ST. ANTONY'S PUBLIC SCHOOL

CBSE AFFILIATION NO: 930037, SCHOOL NO: 75036

CLASS XII

RECORD OF PROJECT WORK IN

COMPUTER SCIENCE

Name : ----------

Roll No : ----------

ST. ANTONY'S PUBLIC SCHOOL

CBSE AFFILIATION NO: 930037, SCHOOL NO: 75036

Project Report submitted in fulfilment of Class XII

Syllabus Requirement By

_____

CERTIFICATE – TERM 1

This is to certify that this Project titled

_____

is the record of bonafide project work carried out by

_____

Roll No. _____ of class XII, Section _____ , during the academic year 2021 – 2022

Teacher in Charge Principal Examiner

ST. ANTONY'S PUBLIC SCHOOL

CBSE AFFILIATION NO: 930037, SCHOOL NO: 75036

Project Report submitted in fulfilment of Class XII

Syllabus Requirement By

_____

CERTIFICATE – TERM 2

This is to certify that this Project titled

_____

is the record of bonafide project work carried out by

_____

Roll No. _____ of class XII, Section _____ , during the academic year 2021 – 2022

Teacher in Charge Principal Examiner

ACKNOWLEDGEMENT

This project is the fruit of the diligent labour that has been invested in it by our group. Apart from our personal hard work, this project has received great support from the school, without which its presentation would have been impossible. We take this opportunity to thank our manager Rev. Fr. Darwin Valumannel, our Principal Fr. Joshy Sebastian, Vice Principal Fr. Manu K Mathew and our teachers who have helped us in our endeavor. A special mention is to make about our computer teachers, Mr. Cherian K Abraham, and Mrs. Thanuja Mathew in this regard. We would also like to thank our friends for their co-operation and suggestions. Finally, we thank the Almighty for all his blessings

ABSTRACT

Pharmacy Management System is a system which maintains the information about the drugs present in stock, quantities, the expiry of the drug, use history and all. This is very difficult to organise manually. Maintenance of all these information manually is a very complex task. Owing to the advancement of technology, organisation of a Digital Pharmacy Management System is a need of the hour. The Digital Pharmacy Management has been designed to computerise and automate the operations performed over the information about Pharmaceuticals, bulk add and remove from stock and all other operations. This computerisation of the Pharmacy helps in many instances of its maintenances. It reduces the workload of the management as most of the manual work done is reduced and automate through the Digital system.

CONTENTS

TERM I

TERM II

(1) INTRODUCTION

(1.1) PROJECT AIMS AND OBJECTIVES

The project aims and objectives that will be achieved after completion of this project are discussed in this subchapter.
The aims and objectives are as follows:

- Add items/ price to the store
- Remove items from the store
- Check the availability of items in the store,
- Clear the store data
- Admin login credentials
- Customer bill generation

(1.2) BACKGROUND OF THE PROJECT

Digital Pharmacy Management System is an application which refers to library systems which are generally medium or big in size. It is used by the pharmacist to manage the pharmacy using a computerised system where he/she can open a portal for preparing and order for the customer for pharmaceutical items. Report & Maintenance modules are also included in this system which would keep track of sales reports of the pharmacy and also a detailed description about user history as well as the ability to modify global values such as tax etc.. With this computerised system there will be no loss of user data or any other details which generally happens when a non-computerised system is used. All these modules will help the help the pharmacist to manage the pharmacy in a convenient and efficient way as compared to the present manual system.

(2) SYSTEM ANALYSIS

In this section, we would be discussing and analysing about the developing process of Pharmacy Billing and Inventory Management System including software requirement specification (SRS) and comparison between existing and proposed system. The functional and nonfunctional requirements are included in SRS part to provide complete description and overview of system requirement before the developing process is carried out. Besides that, existing vs proposed provides a view of how the proposed system will be more efficient than the existing one.

