +image: tkinter PhotoImage object +label1: tkinter Entry object +label2: tkinter Entry object +label3: tkinter Entry object +label4: tkinter Entry object +label5: tkinter Entry object +label6: tkinter Entry object +label7: tkinter Entry object +butt1: tkinter Button object +butt2: tkinter Button object +butt3: tkinter Button object +butt4: tkinter Button object +butt5: tkinter Button object +butt6: tkinter Button object +butt7: tkinter Button object
+on_view_order: None +on_generate_invoice: None +add_product: None +up_product: None +order_product: None +ran_num: None +view_product: None +del_product: None +on_toplevel_close: None

Customer
+name:str +num:int +mail:str +addr:str
+cust:None +get_cust:tuple

Order
+prod_det: list +amt: float +idno: integer
+ordering_pro_win: None +view_order: None +on_click_generate_invoice: None +generate_invoice: None +add_to_tree: None +on_odress: None +on_top_close: None +restoring_orders: None +on_view_close: None +on_view_close_generate: None

Product_Window
+name: str +id: int +address: str +brand: str +type: str +dose: str +weight: str +quan: int +mrp: int +pack: str +exp: str
+adding_product_win: None +update_window: None +view_product: None +on_toplevel_up_close: None +on_toplevel_clos: None +on_toplevel_clo: None +on_added: None +on_update: None +restoring_pro: None

CLASSES

The PHARMA STORE MANAGEMENT SYSTEM involves four distinct classes: Main_window, Product, Order, and Customer.

Main_window :

Responsible for creating and managing the graphical user interface of the main window for ordering products. Implements various methods for adding, updating, and viewing products, as well as generating invoices.

This class has several attributes, including master, toplevel, image, label1-7, and butt1-7, which are all tkinter objects.

The class has several methods:

1. **on_view_order:** This method is called when the user clicks the "View Order" button (butt1). It doesn't take any arguments and its purpose is to display a list of the products
2. **on_generate_invoice:** This method is called when the user clicks the "Generate Invoice" button (butt2). It doesn't take any arguments and its purpose is to create an invoice for the user based on the products they have ordered.
3. **add_product:** This method is called when the user clicks the "Add Product" button (butt3). It doesn't take any arguments and its purpose is to allow the user to add a new product .
4. **up_product:** This method is called when the user clicks the "Update Product" button (butt4). It doesn't take any arguments and its purpose is to allow the user to update the details of a product.
5. **order_product:** This method is called when the user clicks the "Order Product" button (butt5). It doesn't take any arguments and its purpose is to add a product to the user's order.
6. **ran_num:** This method is called when the user clicks the "Generate Random Number" button (butt6). It doesn't take any arguments and its purpose is to generate a random number.
7. **view_product:** This method is called when the user clicks the "View Product" button (butt7). It doesn't take any arguments and its purpose is to display a list of all the available products.
8. **del_product:** This method is called when the user wants to delete a product from their order. It doesn't take any arguments and its purpose is to remove the selected product from the user's order.
9. **on_toplevel_close:** This method is called when the user closes the toplevel window. Its purpose is to destroy the window and free up any resources associated with it.

Order:

Responsible for managing the ordering process, including creating and saving new orders, displaying order details, and generating invoices. Has methods for adding products to orders and restoring previously saved orders.

Order class has the following attributes:

1. **prod_det:** a list of product details (presumably items being ordered)
2. **amt:** a float representing the total amount of the order
3. **idno:** an integer representing the unique identifier for the order

The Order class also has several methods:

1. **on_view_close_generate:** this method closes the window displaying the order details and generates an invoice for the order
2. **ordering_pro_win:** this method opens a window to allow the user to input the details of the products they wish to order
3. **view_order:** this method displays the details of the order in a tree format
4. **on_click_generate_invoice:** this method is triggered when the user clicks on the "generate invoice" button, and it calls the "generate_invoice" method to create an invoice for the order
5. **generate_invoice:** this method creates an invoice for the order and saves it as a word file
6. **add_to_tree:** this method adds the details of the order to a tree view
7. **on_odress:** This is a method that, when a button is pressed in the order window, orders a product and saves its details in the system, and then opens the main window.
8. **on_top_close:** this method closes the window for entering product details
9. **restoring_orders:** this method restores previously saved orders
10. **on_view_close:** this method closes the window displaying the order details

Customer:

Responsible for storing and retrieving customer information, including name, phone number, email, and address. Implements methods for saving customer details and retrieving them as a tuple.

Customer class has the following attributes:

1. **name:** a string representing the customer's name
2. **num:** an integer representing the customer's phone number
3. **mail:** a string representing the customer's email address
4. **addr:** a string representing the customer's address

The Customer class has 2 method:

1. **cust:** This method takes customer details and saves them as attributes.
2. **get_cust:** this method returns a tuple containing the customer's name, phone number, email address, and address.

Product_window:

Responsible for managing the product database, including adding, updating, and viewing product details. Implements methods for restoring previously saved product data from a file.

The class has the following attributes:

1. **name:** a string representing the name of the product
2. **id:** an integer representing the unique identifier for the product
3. **address:** a string representing the address of the product
4. **brand:** a string representing the brand of the product
5. **type:** a string representing the type of the product
6. **dose:** a string representing the dose of the product
7. **weight:** a string representing the weight of the product
8. **quan:** an integer representing the quantity of the product
9. **mrp:** an integer representing the maximum retail price of the product
10. **pack:** a string representing the packaging of the product
11. **exp:** a string representing the expiration date of the product

The Product Window class has several methods:

1. **adding_product_win:** this method opens a window for adding a new product to the system
2. **update_window:** this method opens a window for updating an existing product in the system
3. **view_product:** this method displays the details of a product in a window
4. **on_toplevel_up_close:** this method is triggered when the user closes the window for updating a product
5. **on_toplevel_clos:** this method is triggered when the user closes the window for adding a new product
6. **on_toplevel_clo:** this method is triggered when the user closes the window for viewing a product
7. **on_added:** this method adds a new product to the system and saves the updated product data to a pickle file
8. **on_update:** this method updates an existing product in the system and saves the updated product data to a pickle file
9. **restoring_pro:** this method restores previously saved product data from a pickle file

IMPLEMENTATION

Project folder: [https://drive.google.com/drive/pharma store](https://drive.google.com/drive/pharma%20store)

