

```
===authentication.py===
```

```
from main import ReadDB
```

```
from getpass import getpass
```

```
# prompt password for login and check with decrypted password obtained from MANAGEMENT.DB
```

```
def Login():  
    passfile = ReadDB('MANAGEMENT.DB')  
    x = getpass()  
    if x.strip() == Decoder(passfile['password']):  
        return 0  
    return -1
```

```
# function to hash/encode password
```

```
def Encoder(x):  
    y = ''  
    for i in x:  
        y += str(ord(i))[::-1] + '◆'  
    return y[::-1]
```

```
# function to unhash/decode password
```

```
def Decoder(x):  
    x = x[::-1].split('◆')  
    y = ''  
    for i in x[:-1]:  
        y += chr(int(i[::-1]))  
    return y
```

```
===billingpage.py===
```

```
from management import SearchWithName, Logger
```

```
from ui import tabulate, Clear
```

```
from main import CheckLocalFiles
```

```
from stock import ReadBulkFile, BulkRemove
```

```
from users import FinalBill, AddUserLogs
```

```
from main import ReadDB, WriteDB
```

```
import datetime
```

```
# drug buy menu
```

```
def PrimaryInput(db):  
    if len(db) == 0:  
        input("Stock empty. Press any key to continue.")  
        return -1  
    drugslist = {}  
    def DrugPrompt():  
        Clear()  
        # get drug name using SearchWithName  
        def DrugNameInput():  
            # TODO  
            # print("Input prompted value or '0' to go back to main menu.")  
            med = input("Drug name: ").strip()  
            templist = []  
            for i in SearchWithName(med,db):  
                qty = 0  
                for j in db[i][1]:  
                    qty += db[i][1][j]  
                templist.append([len(templist)+1,i,db[i][0],qty])  
            if len(templist) == 0:  
                Clear()  
                print("No Results Found")  
                return DrugNameInput()  
            return templist  
    templist = DrugNameInput()
```

```

Clear()
print(f"{len(templist)} Results Found:")
tabulate(
    "SNo.,ID,Name,Available".split(","),
    templist)

# selection menu for all search results of DrugNameInput
def choicePrompt():
    choice = input(f"Enter your choice [1-{len(templist)}]:")
    if choice.isnumeric():
        choice = int(choice)
        if choice not in range(1,len(templist)+1):
            print("Invalid input, try again")
            return choicePrompt()
        else: return choice
    else:
        print("Invalid input, try again")
        return choicePrompt()

drugindex = choicePrompt()-1 # relative inside templist
print(f"You chose {templist[drugindex][2]}")

# qty prompt for entering order details of selected choice
def qtyPrompt():
    x = input("Enter quantity: ")
    if x.isnumeric():
        x = int(x)
        if x ≤ 0:
            print("Invalid input, try again")
            return qtyPrompt()
        if x>templist[drugindex][-1]:
            print(f"Not enough available, taking maximum available: {templist[drugindex]
[-1]}")
            return templist[drugindex][-1]
        else:
            return x
    else:
        print("Invalid input, try again")
        return qtyPrompt()
inputqty = qtyPrompt()
drugslist[templist[drugindex][1]] = [templist[drugindex][2],inputqty]

# actual running here
while True:
    customer_name = input("Customer name: ")
    if customer_name.isspace() or customer_name == "":
        print("Invalid input, try again")
    else:
        customer_name = customer_name.title()
        break
# to check if bulk order or not
isbulk = input("Bulk order (y/n): ").lower()
if isbulk == 'y':
    input("Place bulk order list file inside the FILES directory ")
    files = CheckLocalFiles()
    print(f"{len(files)} Results found:")
    tabulate(
        ["SNo.", "File", ],
        [[files.index(i)+1,i] for i in files],
        linesbetweenrows = True
    )

# chose files for bulk add reading
def BulkPrompt():
    y = input(f"Enter your choice [1-{len(files)}]: ")
    if y.isnumeric():
        y = int(y)
        if y not in range(1,len(files)+1):
            print("Invalid input, try again")
            return BulkPrompt()
        else:

```

```

        bulkdb = ReadBulkFile(f"FILES/{files[y-1]}")
        if bulkdb != -1:
            if bulkdb[0] == 3:
                return files[y-1]
            else:
                print("Invalid file, try again")
                return BulkPrompt()
        else:
            print("Invalid file, try again")
            return BulkPrompt()
    else:
        print("Invalid input, try again")
        return BulkPrompt()
filename = BulkPrompt()
Clear()
print(f"You have chosen {filename}")
return [ReadBulkFile(f"FILES/{filename}")[1], customer_name]
else:
    DrugPrompt()
    while True:
        prompt = input("Add more entries (y/n):").lower()
        if prompt == 'y':
            DrugPrompt()
        else:
            Clear()
            break
    return [drugslst, customer_name]

# final bill editing menu
def FinalEditOption(db):
    sedvar = PrimaryInput(db)
    if sedvar != -1:
        temp, name = sedvar
    else:
        return -1
    print("Order summary:")
    order_db = []
    for i in temp:
        order_db.append([len(order_db)+1, i, temp[i][0], temp[i][1]])
    tabulate(
        "Sno.,ID,Name,Quantity".split(","),
        order_db
    )
    needbulkchanges = input("Do you want to make any changes (y/n): ").lower()
    if needbulkchanges == 'y':
        def needchangeprompt():
            x = input(f"Which entry you want to change [1-{len(order_db)}]: ")
            if x.isnumeric():
                x = int(x)
                if x not in range(1, len(order_db)+1):
                    print("Invalid input, try again")
                    return needchangeprompt()
                return x
            else:
                print("Invalid input, try again")
                return needchangeprompt()

        def qtyPrompt(maxqty):
            x = input("Enter quantity: ")
            if x.isnumeric():
                x = int(x)
                if x < 0:
                    print("Invalid input, try again")
                    return qtyPrompt(maxqty)
                if x == 0:
                    pass
                if x > maxqty:
                    print(f"Not enough available, taking maximum available: {maxqty}")
                    return maxqty
                else:
                    return x
            else:

```

```

        print("Invalid input, try again")
        return qtyPrompt(maxqty)

    while True:
        changeindex = needchangeprompt() - 1
        maxqtywehave = 0
        for i in db[order_db[changeindex][1]][1]:
            maxqtywehave += db[order_db[changeindex][1]][1][i]
        inputqty = qtyPrompt(maxqtywehave)
        print(f"Changed qty of {order_db[changeindex][2]} from {order_db[changeindex][-1]} to {inputqty}")
        temp[order_db[changeindex][1]][-1] = inputqty
        order_db[changeindex][-1] = inputqty
        prompt = input("Do you want to make any more changes (y/n): ").lower()
        if prompt == 'y':
            Clear()
            tabulate(
                "Sno.,ID,Name,Quantity".split(","),
                order_db
            )
        else:
            break

    Clear()
    print("Final order summary: ")
    tabulate(
        "Sno.,ID,Name,Quantity".split(","),
        order_db
    )
    x = input("Continue to Payment (y/n): ").lower()
    if x == 'y':
        Clear()
        expblk = {}
        for i in order_db:
            expblk[i[1]] = i[2:]

    def
billprint(header,data,Name,z,tax,printhead=True,linesbetweenrows=False,prependspace=0):
    widths = [len(cell) for cell in header]
    for row in data:
        for i, cell in enumerate(row):
            widths[i] = max(len(str(cell)), widths[i])
        formatted_row = ' | '.join('{:d}' % width for width in widths)
        wide = len('-'*((len(formatted_row.format(*header))+2)))
        print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
        print(prependspace*' '+'|'+SED Pharma".center(wide, " ")+"|")
        print(prependspace*' '+'|'+~~~~~".center(wide, " ")+"|")
        # print(prependspace*' '+'|'+wide*" "+"|")
        print(prependspace*' '+'| Customer: "+Name+(wide-len(Name)-33)*' '+f"{' /
'.join(str(datetime.datetime.now())[:-7].split()))' |")
        if printhead:
            print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
            print(prependspace*' '+'| '+formatted_row.format(*header)+' |')
            print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
        else:
            print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
        endcounter = 0
        for row in data:
            endcounter += 1
            if row[3] == "":
                if endcounter == len(data):
                    continue
                print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
                continue
            print(prependspace*' '+'| '+formatted_row.format(*row)+' |')
        print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
        print(prependspace*" "+"| Total cost before Tax:+" "*(wide-27-len(str(z[0])))+f"Rs
{z[0]} |")
        print(prependspace*" "+f"| Tax [{tax}%]:+" "*(wide-15-len(str(z[1])))+f"Rs {z[1]} |")
        print(prependspace*" "+"| Total cost after Tax:+" "*(wide-26-
len(str(z[1]+z[0])))+f"Rs {z[1]+z[0]} |")
        print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')

```

```

expblk = BulkRemove(expblk,db) # get the db with all info
tax = ReadDB("MANAGEMENT.DB")['tax']
finalbill = FinalBill(expblk,tax)
x_X = ['SNo', 'ID', 'Item', 'Expiry', 'Quantity', 'Rate', 'Net Price']
y_Y = []
for i in finalbill[:-1]:
    temp = list(i[3].items())
    y_Y.append([finalbill.index(i)+1,i[0],i[1],temp[0][0],temp[0][1],i[2],i[-1]])
    for i in temp[1:]:
        y_Y.append(["", "", "", i[0], i[1], "", ""])
    y_Y.append(["", "", "", "", "", "", ""])

billprint(x_X,y_Y,name,finalbill[-1],tax)

# User history store
userdb = ReadDB('USERS.DB')
AddUserLogs(name,expblk,userdb)

WriteDB(userdb, 'USERS.DB')

# Logging
Logger('Transaction made')
input("Press Enter to go to main menu")
else:
    return

```

===debug.py===

# debug script for running unit tests

```
from main import ReadDB, WriteDB
```

```
def setupnew():
    WriteDB({}, 'USERS.DB')
    WriteDB({}, 'STOCK.DB')
    WriteDB({'tax':0, 'password':'❖100❖114❖111❖119❖115❖115❖97❖112'}, 'MANAGEMENT.DB')
```

===init.py===

```
from main import *
from billingpage import *
from management import *
from stock import *
from ui import *
from users import *
from authentication import Login
```

# start database and authentication and proceed to loadingscreen & homepage if passed

```
def initialize():
    counter = 5
    while counter>0:
        if Login() != -1:
            Clear()
            db = ReadDB('STOCK.DB')
            LoadingScreen()
            HomePage(db)
            return 0
        else:
            counter -= 1
            if counter != 0:
                print(f"Password incorrect! {counter} attempts remaining.")
    return 0
```