(2.1) SOFTWARE REQUIREMENT SPECIFICATION:

Pharmacy billing and inventory management system is a computerised system which helps the user (Pharmacist) to manage the daily activities in the pharmacy in an electronic format. It reduces the risk of paper work such as loss of registers, damage of registers and other time consuming activities in the pharmacy. It can help user to manage the transaction or record more effectively and in a time saving manner.

(2.2) PROBLEM STATEMENT:

The problem occurred before having computerised system includes:

- Loss of registers: If computerised system is not implemented, records are always under the threat of mishandling. There is a high chance of human error which can cause loss of records.
- Damaged records: If a computerised system is not in existence, there is high risk of losing the records due to spilling of water by some members accidentally. Also natural disasters are a high threat to hard copy of records.
- Difficult to search record: When hard copy of records are stored for a very long time it would become very difficult to search some particular record.
- Space Consuming: When number of records start increasing, hard copy of those records could be space consuming.
- Cost Consuming: When records are stored as hard copy, the hard copy records/registers would be required every year. This will increase the cost for management of a Pharmacy.

(2.3) EXISTING VS PROPOSED:

- Existing System: Early day pharmacies are managed manually. It was time consuming as it needs a lot of time to record or retrieve the details. The pharmacist had to take their jobs very seriously as a small mistake could create a lot of problems. Also security of information is very less. Maintenance of Pharmacy catalogue and arrangement of the catalogue is a very complex task. In addition to its maintenance of member details, issue dates and return dates etc. manually is a complex task.
- Proposed System: To solve the inconveniences as mentioned in the existing system, a Digital Pharmacy is proposed. The proposed system contains the following features:
    - Get drug stock information
    - Update drug details
    - User data and history fetching
    - Sales Reports
    - Produce medicine reports, etc..

(2.4) HARDWARE & SOFTWARE SPECIFICATIONS:

The following is the specification of the system on which the software has been developed:

- System Specification:
    - Operating System: Linux, MacOS, Windows 7/10/11: The software has been developed in Linux as it is very versatile and provides a wide variety of tools for testing and developing software. The software runs on almost all Unix, Unix-like and Windows systems with minimal dependancies and overheads.
    - Machine: Pentium Dual Core Processor 2.6 GHz or above, 2 GB RAM or above ,500 GB Hard Disk or above. We used Intel core i5 2nd generation based system, it is fast than other processors and provide reliable and stable performance and we can run our pc for longtime. By using this processor, we can keep on developing our project without any worries. 4GB RAM is used as it will provide fast reading and writing capabilities and will support in processing.
- Software Specification:
    - Frontend used: Python 3.8+
    - Backend used: Data files (Binary, CSV, Text)

(3) SYSTEM DESIGN

(3.1) DATA FILE / TABLE DESIGN:

- STOCK.DB

ID	Name	Expiry	Qty	Cost
XXX1	Drug_1	Expiry_1 Expiry_2 Expiry_N	XX XX XX	XXX
XXX2	Drug_2	Expiry_N	XX	XXX
XXX3	Drug_3	Expiry_N	XX	XXX
XXXN	Drug_N	Expiry_N	XX	XXX