Main.py code:

```
from tkinter import *
from tkinter.ttk import Progressbar

import pharama

root = Tk()

# Set the size and position of the window in the center of the screen
height = 430
width = 530
x = (root.winfo_screenwidth() // 2) - (width // 2)
y = (root.winfo_screenheight() // 2) - (height // 2)
root.geometry('{}x{}'.format(width, height, x, y))

# Remove the window border and set the background color
root.overrideredirect(True)
root.config(background='#1B3D81')

# Create a heading label and add an image
heading = Label(root, text='Pharma store', bg='#1B3D81', font=('Comic Sans MS', 25, 'bold'))
heading.place(x=160, y=3)

bg = PhotoImage(file='images\\pharma.png')
bg_image = Label(root, image=bg, bg='#1B3D81', activebackground='#1B3D81')
bg_image.place(x=100, y=70)

# Add a progress bar and a label
progress_label = Label(root, text='Please wait...', bg='#1B3D81', font=('Comic Sans MS', 15, 'bold'))
progress_label.place(x=15, y=350)

progress = Progressbar(root, orient=HORIZONTAL, length=500, mode='determinate')
progress.place(x=15, y=390)

# Define the load function to update the progress bar

i = 0
```

```

def load():
    global i

    if i <= 10:
        txt = 'Please wait... ' + (str(10 * i) + '%')
        progress_label.config(text=txt)
        progress_label.after(100, load)
        progress['value'] = 10 * i
        i += 1

# Call the load function to start the progress bar
load()

def new_window():
    pharama.Restore(root)
    root.withdraw()

root.after(2500, new_window)

# Start the GUI event loop
root.mainloop()

```

Pharma.py code:

```

from tkinter import *
from tkinter import ttk
from docxtpl import DocxTemplate
import pickle
import os
import random
import datetime

master1 = None

product = {}
order = {}

def restoring():
    global product, order

```

```

with open('data/product.pkl', 'rb') as handle:
    l = pickle.load(handle)

    for i in l:
        product[i[0]] = Product_Window()
        product[i[0]].restoring_pro( i[1], i[2], i[3], i[4], i[5], i[6], i[7], i[8], i[9], i[10], i[11] )


with open('data/order.pkl', 'rb') as hand:
    l1 = pickle.load(hand)

    for j in l1:
        order[j[0]] = Order()
        order[j[0]].restoring_orders( j[1], j[2], j[3], j[4], j[5], j[6], j[7])

hand.close()

print(order)

def saving():
    global product, order

    l = []
    l1 = []
    for i in product:
        l.append([i , product[i].name , product[i].id ,product[i].address, product[i].brand ,
        product[i].type , product[i].dose , product[i].weight, product[i].quan , product[i].mrp,
        product[i].pack ,product[i].exp ])

    for j in order:
        l1.append([j,order[j].idno, order[j].name, order[j].num , order[j].mail, order[j].addr,
        order[j].prod_det, order[j].amt])

    with open('data/product.pkl', 'wb') as handle:
        pickle.dump(l, handle)
        handle.close()

    with open('data/order.pkl', 'wb') as hand:
        pickle.dump(l1, hand)
        hand.close()

class Restore:
    def __init__(self, master):
        restoring()
        Main_Window(master)

```

```

class Product_Window:
    def adding_product_win(self, name ):
        self.win = Toplevel()
        self.win.protocol("WM_DELETE_WINDOW", self.on_toplevel_clos)
        self.win.geometry('{}x{}'.format(900, self.win.wininfo_screenheight()-100))
        self.win.state('zoomed')

        image = PhotoImage(file='images\\images.png')
        self.win.iconphoto(False, image)

        Label(self.win, text='Add product', font=('Comic Sans MS', 25)).place(x=400, y=10)
        self.win.title('Add Product ')

        Label(self.win, text=' ', font=('Comic Sans MS', 15)).grid(row=0, pady=20)

        for i in range(11, 16, 2):
            Label(self.win, text=':', font=('Comic Sans MS', 15)).grid(row=i, column=5)

        for i in range(1,16,2 ):
            Label(self.win,text=':',font=('Comic Sans MS', 15)).grid(row=i, column=1)

        for i in range(1,16,2 ):
            Label(self.win,text=' ',font=('Comic Sans MS', 15)).grid(row=i, column=3,
padx=20)

        for i in range(2,20,2 ):
            Label(self.win,text=' ').grid(row=i, pady=4)

        self.name = name
        Label(self.win, text= 'product name', font=('Comic Sans MS', 15)).grid(row=1)
        Label(self.win, text= self.name, font=('Comic Sans MS', 15, 'bold') ).grid(row=1,
column=2, sticky='nsew')

        Label(self.win, text= 'Product id ', font=('Comic Sans MS', 15,)).grid(row=3)
        id = Entry(self.win, font=('Comic Sans MS', 15,))
        id.grid(row=3, column=2)

        Label(self.win, text='manufacturer \n address', font=('Comic Sans MS',
15,)).grid(row=5)
        address = Entry(self.win, font=('Comic Sans MS', 15,))
        address.grid(row=5, column=2)

        Label(self.win, text= 'Brand name ', font=('Comic Sans MS', 15,)).grid(row=7)