# to avoid initialization call when externally importing

# to be executed only when directly run

```
if __name__ == "__main__":
    initialize()
```

===inventorypage.py===

```
from os import system, name
from pprint import pprint
from management import SearchWithName
```

```

from main import ReadDB,WriteDB
from stock import ReadBulkFile, BulkAdd as BulkAddToDB, GetExpired, RemoveExpired, CleanDB
from main import CheckLocalFiles

def Clear():
    if name == 'nt': _ = system('cls')
    else: _ = system('clear')

# modified tabulate function for this use case
def tabulate(header,data,printhead=True,linesbetweenrows=False,prependspace=0):
    widths = [len(cell) for cell in header]
    for row in data:
        for i, cell in enumerate(row):
            widths[i] = max(len(str(cell)), widths[i])
    formatted_row = ' | '.join('{: %d}' % width for width in widths)
    if printhead:
        print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
        print(prependspace*' '+'| '+formatted_row.format(*header)+' |')
        print(prependspace*' '+'+'+'='*(len(formatted_row.format(*header))+2)+'+')
    else:
        print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
    for row in data:
        print(prependspace*' '+'| '+formatted_row.format(*row)+' |')
        if linesbetweenrows:
            print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
        if linesbetweenrows == False: print(prependspace*' '+'+'+'-'*
(len(formatted_row.format(*header))+2)+'+')

# inventory page main menu
def MainPage(db):
    Clear()
    def takeinput():
        x = input("> ")
        if x in "1 2 3 4 0".split():
            if x == '1':
                SearchAndEditPage(db)
            if x == '2':
                WholeInventory(db)
            if x == '3':
                BulkAdd(db)
            if x == '4':
                Expired(db)
            if x == '0':
                return -1
        else:
            print("Invalid syntax, Try again:")
            takeinput()
    while True:
        print("Type the option you want to choose and press Enter [0-3]:")
        tabulate(
            "Option Service".split(),
            [
                "1,Search and Edit".split(','),
                "2,View Inventory".split(','),
                "3,Bulk add".split(','),
                "4,Expired".split(','),
                "0,Go back".split(','),
            ],
            linesbetweenrows=True,
        )
        if takeinput() == -1:
            Clear()
            break
        Clear()

        WriteDB(db,'STOCK.DB')

# search for drug and edit variables
def SearchAndEditPage(db):
    def DrugPrompt():
        Clear()
        # sequence matched drug name prompt

```

```

def DrugNameInput():
    med = input("Drug name: ")
    templist = []
    for i in SearchWithName(med,db):
        templist.append([len(templist)+1,i,db[i][0]])
    if len(templist) == 0:
        Clear()
        return -1
    return templist
templist = DrugNameInput()
if templist == -1:
    input("No results found. Press any key to continue.")
    return -1
Clear()
print(f"{len(templist)} Results Found:")
tabulate(
    "SNo.,ID,Name".split(","),
    templist)

# choice prompt for all entries from DrugNameInput
def choicePrompt():
    choice = input(f"Enter your choice [1-{len(templist)}]:")
    if choice.isnumeric():
        choice = int(choice)
        if choice not in range(1,len(templist)+1):
            print("Invalid input, try again")
            return choicePrompt()
        else: return choice
    else:
        print("Invalid input, try again")
        return choicePrompt()

drugindex = choicePrompt()-1 # relative inside templist
Clear()
print(f"You chose {templist[drugindex][2]}")

# operation prompt
Clear()
def nameEdit():
    dat = db[templist[drugindex][1]]
    Clear()
    tabulate(
        "Option Value".split(),
        [
            ["Old Name",dat[0]],
        ],
        printhead=False
    )
    while True:
        newname = input("Input new name: ")
        if newname.isalnum():
            break
        else:
            print("Invalid syntax, Try again:")
    dat[0] = newname.strip()
    tabulate(
        "Option Value".split(),
        [
            ["New Name",dat[0]],
        ],
        printhead=False
    )
    input("Press Enter to continue.")

# quantity edit option for selected drug
def qtyEdit():
    dat = db[templist[drugindex][1]]
    exp_list = []
    qty_counter = 1
    for i in dat[1].items():
        exp_list.append([qty_counter,i[0],i[1]])
        qty_counter += 1

```

```

    tabulate(
        "Sno Expiry Qty".split(),
        exp_list,
        linesbetweenrows=True,
    )
    while True:
        choice = input(f"Enter your choice [1-{len(exp_list)}]: ")
        if choice.isnumeric() and int(choice)>0 and int(choice)<len(exp_list)+1:
            choice = int(choice)
            break
        print("Invalid syntax, Try again:")
        tabulate(
            "Option Value".split(),
            [
                ["Old Qty", exp_list[choice-1][2]],
            ],
            printhead=False
        )
        while True:
            newqty = input("Input new Qty: ")
            if newqty.isnumeric():
                break
            else:
                print("Invalid syntax, Try again:")

        dat[1][exp_list[choice-1][1]] = int(newqty)

        tabulate(
            "Option Value".split(),
            [
                ["New Qty", dat[1][exp_list[choice-1][1]]],
            ],
            printhead=False
        )
        input("Press Enter to continue.")

# price edit option for selected drug
def priceEdit():
    dat = db[templist[drugindex][1]]
    Clear()
    tabulate(
        "Option Value".split(),
        [
            ["Old Price", dat[2]],
        ],
        printhead=False
    )
    while True:
        newname = input("Input new price: ")
        if newname.isnumeric():
            break
        else:
            print("Invalid syntax, Try again:")
    dat[2] = int(newname)
    tabulate(
        "Option Value".split(),
        [
            ["New price", dat[2]],
        ],
        printhead=False
    )
    input("Press Enter to continue.")

# menu for variable edit option
def takeinput():
    x = input("> ")
    if x in "0 1 2 3".split():
        if x == '1':
            return nameEdit()
        if x == '2':
            return qtyEdit()
        if x == '3':