- MANAGEMENT.DB

Variable	Value
Tax	xx
Password (encrypted)	**********

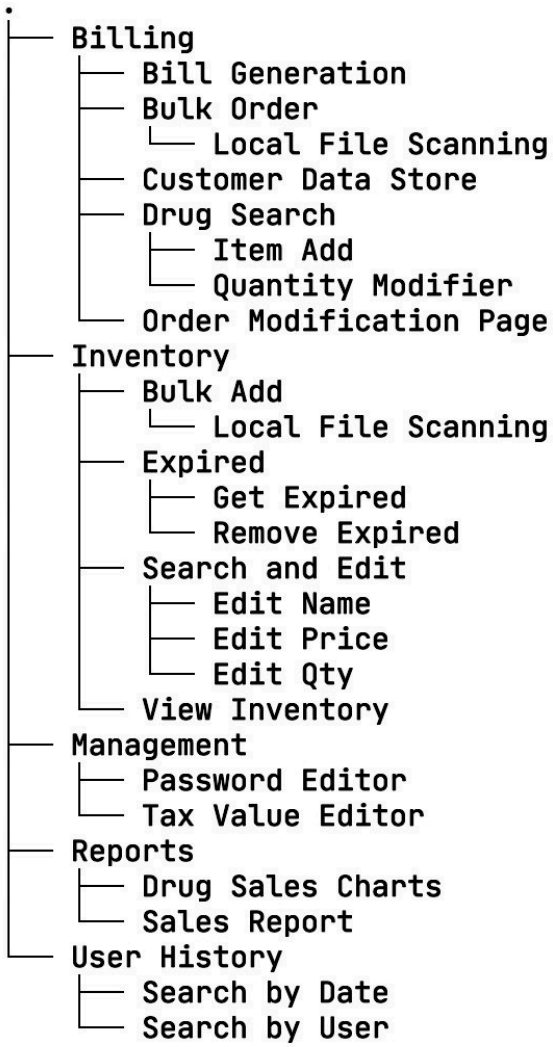
- USERS.DB

Date	Time	Username	Data
dd-mm-yy_1	xx:yy_1 xx:yy_2 xx:yy_N	Username_1 Username_2 Username_3	data_record_1 data_record_2 data_record_3
dd-mm-yy_2	xx:yy_N	Username_N	data_record_N
dd-mm-yy_3	xx:yy_N	Username_N	data_record_N
dd-mm-yy_N	xx:yy_N	Username_N	data_record_N