```



```
brand = Entry(self.win, font=('Comic Sans MS', 15,))
brand.grid(row=7,column=2)
```

```
Label(self.win, text= 'Product type ', font=('Comic Sans MS', 15, )).grid(row=9)
type = Entry(self.win , font=('Comic Sans MS', 15,))
type.grid(row=9,column=2)
```

```
Label(self.win, text= 'Product dose', font=('Comic Sans MS', 15, )).grid(row=11)
dose = Entry(self.win, font=('Comic Sans MS', 15,))
dose.grid(row=11, column=2)
```

```
Label(self.win, text= 'Net quantity', font=('Comic Sans MS', 15, )).grid(row=13)
weight = Entry(self.win, font=('Comic Sans MS', 15,))
weight.grid(row=13, column=2)
```

```
Label(self.win, text='quantity', font=('Comic Sans MS', 15,)).grid(row=15)
quan = Entry(self.win, font=('Comic Sans MS', 15,))
quan.grid(row=15, column=2)
```

```
Label(self.win, text='product mrp', font=('Comic Sans MS', 15,)).grid(row=11,
column=4)
mrp = Entry(self.win, font=('Comic Sans MS', 15,))
mrp.grid(row=11, column=6)
```

```
Label(self.win, text='packing date', font=('Comic Sans MS', 15,)).grid(row=13,
column=4)
pack = Entry(self.win, font=('Comic Sans MS', 15,))
pack.grid(row=13, column=6)
```

```
Label(self.win, text='expiary date ', font=('Comic Sans MS', 15,)).grid(row=15,
column=4)
exp = Entry(self.win, font=('Comic Sans MS', 15,))
exp.grid(row=15, column=6)
```

```
add = Button(self.win, text='ADD', font=('Comic Sans MS', 15,), command= lambda :
self.on_added(id, address, brand, type, dose, weight , quan, mrp, pack, exp))
add.grid(row=17, column=6, sticky='nesw')
dele = Button(self.win, text='EXIT', font=('Comic Sans MS', 15,), command= lambda
:self.on_toplevel_clos())
dele.grid(row=19, column=6, sticky='nesw')
```

```
def update_window(self):
```

```

self.up_window = Toplevel()
self.up_window.protocol("WM_DELETE_WINDOW", self.on_toplevel_up_close)
self.up_window.geometry('{}x{}'.format(900, self.up_window.winfo_screenheight() -
100))
self.up_window.state('zoomed')

image = PhotoImage(file='images\\images.png')
self.up_window.iconphoto(False, image)

Label(self.up_window, text='Update product', font=('Comic Sans MS',
25)).place(x=400, y=10)
self.up_window.title('Update Product ')

Label(self.up_window, text=' ', font=('Comic Sans MS', 15)).grid(row=0, pady=20)

for i in range(11, 16, 2):
    Label(self.up_window, text=':', font=('Comic Sans MS', 15)).grid(row=i, column=5)

for i in range(1, 16, 2):
    Label(self.up_window, text=':', font=('Comic Sans MS', 15)).grid(row=i, column=1)

for i in range(1, 16, 2):
    Label(self.up_window, text=' ', font=('Comic Sans MS', 15)).grid(row=i, column=3,
padx=20)

for i in range(2, 20, 2):
    Label(self.up_window, text=' ').grid(row=i, pady=4)

Label(self.up_window, text='product name', font=('Comic Sans MS', 15)).grid(row=1)
Label(self.up_window, text=self.name, font=('Comic Sans MS', 15, 'bold')).grid(row=1,
column=2, sticky='nsew')

Label(self.up_window, text='Product id ', font=('Comic Sans MS', 15,)).grid(row=3)
id = Entry(self.up_window, font=('Comic Sans MS', 15,))
id.insert(1, self.id)
id.grid(row=3, column=2)

Label(self.up_window, text='manufacturer \n address', font=('Comic Sans MS',
15,)).grid(row=5)
address = Entry(self.up_window, font=('Comic Sans MS', 15,))
address.insert(1, self.address)
address.grid(row=5, column=2)

Label(self.up_window, text='Brand name ', font=('Comic Sans MS', 15,)).grid(row=7)
brand = Entry(self.up_window, font=('Comic Sans MS', 15,))
brand.insert(1, self.brand)
brand.grid(row=7, column=2)

```

```

Label(self.up_window, text='Product type ', font=('Comic Sans MS', 15,)).grid(row=9)
type = Entry(self.up_window, font=('Comic Sans MS', 15,))
type.insert(1, self.type)
type.grid(row=9, column=2)

Label(self.up_window, text='Product dose', font=('Comic Sans MS', 15,)).grid(row=11)
dose = Entry(self.up_window, font=('Comic Sans MS', 15,))
dose.insert(1, self.dose)
dose.grid(row=11, column=2)

Label(self.up_window, text='Net quantity', font=('Comic Sans MS', 15,)).grid(row=13)
weight = Entry(self.up_window, font=('Comic Sans MS', 15,))
weight.insert(1, self.weight)
weight.grid(row=13, column=2)

Label(self.up_window, text='quantity', font=('Comic Sans MS', 15,)).grid(row=15)
quan = Entry(self.up_window, font=('Comic Sans MS', 15,))
quan.insert(1, self.quan)
quan.grid(row=15, column=2)

Label(self.up_window, text='product mrp', font=('Comic Sans MS', 15,)).grid(row=11,
column=4)
mrp = Entry(self.up_window, font=('Comic Sans MS', 15,))
mrp.insert(1, self.mrp)
mrp.grid(row=11, column=6)

Label(self.up_window, text='packing date', font=('Comic Sans MS', 15,)).grid(row=13,
column=4)
pack = Entry(self.up_window, font=('Comic Sans MS', 15,))
pack.insert(1, self.pack)
pack.grid(row=13, column=6)

Label(self.up_window, text='expiary date ', font=('Comic Sans MS', 15,)).grid(row=15,
column=4)
exp = Entry(self.up_window, font=('Comic Sans MS', 15,))
exp.insert(1, self.exp)
exp.grid(row=15, column=6)

update = Button(self.up_window, text='UPDATE', font=('Comic Sans MS',
15,), command=lambda: self.on_update(id, address, brand, type, dose, weight, quan, mrp,
pack, exp))
update.grid(row=17, column=6, sticky='nesw')

def view_product(self):
    self.view = Toplevel()
    self.view.protocol("WM_DELETE_WINDOW", self.on_toplevel_clo)
    self.view.geometry('{}x{}'.format(900, self.view.winfo_screenheight() - 100))