```



```

        return priceEdit()
    if x == '0':
        return -1
    else:
        print("Invalid syntax, Try again:")
        takeinput()
while True:
    print("Type the option you want to edit and press Enter [0-3]:")
    tabulate(
        "Option Value".split(),
        [
            "1,Name".split(','),
            "2,Qty".split(','),
            "3,Price".split(','),
            "0,Go back".split(','),
        ],
        linesbetweenrows=True,
    )
    if takeinput() == -1:
        Clear()
        break
    Clear()

# actual running here
DrugPrompt()
CleanDB(db)

# parser function before using BulkAdd to db function
def BulkAdd(db):
    # list local available files
    Clear()
    input("Place bulk import file inside the FILES directory ")
    files = CheckLocalFiles()
    print(f"{len(files)} Results found:")
    tabulate(
        ["SNo.", "File", ],
        [[files.index(i)+1,i] for i in files],
        linesbetweenrows = True
    )

# prompt for choosing local files
def BulkPrompt():
    y = input(f"Enter your choice [1-{len(files)}]: ")
    if y.isnumeric():
        y = int(y)
        if y not in range(1,len(files)+1):
            print("Invalid input, try again")
            return BulkPrompt()
        else:
            bulkdb = ReadBulkFile(f"FILES/{files[y-1]}")
            if bulkdb != -1:
                if bulkdb[0] == 5:
                    return files[y-1]
                else:
                    print("Invalid file, try again")
                    return BulkPrompt()
            else:
                print("Invalid file, try again")
                return BulkPrompt()
    else:
        print("Invalid input, try again")
        return BulkPrompt()
filename = BulkPrompt()
Clear()
print(f"You have chosen {filename}")
adddb = ReadBulkFile(f"FILES/{filename}")[1] # ignore linter
temp_list = []
counter = 0
for i in adddb:
    for j in adddb[i][1]:
        temp_list.append([counter+1,i,adddb[i][0],j,adddb[i][1][j],adddb[i][-1]])
    counter += 1

```

```

tabulate("SNo ID Name Expiry Qty Cost".split(),temp_list)
# actually adding to DB
BulkAddToDB(adddb,db)
input(f"Successfully added {len(adddb)} items. Press Enter to continue.")

# print out the whole inventory with details
def WholeInventory(db):
    Clear()
    if len(db) == 0:
        input("No items found. Press Enter to continue.")
        return
    header = "ID Name Expiry Qty Cost".split()
    data = []
    for i in db.items():
        curstuf = list(i[1][1].items())
        data.append([i[0],i[1][0],curstuf[0][0],str(curstuf[0][1]),i[1][-1]])
        for j in curstuf[1:]:
            data.append([" ", " ",j[0],str(j[1])," "])

    widths = [len(cell) for cell in header]
    for row in data:
        for i, cell in enumerate(row):
            widths[i] = max(len(str(cell)), widths[i])
    formatted_row = ' | '.join('{:%d}' % width for width in widths)
    wide = len('-'*((len(formatted_row.format(*header))+2)))
    print('+++'*(len(formatted_row.format(*header))+2)+'+')
    print("|"+"Inventory".center(wide," ")+"|")
    print('+++'*(len(formatted_row.format(*header))+2)+'+')
    print('| '+formatted_row.format(*header)+' |')
    print('+++'*(len(formatted_row.format(*header))+2)+'+')
    print('| '+formatted_row.format(*data[0])+' |')
    for row in data[1:-1]:
        if row[0] == " ":
            print('| '+formatted_row.format(*row)+' |')
        else:
            print('+++'*(len(formatted_row.format(*header))+2)+'+')
            print('| '+formatted_row.format(*row)+' |')
    if data[-1][0] == " ":
        print('| '+formatted_row.format(*data[-1])+' |')
    else:
        print('+++'*(len(formatted_row.format(*header))+2)+'+')
        print('| '+formatted_row.format(*data[-1])+' |')
    print('+++'*(len(formatted_row.format(*header))+2)+'+')

    input("Press Enter to continue.")

# expired remove ui
def Expired(db):
    Clear()
    CleanDB(db)
    to_be_removed = GetExpired(db)

    if len(to_be_removed) == 0:
        input("No expired items found. Press Enter to continue.")
        return
    Clear()
    header = "ID Name Expiry Qty".split()
    data = []
    for i in to_be_removed.items():
        curstuf = list(i[1][1].items())
        data.append([i[0],i[1][0],curstuf[0][0],str(curstuf[0][1])])
        for j in curstuf[1:]:
            data.append([" ", " ",j[0],str(j[1])])
    widths = [len(cell) for cell in header]
    for row in data:
        for i, cell in enumerate(row):
            widths[i] = max(len(str(cell)), widths[i])
    formatted_row = ' | '.join('{:%d}' % width for width in widths)
    wide = len('-'*((len(formatted_row.format(*header))+2)))
    print('+++'*(len(formatted_row.format(*header))+2)+'+')
    print("|"+"Expired".center(wide," ")+"|")
    print('+++'*(len(formatted_row.format(*header))+2)+'+')