```

```

self.view.state('zoomed')

image = PhotoImage(file='images\\images.png')
self.view.iconphoto(False, image)

Label(self.view, text='product Details', font=('Comic Sans MS', 25)).place(x=400, y=10)
self.view.title('Product Details')

Label(self.view, text=' ', font=('Comic Sans MS', 15)).grid(row=0, pady=20)

for i in range(11, 16, 2):
    Label(self.view, text=':', font=('Comic Sans MS', 15)).grid(row=i, column=5)

for i in range(1, 16, 2):
    Label(self.view, text=':', font=('Comic Sans MS', 15)).grid(row=i, column=1)

for i in range(1, 16, 2):
    Label(self.view, text=' ', font=('Comic Sans MS', 15)).grid(row=i, column=3,
padx=20)

for i in range(2, 20, 2):
    Label(self.view, text=' ').grid(row=i, pady=4)

Label(self.view, text='product name', font=('Comic Sans MS', 15)).grid(row=1)
Label(self.view, text=self.name, font=('Comic Sans MS', 15, 'bold')).grid(row=1,
column=2, sticky='nsew')

Label(self.view, text='Product id ', font=('Comic Sans MS', 15,)).grid(row=3)
Label(self.view, text=self.id, font=('Comic Sans MS', 15,)).grid(row=3, column=2)

Label(self.view, text='manufacturer \n address', font=('Comic Sans MS',
15,)).grid(row=5)
Label(self.view, text=self.address, font=('Comic Sans MS', 15,)).grid(row=5, column=2)

Label(self.view, text='Brand name ', font=('Comic Sans MS', 15,)).grid(row=7)
Label(self.view, text=self.brand, font=('Comic Sans MS', 15,)).grid(row=7, column=2)

Label(self.view, text='Product type ', font=('Comic Sans MS', 15,)).grid(row=9)
Label(self.view, text=self.type, font=('Comic Sans MS', 15,)).grid(row=9, column=2)

Label(self.view, text='Product dose', font=('Comic Sans MS', 15,)).grid(row=11)
Label(self.view, text=self.dose, font=('Comic Sans MS', 15,)).grid(row=11, column=2)

Label(self.view, text='Net quantity', font=('Comic Sans MS', 15,)).grid(row=13)
Label(self.view, text=self.weight, font=('Comic Sans MS', 15,)).grid(row=13,
column=2)

Label(self.view, text='quantity', font=('Comic Sans MS', 15,)).grid(row=15)

```

```

Label(self.view,text= self.quan, font=('Comic Sans MS', 15,)).grid(row=15, column=2)

Label(self.view, text='product mrp', font=('Comic Sans MS', 15,)).grid(row=11,
column=4)
Label(self.view,text=self.mrp, font=('Comic Sans MS', 15,)).grid(row=11, column=6)

Label(self.view, text='packing date', font=('Comic Sans MS', 15,)).grid(row=13,
column=4)
Label(self.view, text=self.pack, font=('Comic Sans MS', 15,)).grid(row=13, column=6)

Label(self.view, text='expiary date ', font=('Comic Sans MS', 15,)).grid(row=15,
column=4)
Label(self.view, text=self.exp, font=('Comic Sans MS', 15,)).grid(row=15, column=6)

dele = Button(self.view, text='EXIT', font=('Comic Sans MS', 15,), command=lambda:
self.on_toplevel_clo())
dele.grid(row=19, column=6, sticky='nesw')

def on_toplevel_up_close(self):
    Main_Window(master1)
    self.up_window.withdraw()

def on_toplevel_clos(self):
    global product
    del product[self.name]
    Main_Window(master1)
    self.win.withdraw()

def on_toplevel_clo(self):
    Main_Window(master1)
    self.view.withdraw()

def on_added(self, id, address, brand, type, dose, weight , quan, mrp, pack, exp):
    self.id = id.get()
    self.address = address.get()
    self.brand = brand.get()
    self.type = type.get()
    self.dose = dose.get()
    self.weight = weight.get()
    self.quan = quan.get()
    self.mrp = mrp.get()
    self.pack = pack.get()
    self.exp = exp.get()
    Main_Window(master1)
    self.win.withdraw()

```

```

saving()

def on_update(self, id, address, brand, type, dose, weight , quan, mrp, pack, exp):
    self.id = id.get()
    self.address = address.get()
    self.brand = brand.get()
    self.type = type.get()
    self.dose = dose.get()
    self.weight = weight.get()
    self.quan = quan.get()
    self.mrp = mrp.get()
    self.pack = pack.get()
    self.exp = exp.get()
    Main_Window(master1)
    self.up_window.withdraw()

def restoring_pro(self, name , id, address, brand, type, dose, weight , quan, mrp, pack,
exp):
    self.name = name
    self.id = id
    self.address = address
    self.brand = brand
    self.type = type
    self.dose = dose
    self.weight = weight
    self.quan = quan
    self.mrp = mrp
    self.pack = pack
    self.exp = exp

class Customer:
    def cust(self, name , num, mail, addr):
        self.name = name
        self.num = num
        self.mail = mail
        self.addr = addr

    def get_cust(self):
        return self.name, self.num, self.mail, self.addr

class Order(Customer):
    def ordering_pro_win(self, num , name ):
        global product
        self.prod_det = []
        self.amt = float(0)

        self.or_win = Toplevel()
        self.or_win.protocol("WM_DELETE_WINDOW", self.on_top_close)

```

```

image = PhotoImage(file='images\\images.png')
self.or_win.iconphoto(False, image)

self.or_win.title('Place Order')

self.idno = num
self.cu_name = name.get()
frame = Frame(self.or_win)
frame.pack(padx=30, pady=5)

self.or_win.state('zoom')

Label(frame, text=' ', font=('Comic Sans MS', 25)).grid(row=0, column=0,
columnspan=6, pady=20)
Label(frame, text=f" Order id : {self.idno}", font=('Comic Sans MS', 25)).grid(row=0,
column=0, columnspan=6)

for i in range(1, 5):
    Label(frame, text=' ', font=('Comic Sans MS', 12)).grid(row=i, column=4, padx=7)

for i in range(1,5):
    Label(frame, text=':', font=('Comic Sans MS', 12)).grid(row=i, column=1)

Label(frame, text='customer name ', font=('Comic Sans MS', 12)).grid(row=1,
column=0, pady=2)
en1 = Entry(frame, font=('Comic Sans MS', 12))
en1.grid(row=1, column=2)

en1.insert(1, self.cu_name)

Label(frame, text='mobile number', font=('Comic Sans MS', 12)).grid(row=2,
column=0, pady=2)
en2 = Entry(frame, font=('Comic Sans MS', 12))
en2.grid(row=2, column=2)

Label(frame, text='Email id ', font=('Comic Sans MS', 12)).grid(row=3, column=0,
pady=2)
en3 = Entry(frame, font=('Comic Sans MS', 12))
en3.grid(row=3, column=2)

Label(frame, text='customer adress', font=('Comic Sans MS', 12)).grid(row=4,
column=0, pady=2)
en4 = Entry(frame, font=('Comic Sans MS', 12))
en4.grid(row=4, column=2)

Label(frame, text='product name ', font=('Comic Sans MS', 12)).grid(row=1, column= 5,

```

```

pady=2)
    en5 = Entry(frame, font=('Comic Sans MS', 12))
    en5.grid(row=2, column=5, pady=2)
    Label(frame, text='Quantity ', font=('Comic Sans MS', 12)).grid(row=3, column=5,
pady=2)
    en6 = Spinbox(frame, from_=0, to=1000, font=('Comic Sans MS', 12))
    en6.grid(row=4, column=5, pady=2)

    tree = ttk.Treeview(frame, columns=('qty', 'prod', 'mrp', 'total'), show='headings',
height=16)
    tree.heading('qty', text='Qty')
    tree.heading('prod', text='Product ')
    tree.heading('mrp', text='mrp')
    tree.heading('total', text='Total')

    tree.grid(row=6, column=0, columnspan=6, padx=10, pady=10)

    but = Button(frame, text='Add Product ', font=('Comic Sans MS', 12),
command=lambda: self.add_to_tree(en5, en6, tree))
    but.grid(row=5, column=5)

    Label(frame, text=f'Total amount : {self.amt} /-', font=('Comic Sans MS',
15)).grid(row=7, column=5, sticky='e')

    but1 = Button(frame, text='Place Order ', font=('Comic Sans MS', 12), command=
lambda :self.on_odress( en1, en2, en3, en4))
    but1.grid(row=8, column=0, columnspan=6, sticky='nesw')

    self.frame = frame

def view_order(self):
    self.v_or_win = Toplevel()

    self.v_or_win.protocol("WM_DELETE_WINDOW", self.on_view_close)

    self.v_or_win.state('zoom')
    frame = Frame(self.v_or_win)
    frame.pack(pady=10, padx=30)

    Label(frame, text=' ', font=('Comic Sans MS', 15)).grid(row=0, column=0,
columnspan=7, padx=20, pady=20)
    Label(frame, text= f'ORDER ID : {self.idno}', font=('Comic Sans MS',
17)).grid(row=0, column=0, columnspan=7)
    Label(frame, text='customer Details ', font=('Comic Sans MS', 15)).grid(row=1,
column=0, columnspan=2, sticky='e')

```



```

name, num, mail, addr = super().get_cust()

Label(frame, text='Name' , font=('Comic Sans MS', 12)).grid(row=2, sticky='e')
Label(frame, text=name , font=('Comic Sans MS', 12)).grid(row=2, column=1)

Label(frame, text='mobile', font=('Comic Sans MS', 12)).grid(row=3, sticky='e')
Label(frame, text=num, font=('Comic Sans MS', 12)).grid(row=3, column=1)

Label(frame, text='Email' , font=('Comic Sans MS', 12)).grid(row=4, sticky='e')
Label(frame, text=mail, font=('Comic Sans MS', 12)).grid(row=4, column=1)

Label(frame, text='adress', font=('Comic Sans MS', 12)).grid(row=5, sticky='e')
Label(frame, text=addr, font=('Comic Sans MS', 12)).grid(row=5, column=1)

Label(frame, text='Order Details ', font=('Comic Sans MS', 15)).grid(row=6, column=0,
columnspan=2, sticky='e')

k=0

Label(frame, text='Quantity', font=('Comic Sans MS', 15, 'bold')).grid(row=7,
column=1, padx=5)
Label(frame, text='product name ', font=('Comic Sans MS', 15, 'bold')).grid(row=7,
column=2, padx=7)
Label(frame, text='product mrp', font=('Comic Sans MS', 15, 'bold')).grid(row=7,
column=3, padx=5)
Label(frame, text='amount', font=('Comic Sans MS', 15, 'bold')).grid(row=7, column=4,
padx=5)

for i in range(8, 8+len(self.prod_det)):
    Label(frame, text=self.prod_det[i-8][0], font=('Comic Sans MS', 15)).grid(row=i,
column=1, padx=5)
    Label(frame, text=self.prod_det[i-8][1], font=('Comic Sans MS', 15)).grid(row=i,
column=2, padx=7)
    Label(frame, text=self.prod_det[i-8][2], font=('Comic Sans MS', 15)).grid(row=i,
column=3, padx=5)
    Label(frame, text=self.prod_det[i-8][3], font=('Comic Sans MS', 15)).grid(row=i,
column=4, padx=5)
    k=i
    k=k+1
    Label(frame, text="", font=('Comic Sans MS', 15)).grid(row=k, column=1,
columnspan=4)

    k = k + 1
    Label(frame, text=f"Total amt : {self.amt}/-", font=('Comic Sans MS', 15)).grid(row=k,
column=4, columnspan=2)

    k = k + 1
    but = Button(frame, text='Generate invoice', font=('Comic Sans MS', 15),
command=lambda : self.generate_invoice())

```

```

        but.grid(row=k, column=4, sticky='nesw')

        k = k + 1
        but2 = Button(frame, text='EXIT', font=('Comic Sans MS', 15), command= lambda
:self.on_view_close())
        but2.grid(row=k, column=4, sticky='nesw')

    def on_click_generate_invoice(self):
        doc = DocxTemplate('invoice.docx')
        doc.render({"name":self.name, "numb":self.num, "mail":self.mail, "ad": self.addr,
"invoice":self.prod_det, "amount":self.amt, "idn":self.idno} )
        downloads_dir = os.path.join(os.path.expanduser("~"), "Downloads")
        downloads_dir = downloads_dir + '/' + f'{self.name}-{self.idno}-invoice.docx'
        doc.save(downloads_dir)

    def generate_invoice(self):
        doc = DocxTemplate('invoice.docx')
        doc.render({"name":self.name, "numb":self.num, "mail":self.mail, "ad": self.addr,
"invoice":self.prod_det, "amount":self.amt, "idn":self.idno} )
        downloads_dir = os.path.join(os.path.expanduser("~"), "Downloads")
        downloads_dir = downloads_dir + '/' + f'{self.name}-{self.idno}-invoice.docx'
        doc.save(downloads_dir)

    Main_Window(master1)
    self.v_or_win.withdraw()

    def add_to_tree(self, pro, quan, tree):
        global product
        if pro.get() in product:
            product[pro.get()].quan = str(int(product[pro.get()].quan) - int(quan.get()))
            tree.insert("", 0, values=[quan.get(), pro.get(), product[pro.get()].mrp, int(quan.get()) *
float(product[pro.get()].mrp)])
            self.prod_det.append([quan.get(), pro.get(), product[pro.get()].mrp, int(quan.get()) *
float(product[pro.get()].mrp)])
            self.amt = self.amt + int(quan.get()) * float(product[pro.get()].mrp)
            Label(self.frame, text=f'Total amount :{self.amt} /-', font=('Comic Sans MS',
15)).grid(row=7, column=5,sticky='e')
            pro.delete(0, END)
            quan.delete(0, END)
            quan.insert(0, '0')
        else:
            pro.delete(0, END)