```

```

print('| '+formatted_row.format(*header)+' |')
print('+'+'='*(len(formatted_row.format(*header))+2)+'+')
print('| '+formatted_row.format(*data[0])+' |')
for row in data[1:-1]:
    if row[0] == " ":
        print('| '+formatted_row.format(*row)+' |')
    else:
        print('+'+'-'*(len(formatted_row.format(*header))+2)+'+')
        print('| '+formatted_row.format(*row)+' |')
if data[-1][0] == " ":
    print('| '+formatted_row.format(*data[-1])+' |')
else:
    print('+'+'-'*(len(formatted_row.format(*header))+2)+'+')
    print('| '+formatted_row.format(*data[-1])+' |')
print('+'+'='*(len(formatted_row.format(*header))+2)+'+')

# confirmation menu for removal
while True:
    x = input("Proceed to removal [y/n]: ")
    if x in ("Y", "y"):
        RemoveExpired(db)
        print(f"removed {len(data)} entries.")
        break
    elif x in ("N", "n"):
        print("No entries removed.")
        break
    else:
        print("Invalid syntax, Try again:")
CleanDB(db)
input("Press Enter to continue.")

```

```

===main.py===
import pickle
import os

```

```

# to write a data object to a file
def WriteDB(db, file):
    with open(f'./database/{file}', 'wb') as db_file:
        pickle.dump(db, db_file)

# to read a stored data object from a file
def ReadDB(file):
    with open(f'./database/{file}', 'rb') as db_file:
        db = pickle.load(db_file)
    return db

# get list of files that has .csv extension in the FILES dir of current folder
def CheckLocalFiles():
    files = [i for i in os.listdir("FILES") if i.endswith(".csv")]
    return files

```

```

===management.py===
from difflib import SequenceMatcher
import datetime

# manually edit name/cost of a drug from a db
def ManualEdit(ID, db, name=-1, cost=-1):
    if name != -1:
        db[ID][0] = name
    if cost != -1:
        db[ID][-1] = cost

# sequence matching search to find similar drug names
def SearchWithName(name, db):
    similar = []
    for i in db:
        if SequenceMatcher(None, name, db[i][0]).ratio() >= 0.6:
            similar.append(i)
    return similar

```

```

# logger function
def Logger(msg):
    with open('database/database.log','a') as file:
        file.write(f"[{datetime.datetime.now()}] {msg}\n")

===managementpage.py===
from os import system, name
from main import ReadDB,WriteDB
from authentication import Encoder, Decoder

def Clear():
    if name == 'nt': _ = system('cls')
    else: _ = system('clear')

# modified tabulation ui for this use case
def tabulate(header,data,printhead=True,linesbetweenrows=False,prependspace=0):
    widths = [len(cell) for cell in header]
    for row in data:
        for i, cell in enumerate(row):
            widths[i] = max(len(str(cell)), widths[i])
    formatted_row = ' | '.join('{: %d}' % width for width in widths)
    if printhead:
        print(prependspace*' '+'+'-'*(len(formatted_row.format(*header))+2)+'+')
        print(prependspace*' '+'| '+formatted_row.format(*header)+' |')
        print(prependspace*' '+'+'+'='*(len(formatted_row.format(*header))+2)+'+')
    else:
        print(prependspace*' '+'+'-'*(len(formatted_row.format(*header))+2)+'+')
    for row in data:
        print(prependspace*' '+'| '+formatted_row.format(*row)+' |')
        if linesbetweenrows:
            print(prependspace*' '+'+'-'*(len(formatted_row.format(*header))+2)+'+')
    if linesbetweenrows == False: print(prependspace*' '+'+'-'*(
(len(formatted_row.format(*header))+2)+'+')

# main menu for management page
def MainPage():
    Clear()
    def takeinput():
        x = input("> ")
        if x in "1 2 0".split():
            if x == '1':
                TaxUpdate()
            if x == '2':
                PasswordUpdate()
            if x == '0':
                return -1
        else:
            print("Invalid syntax, Try again:")
            takeinput()
    while True:
        print("Type the option you want to choose and press Enter [0-2]:")
        tabulate(
            "Option Service".split(),
            [
                "1,Edit tax value".split(','),
                "2,Change access password".split(','),
                "0,Go back".split(','),
            ],
            linesbetweenrows=True,
        )
        if takeinput() == -1:
            Clear()
            break
    Clear()

# update tax in MANAGEMENT.DB from user input
def TaxUpdate():
    Clear()
    dbfile = ReadDB('MANAGEMENT.DB')
    tabulate(
        "Option Service".split(),