```

```

        quan.delete(0, END)
        quan.insert(0, '0')
        return

def on_odress(self, name , num, mail, addr):
    super().cust(name.get() , num.get(), mail.get(), addr.get())

    Main_Window(master1)
    self.or_win.withdraw()
    saving()

def on_top_close(self):
    global order
    for i in self.prod_det:
        product[i[1]].quan = str(int(product[i[1]].quan) + int(i[0]))
    del order[self.idno]
    Main_Window(master1)
    self.or_win.withdraw()

def restoring_orders(self, idno, name, num, mail, addr, pro_det, amt):
    self.idno = idno
    super().cust(name, num, mail, addr)
    self.prod_det = pro_det
    self.amt = amt

def on_view_close(self):

    Main_Window(master1)
    self.v_or_win.withdraw()

def on_view_close_generate(self):
    self.generate_invoice()

    Main_Window(master1)
    self.v_or_win.withdraw()

class Main_Window:
    def __init__(self, master):
        saving()
        global master1
        master1 = self.master = master

        self.toplevel = Toplevel(self.master)
        self.toplevel.protocol("WM_DELETE_WINDOW", self.on_toplevel_close)
        self.toplevel.geometry('{}x{}'.format(self.toplevel.winfo_screenwidth() - 50,
        self.toplevel.winfo_screenheight() - 50))

```

```

# icon
image1 = PhotoImage(file='images\\images.png')
self.toplevel.iconphoto(False, image1)

# title
self.toplevel.title('main menu')

# window zoom
self.toplevel.state('zoomed')

x = self.toplevel.winfo_screenwidth()
y = self.toplevel.winfo_screenheight() // 2

# image
self.image = PhotoImage(file='images\\pharmacy.png')
Label(self.toplevel, image=self.image, width=x, height=y).place(x=1, y=10)

# spaces
Label(self.toplevel, text=' ').grid(row=0, pady=self.toplevel.winfo_screenwidth() / 7)
Label(self.toplevel, text=' ').grid(row=1, padx=30)

# add product
Label(self.toplevel, text='Add product').grid(row=1, column=1)
self.label1 = Entry(self.toplevel)
self.label1.grid(row=2, column=1)
self.butt1 = Button(self.toplevel, text='add', width=10, height=1, command=lambda:
self.add_product(self.label1))
self.butt1.grid(row=2, column=2, padx=7)

Label(self.toplevel, text=' ').grid(row=3, pady=10)

# view product
Label(self.toplevel, text='view product').grid(row=4, column=1)
self.label2 = Entry(self.toplevel)
self.label2.grid(row=5, column=1)
self.butt2 = Button(self.toplevel, text='view', width=10, height=1, command= lambda :
self.view_product(self.label2))
self.butt2.grid(row=5, column=2, padx=7)

# update product
Label(self.toplevel, text=' ').grid(row=1, column=3, padx=130)
Label(self.toplevel, text='update product').grid(row=1, column=4)
self.label3 = Entry(self.toplevel)
self.label3.grid(row=2, column=4)
self.butt3 = Button(self.toplevel, text='update', width=10, height=1, command= lambda :
self.up_product(self.label3))
self.butt3.grid(row=2, column=5, padx=7)

# delete product

```

```

Label(self.toplevel, text='delete product').grid(row=4, column=4)
self.label4 = Entry(self.toplevel)
self.label4.grid(row=5, column=4)
self.butt4 = Button(self.toplevel, text='delete', width=10, height=1, command=lambda:
self.del_product(self.label4))
self.butt4.grid(row=5, column=5, padx=7)

# generate invoice
Label(self.toplevel, text='').grid(row=6, pady=10)
Label(self.toplevel, text='generate invoice').grid(row=7, column=4)
self.label5 = Entry(self.toplevel)
self.label5.grid(row=8, column=4)
self.butt5 = Button(self.toplevel, text='Generate', width=10, height=1,
command=lambda :self.on_generate_invoice())
self.butt5.grid(row=8, column=5, padx=7)

#place order
Label(self.toplevel, text='').grid(row=1,column=6, padx=130)
Label(self.toplevel, text='place order\n enter customer name ').grid(row=1, column=7)
self.label6 = Entry(self.toplevel)
self.label6.grid(row=2, column=7)
self.butt6 = Button(self.toplevel, text='order', width=10, height=1, command= lambda
:self.order_product(self.label6))
self.butt6.grid(row=2, column=8, padx=7)

#view order
Label(self.toplevel, text='').grid(row=3, pady=10)
Label(self.toplevel, text='view order').grid(row=4, column=7)
self.label7 = Entry(self.toplevel)
self.label7.grid(row=5, column=7)
self.butt7 = Button(self.toplevel, text='view', width=10, height=1, command=lambda
:self.on_view_order(self.label7))
self.butt7.grid(row=5, column=8, padx=7)

def on_view_order(self, idn):
    global order
    idno = idn.get()
    idn.delete(0, END)
    if idno == "":
        return
    elif idno in order:
        order[idno].view_order()
        self.toplevel.withdraw()
    else :
        return

def on_generate_invoice(self):
    idn = self.label5.get()
    self.label5.delete(0, END)

```

```

        if idn in order:
            order[idn].on_click_generate_invoice()
        else:
            return

def add_product(self,name):
    global product
    self.name = name
    self.name = self.name.get()
    self.label1.delete(0, END)
    if self.name == "":
        return
    else:
        product[self.name] = Product_Window()
        product[self.name].adding_product_win(self.name)
        self.toplevel.withdraw()

def up_product(self, name):
    global product
    self.name = name.get()
    self.label3.delete(0, END)
    if self.name == "":
        return
    else:
        product[self.name].update_window()
        self.toplevel.withdraw()

def order_product(self, name):
    global order
    Name = name.get()
    self.or_num = self.ran_num()
    if Name == "":
        name.delete(0, END)
        return
    else:
        order[self.or_num] = Order()
        order[self.or_num].ordering_pro_win(self.or_num, name)
        self.toplevel.withdraw()

def ran_num(self):
    now = datetime.datetime.now()
    x = random.randint(0, 10)
    return str(now.strftime("%d%m%y%H%M%S")) + str(x)

def view_product(self, name):
    global product
    self.name = name
    self.name = self.name.get()
    self.label2.delete(0, END)
    if self.name == "":