```

```

        [
            ["Old tax value",dbfile["tax"]],
        ],
        printhead=False
    )
    while True:
        tax = input("Input new tax percentage value: ")
        if tax.isnumeric():
            tax = int(tax)
            break
        else:
            print("Invalid syntax, Try again:")

    dbfile['tax'] = tax
    tabulate(
        "Option Service".split(),
        [
            ["New tax value",dbfile["tax"]],
        ],
        printhead=False
    )
    input("Press Enter to continue.")
    WriteDB(dbfile,'MANAGEMENT.DB')

# update password and store in MANAGEMENT.DB after encrypting
def PasswordUpdate():
    Clear()
    dbfile = ReadDB('MANAGEMENT.DB')
    tabulate(
        "Option Service".split(),
        [
            ["Old password",Decoder(dbfile["password"])],
        ],
        printhead=False
    )
    pas = input("Input new password: ")

    dbfile['password'] = Encoder(pas)
    tabulate(
        "Option Service".split(),
        [
            ["New password",Decoder(dbfile["password"])],
        ],
        printhead=False
    )
    input("Press Enter to continue.")
    WriteDB(dbfile,'MANAGEMENT.DB')

```

===reportpage.py===

```

from main import ReadDB
from os import system, name

def Clear():
    if name == 'nt': _ = system('cls')
    else: _ = system('clear')

# modified tabulation ui for this use case
def tabulate(header,data,printhead=True,linesbetweenrows=False,prependspace=0):
    widths = [len(cell) for cell in header]
    for row in data:
        for i, cell in enumerate(row):
            widths[i] = max(len(str(cell)), widths[i])
    formatted_row = ' | '.join('{:%d}' % width for width in widths)
    if printhead:
        print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
        print(prependspace*' '+'| '+formatted_row.format(*header)+' | ')
        print(prependspace*' '+'+'+'='*(len(formatted_row.format(*header))+2)+'+')
    else:
        print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
    for row in data:

```

```

        print(prependspace*' '+'| '+formatted_row.format(*row)+' |')
        if linesbetweenrows:
            print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
        if linesbetweenrows == False: print(prependspace*' '+'+'+'-'*
(len(formatted_row.format(*header))+2)+'+')

# print graph from graphing data provided
def BarGrapher(data_,header='Drug Sales Chart'):
    Clear()
    data = {}
    maxlen = max(max(len(i) for i in data_),5)
    mindata = int(min(data_.values()))
    maxdata = int(max(data_.values()))
    for i in data_:
        data[i] = int((74-maxlen)*(data_[i]-mindata+1)/(maxdata-mindata+1))
        if data[i] ≤ 0:
            data[i] = 1
    print('+'+'='*80+'+')
    print('| '+header.center(79)+'|')
    print('+'+'='*(maxlen+2)+'+'+(77-maxlen)*'-'+'+')
    for i in data:
        print('| '+i.rjust(maxlen)+' | '+'█'*data[i]+' '* (76-maxlen-data[i])+'|')
        print('+'+'-'*(maxlen+2)+'+'+(77-maxlen)*'-'+'+')

    x_axis = [mindata,mindata+(maxdata-mindata)/4,(maxdata-mindata)/2,maxdata-(maxdata-
mindata)/4,maxlen]
    axis_len_sum = 0
    for i in x_axis:
        axis_len_sum += len(str(i))
    l = ''
    for i in x_axis[:-1]:
        l += str(int(i))+" "*int((76-maxlen-axis_len_sum)/4)
    l += str(maxdata)
    print('| '+ "Sales".center(maxlen)+'→'+l.center(76-maxlen)+'|')
    print('+'+'='*80+'+')
    input("Press Enter to continue.")

# print date-wise salesgraph from userdb
def SalesGraph(userdb):
    temp_db = {}
    for i in userdb:
        for j in userdb[i]:
            for k in userdb[i][j]:
                if i in temp_db:
                    temp_db[i] += userdb[i][j][k][-1][0]
                else:
                    temp_db[i] = userdb[i][j][k][-1][0]
    BarGrapher(temp_db,'Sales Report')

# main menu for report page
def MainPage():
    temp_db={}
    userdb = ReadDB('USERS.DB')
    if len(userdb) == 0:
        Clear()
        input("Insufficent data available. Press enter to continue.")
        return -1
    for i in userdb.items():
        for j in i[1].items():
            for k in j[1].items():
                for l in k[1][:-1]:
                    if l[1] in temp_db:
                        temp_db[l[1]] += l[-1]
                    else:
                        temp_db.update({l[1]:l[-1]})
    Clear()
    def takeinput():
        x = input("> ")
        if x in "1 2 0".split():
            if x == '1':
                BarGrapher(temp_db)
            if x == '2':

```

```

        SalesGraph(userdb)
    if x == '0':
        return -1
    else:
        print("Invalid syntax, Try again:")
        takeinput()
while True:
    print("Type the option you want to choose and press Enter [0-2]:")
    tabulate(
        "Option Service".split(),
        [
            "1,Drug sales chart".split(','),
            "2,Sales Report".split(','),
            "0,Go back".split(','),
        ],
        linesbetweenrows=True,
    )
    if takeinput() == -1:
        Clear()
        break
    Clear()

```

==run.py==

```

import init
import os
# checks if windows system and runs the program in a cmd instance
if os.name == 'nt':
    print("Program is running in another cmd window.")
    os.system('start cmd /K py init.py')
else:
    init.initialize()
# if its unix/unix-like system it runs it in the current shell session