```

```

        return
    else:
        product[self.name].view_product()
        self.toplevel.withdraw()

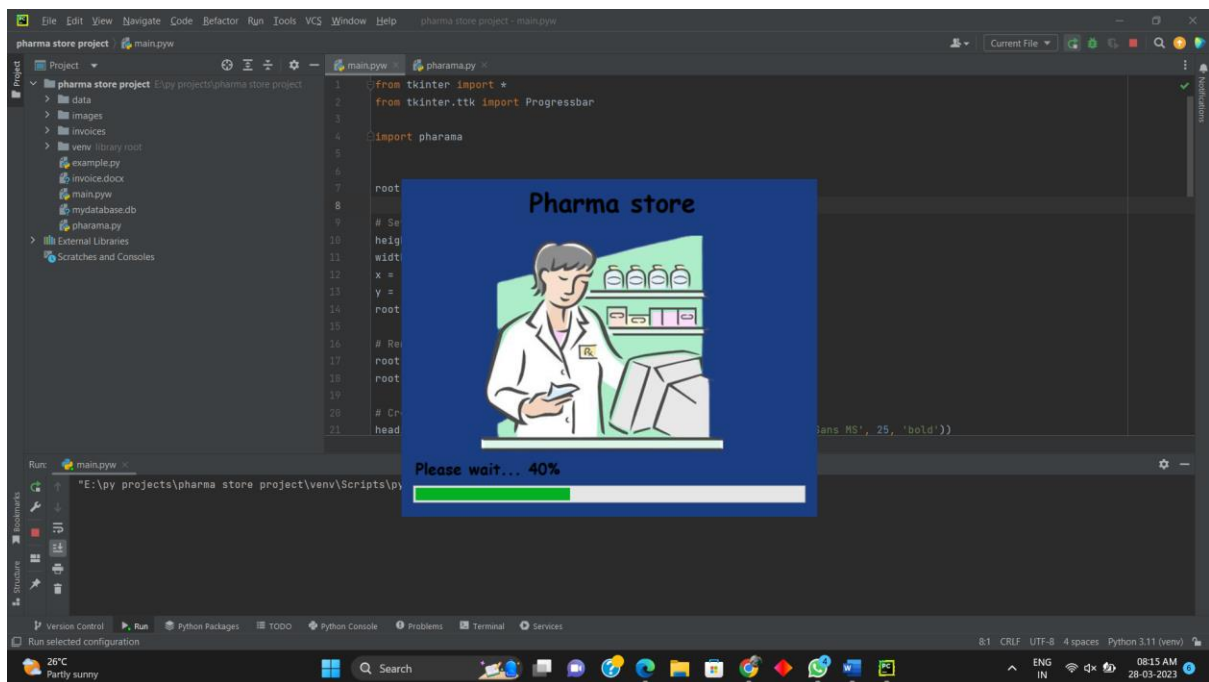
def del_product(self, name):
    global product
    self.name = name
    self.name = self.name.get()
    self.label4.delete(0, END)
    if self.name == "":
        return
    else:
        pop = Toplevel()
        pop.geometry('{}x{}'.format(400, 200, self.toplevel.winfo_screenwidth() // 2 -
200, self.toplevel.winfo_screenheight() // 2 - 100))
        pop.overridereirect(True)
        pop.config(background='#8c8c89')
        if self.name in product:
            del product[self.name]
            Label(pop, text='product successfully deleted ', background='#8c8c89',
font=('Comic Sans MS', 15, 'bold')).place(x=60, y=60)
        else:
            Label(pop, text='No such product Found \n re-enter product name ',
background='#8c8c89', font=('Comic Sans MS', 15, 'bold')).place(x=75, y=60)
            pop.after(2000, lambda :pop.withdraw())

def on_toplevel_close(self):
    pop = Toplevel()
    pop.geometry('{}x{}'.format(400, 200, self.toplevel.winfo_screenwidth() // 2 -
200, self.toplevel.winfo_screenheight() // 2 - 100))
    pop.overridereirect(True)
    pop.config(background='#8c8c89')
    Label(pop, text='Do you want exit?', background='#8c8c89', font=('Comic Sans MS', 15,
'bold')).place(x=20, y=35)
    butt1 = Button(pop, text='yes', background='#8c8c89', font=('Comic Sans MS', 10),
width=7, command=lambda: self.master.destroy())
    butt1.place(x=245, y=160)
    butt2 = Button(pop, text='no', background='#8c8c89', font=('Comic Sans MS', 10),
width=7, command=lambda: pop.withdraw())
    butt2.place(x=320, y=160)
    saving()

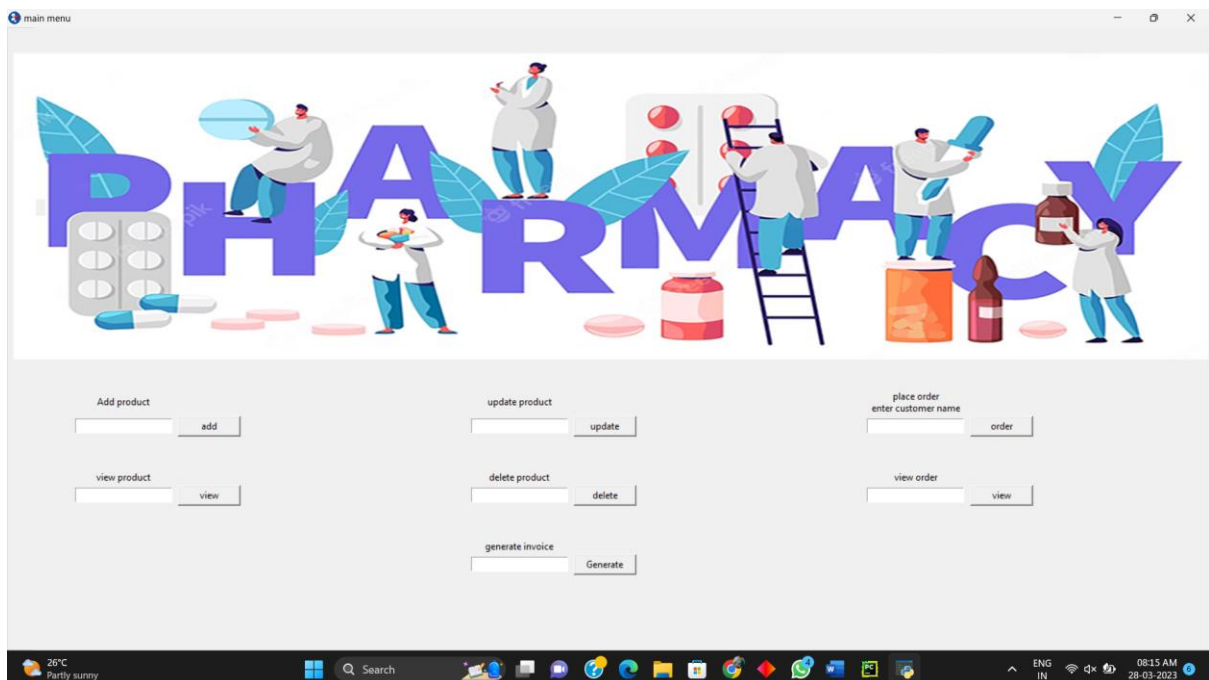
```

RESULT

Loading:



Main menu:



Add Product

Add product

product name : dolo

Product id : 2255151815

manufacturer address :

Brand name :

Product type :

Product dose : product mrp : 10

Net quantity : 5 packing date : 28/03/2023

quantity : 200 expiary date : 28/03/2040

ADD

EXIT

27°C
Light rain

Search

ENG
IN

08:47 AM
28-03-2023

Place Order

Order id : 2803230849375

customer name : jashu

mobile number : 9177227914

Email id : jashwanthai0022@gmail.cc

customer adress : vijayawada

product name : dolo650

Quantity : 250

Add Product

Qty	Product	mrp	Total
20	dolo650	20	400.0

Total amount :400.0 /-

Place Order

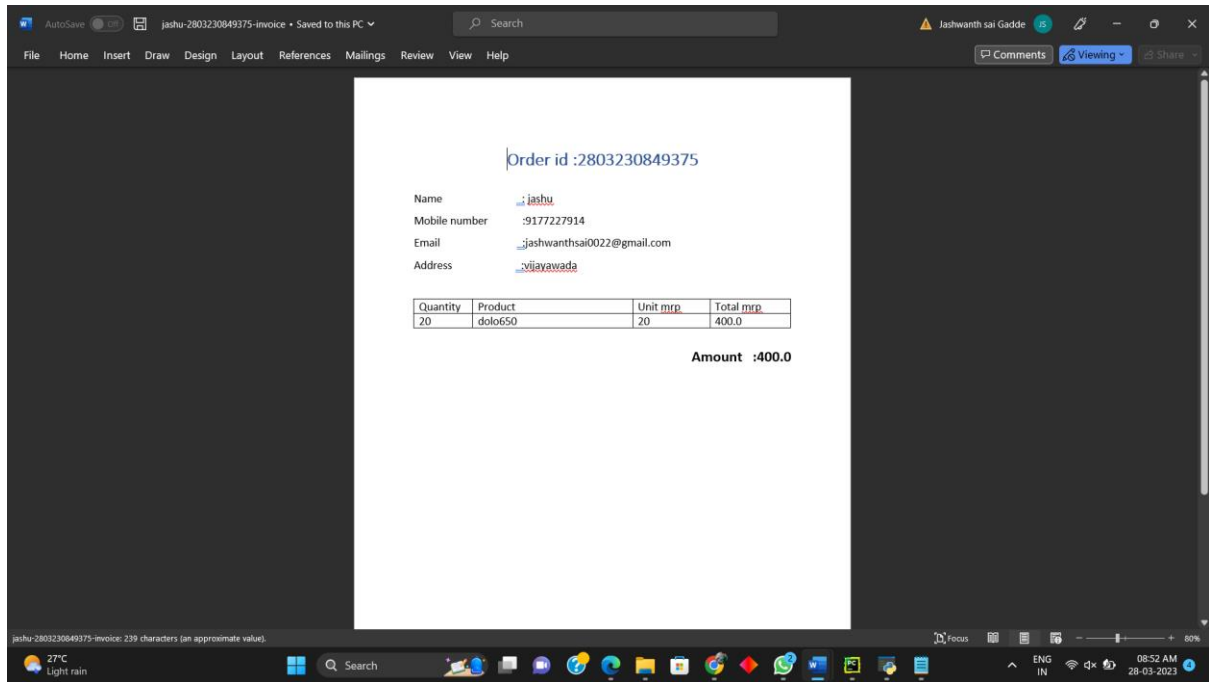
27°C
Light rain

Search

ENG
IN

08:51 AM
28-03-2023

Auto generated invoice in word document:



The screenshot shows a Microsoft Word window titled "jashu-2803230849375-invoice" with the following content:

Order id :2803230849375

Name : jashu
Mobile number :9177227914
Email :jashwanthasai0022@gmail.com
Address :vijayawada

Quantity	Product	Unit	mrp	Total mrp
20	dolo650	20		400.0

Amount :400.0

The document is displayed in a dark-themed Word interface. The status bar at the bottom indicates the document contains 239 characters (approximate value). The Windows taskbar at the bottom shows the date as 28-03-2023 and the time as 08:52 AM.

CONCLUSION

In conclusion, the Pharma Store Management System project using object-oriented programming concepts in Python and GUI with tkinter is a highly effective solution for managing the inventory and customer information in a medical store. By using OOP concepts, the project enables developers to build a modular and scalable system, making it easier to maintain and update in the future. With the added feature of a graphical user interface using tkinter, the system becomes more user-friendly and accessible, allowing users to navigate and manage the software with ease.

Through the development of this project, the team has gained practical experience in Python programming and GUI development. Additionally, the project has demonstrated the importance of using OOP and GUI in building a functional, user-friendly, and efficient management system for a pharmacy store. Overall, the Pharma Store Management System project is a valuable asset for pharmacy store managers, providing an effective solution to reduce complexities in record keeping and documentation in inventory management.