```

==stock.py==

```

import csv
import datetime

# helper function to sort date
def DateSorter(date):
    date = [int(i) for i in date[0].split('-')]
    return (date[1]*100)+date[0]

# to check if a date is expired by comparing with current date
def IsExpired(date) :
    td = datetime.date.today()
    d = [int(i) for i in date.split('-')]
    if d[1]<td.year:
        return True
    if d[1] == td.year and d[0]≤td.month:
        return True
    return False

# get dict of expired stuff from db
def GetExpired(db):
    expireddb = {}
    for ID in db:
        x = {}
        for i in db[ID][1]:
            if IsExpired(i):
                x.update({i:db[ID][1][i]})
        expireddb.update({ID:[db[ID][0],x]})
        if len(expireddb[ID][1]) == 0:
            del expireddb[ID]
    return expireddb

# remove all expired stuff from db
def RemoveExpired(db):
    for ID in db:
        temp = db[ID][1].copy()
        for i in db[ID][1]:

```

```

        if IsExpired(i):
            del temp[i]
        db[ID][1] = temp

# remove all drug entries with qty ≤ 0 from db
def CleanDB(db):
    for item_id in db:
        for i in list(db[item_id][1].items()):
            if i[1] ≤ 0:
                del db[item_id][1][i[0]]
    for j in list(db.items()):
        if j[1][1] == {}:
            del db[j[0]]

# add single item to db
def ItemAdd(item_id,item_info,db):
    if item_id in db:
        for i in item_info[1]:
            if i in db[item_id][1]:
                db[item_id][1][i] += item_info[1][i]
            else:
                db[item_id][1][i] = item_info[1][i]
    else:
        db[item_id] = item_info
    for j in db:
        db[j][1] = dict(sorted(db[j][1].items(),key=DateSorter))

# remove single item from db
def ItemRemove(ID,qty,db):
    if ID in db:
        rdict = {}
        for i in db[ID][1]:
            if db[ID][1][i]<qty:
                rdict[i] = db[ID][1][i]
                qty -= db[ID][1][i]
                db[ID][1][i] = 0
            else:
                rdict[i] = qty
                db[ID][1][i] -= qty
                CleanDB(db)
        return rdict
    CleanDB(db)
    return rdict
else:
    return -1

# adds a dict of items (existant & non-existant) items to to_dict
def BulkAdd(from_dict,to_dict):
    for i in from_dict:
        ItemAdd(i,from_dict[i],to_dict)

# removes a dict of items (existant) from to_dict
# returns a list with dicts of expiry:qty if the item exists else -1
def BulkRemove(from_dict,to_dict):
    rlist = []
    for i in from_dict:
        if i in to_dict:
            rlist.append([i,to_dict[i][0],to_dict[i][-1],ItemRemove(i,from_dict[i][1],to_dict)])
    return rlist

# Read csv file and return a dict of items. different types of dicts are
# returned based on row length. if invalid csv is passed returns -1
def ReadBulkFile(file):
    data = []
    data_dict = {}
    with open(file,'r') as csvfile:
        reader = csv.reader(csvfile)
        try:
            column_len = len(next(reader))
        except StopIteration:
            return -1
        for i in reader:

```



[illegible]

```

        inventorypage.MainPage(db)
    if x == '3':
        managementpage.MainPage()
    if x == '4':
        userhistorypage.MainPage()
    if x == '5':
        reportpage.MainPage()
    if x == '0':
        return -1
else:
    print("Invalid syntax, Try again:")
    takeinput()
while True:
    print("Type the option you want to choose and press Enter [0-5]:")
    tabulate(
        "Option Service".split(),
        [
            "1 Billing".split(),
            "2 Inventory".split(),
            "3 Management".split(),
            "4,User History".split(','),
            "5 Reports".split(),
            "0 Quit".split(),
        ],
        linesbetweenrows=True,
    )

    WriteDB(db, 'STOCK.DB')

    if takeinput() == -1:
        Clear()
        break
    Clear()

```

===userhistorypage.py===

```

from os import system, name
from main import ReadDB

def Clear():
    if name == 'nt': _ = system('cls')
    else: _ = system('clear')

# modified tabulation ui for this use case
def tabulate(header,data,printhead=True,linesbetweenrows=False,prependspace=0):
    widths = [len(cell) for cell in header]
    for row in data:
        for i, cell in enumerate(row):
            widths[i] = max(len(str(cell)), widths[i])
    formatted_row = ' | '.join('{:%d}' % width for width in widths)
    if printhead:
        print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
        print(prependspace*' '+'| '+formatted_row.format(*header)+' |')
        print(prependspace*' '+'+'+'='*(len(formatted_row.format(*header))+2)+'+')
    else:
        print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
    for row in data:
        print(prependspace*' '+'| '+formatted_row.format(*row)+' |')
        if linesbetweenrows:
            print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
    if linesbetweenrows == False: print(prependspace*' '+'+'+'-'*
(len(formatted_row.format(*header))+2)+'+')

# main menu for user history page
def MainPage():
    userdb = ReadDB('USERS.DB')
    Clear()
    def takeinput():
        x = input("> ")
        if x in "1 2 3 0".split():
            if x == '1':
                SearchByUser(userdb)

```

```

        if x == '2':
            SearchByDate(userdb)
        if x == '0':
            return -1
    else:
        print("Invalid syntax, Try again:")
        takeinput()
while True:
    print("Type the option you want to choose and press Enter [0-2]:")
    tabulate(
        "Option Service".split(),
        [
            "1,Search by User".split(', '),
            "2,Search by Date".split(', '),
            "0,Go back".split(', '),
        ],
        linesbetweenrows=True,
    )
    if takeinput() == -1:
        Clear()
        break
    Clear()

# to get user purchases of a specified username
def SearchByUser(udb):
    Clear()
    username = input('Enter Username that you want to check: ').title()
    templist = []
    date_list = []
    for i in udb.items():
        for j in i[1].items():
            for k in j[1].items():
                if k[0] == username:
                    templist.append(k[1])
                    date_list.append([i[0], j[0]])
    if len(templist) == 0:
        print(f"No Transactions available for {username}.")
    else:
        print(f"{len(templist)} Transactions found: ")
    counter = 0
    for finalbill in templist:
        tax = ReadDB("MANAGEMENT.DB")['tax']
        x_X = ['SNo', 'ID', 'Item', 'Expiry', 'Quantity', 'Rate', 'Net Price']
        y_Y = []
        for i in finalbill[:-1]:
            temp = list(i[3].items())
            y_Y.append([finalbill.index(i)+1, i[0], i[1], temp[0][0], temp[0][1], i[2], i[-1]])
            for i in temp[1:]:
                y_Y.append(["", "", "", i[0], i[1], "", ""])
            y_Y.append(["", "", "", "", "", "", ""])
        billprint(x_X, y_Y, username, finalbill[-1], tax, date_list[counter])
        counter += 1
    print()
    input("Press Enter to continue.")

# to get purchases of a specified date
def SearchByDate(udb):
    date = input('Enter date that you want to check[dd-mm-yy]: ')
    namelist = []
    billlist = []
    date_list = []
    for i in udb.items():
        if i[0] == date:
            for j in i[1].items():
                for k in j[1].items():
                    namelist.append(k[0])
                    billlist.append(k[1])
                    date_list.append([i[0], j[0]])
    if len(namelist) == 0:
        print(f"No Transactions available for {date}.")
    else:
        print(f"{len(namelist)} Transactions found: ")

```

```

for k in range(len(namelist)):
    finalbill=billlist[k]
    username=namelist[k]
    tax = ReadDB("MANAGEMENT.DB")['tax']
    x_X = ['SNo', 'ID', 'Item', 'Expiry', 'Quantity', 'Rate', 'Net Price']
    y_Y = []
    for i in finalbill[:-1]:
        temp = list(i[3].items())
        y_Y.append([finalbill.index(i)+1,i[0],i[1],temp[0][0],temp[0][1],i[2],i[-1]])
        for i in temp[1:]:
            y_Y.append(["", "", "", "", i[0], i[1], "", ""])
            y_Y.append(["", "", "", "", "", "", ""])
    billprint(x_X,y_Y,username,finalbill[-1],tax,date_list[k])
    print()
input("Press Enter to continue.")

# print bill of each transaction found in SearchByUser and SearchByDate functions
def billprint(header,data,Name,z,tax,dat,printhead=True,prependspace=0):
    widths = [len(cell) for cell in header]
    for row in data:
        for i, cell in enumerate(row):
            widths[i] = max(len(str(cell)), widths[i])
    formatted_row = ' | '.join('{: %d}' % width for width in widths)
    wide = len('-'*len(formatted_row.format(*header))+2)
    print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
    print(prependspace*' '+'|"SED Pharma".center(wide," ")+"|")
    print(prependspace*' '+'|"~~~~~".center(wide," ")+"|")
    # print(prependspace*' '+'|'+wide*' '+'|')
    print(prependspace*' '+'| Customer: "+Name+(wide-len(Name)-33)*' '+f' {' / '.join(dat)} |")
    if printhead:
        print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
        print(prependspace*' '+'| '+formatted_row.format(*header)+' |')
        print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
    else:
        print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
    endcounter = 0
    for row in data:
        endcounter += 1
        if row[3] == "":
            if endcounter == len(data):
                continue
            print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
            continue
        print(prependspace*' '+'| '+formatted_row.format(*row)+' |')
    print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')
    print(prependspace*" "+"| Total cost before Tax:"+" *(wide-27-len(str(z[0])))+f"Rs {z[0]} |")
    print(prependspace*" "+f"| Tax [{tax}%]:"+" *(wide-15-len(str(z[1])))+f"Rs {z[1]} |")
    print(prependspace*" "+"| Total cost after Tax:"+" *(wide-26-len(str(z[1]+z[0])))+f"Rs {z[1]+z[0]} |")
    print(prependspace*' '+'+'+'-'*(len(formatted_row.format(*header))+2)+'+')

```

==users.py==

```
import datetime
```

```
# get formatted date
```

```
def Today():
```

```
    today= datetime.datetime.now()
```

```
    return [today.strftime("%d-%m-%y"),today.strftime("%X")]
```

```
# add user logs (name,purchased items) to userlogdb
```

```
def AddUserLogs(name,purchase_list,userlogdb):
```

```
    name = name.strip().title()
```

```
    if Today()[0] in userlogdb:
```

```
        userlogdb[Today()[0]].update({Today()[1]:{name:purchase_list}})
```

```
    else:
```

```
        userlogdb[Today()[0]] = {Today()[1]:{name:purchase_list}}
```

```
# purchase_list is modified to a format which can be parsed for bill generation
```

```
def FinalBill(purchase_list,tax):
```

```
    totalcost = 0
```

```
    for i in purchase_list:
```

```
count = 0
for j in i[3]:
    count += i[3][j]
costperitem = count*i[2]
i.append(costperitem)
totalcost += costperitem
purchase_list.append([float(totalcost),float(totalcost)*tax/100])
return purchase_list
```