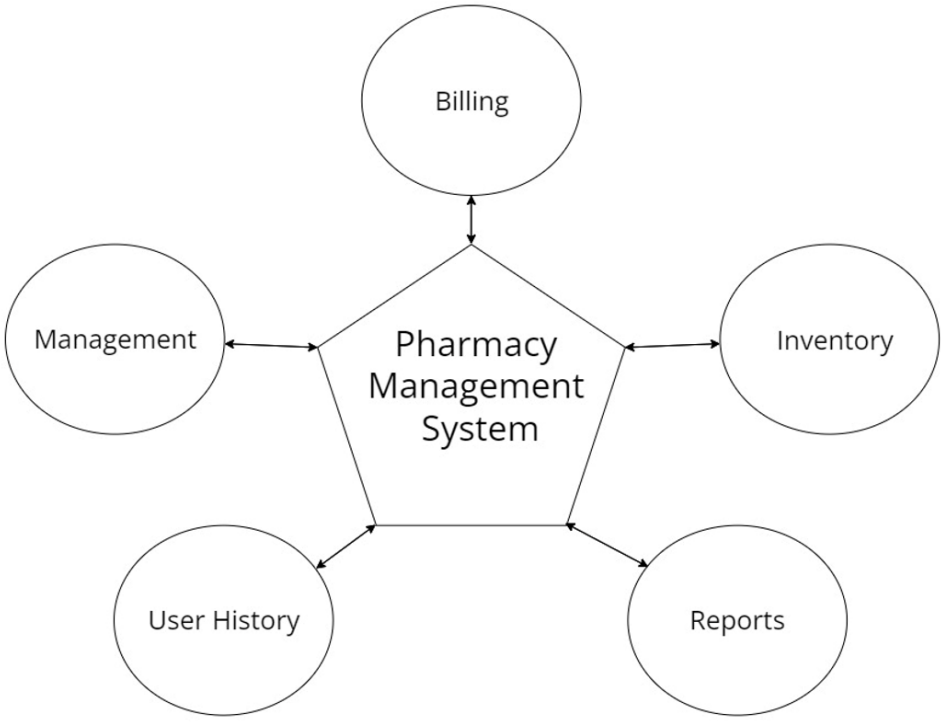
- DATABASE.LOG

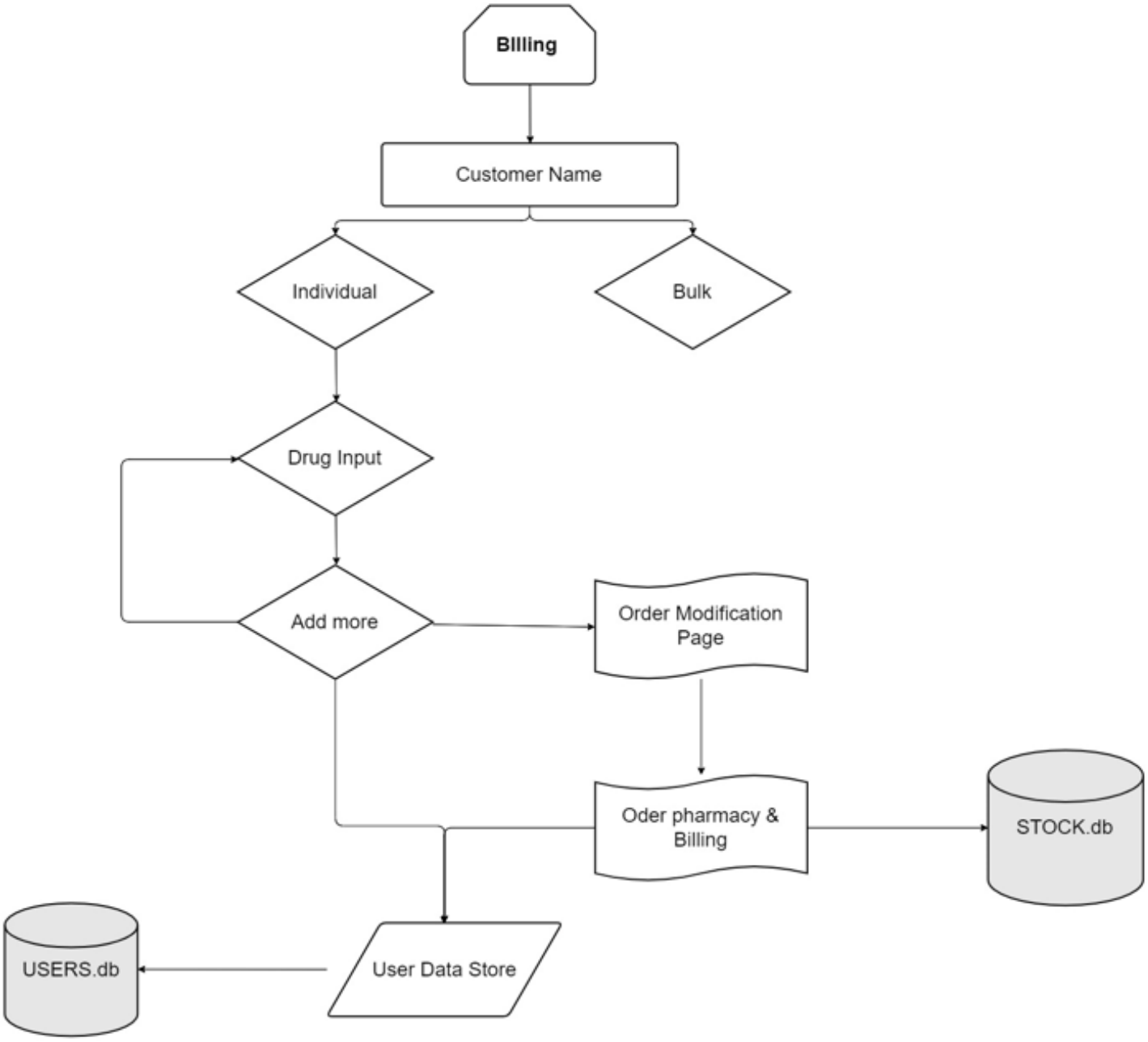
Logging Time	Logging Data
[yyyy-mm-dd hh:mm:ss.xxxxx]	Transaction made
[yyyy-mm-dd hh:mm:ss.xxxxx]	Database updated
[yyyy-mm-dd hh:mm:ss.xxxxx]	Etc..

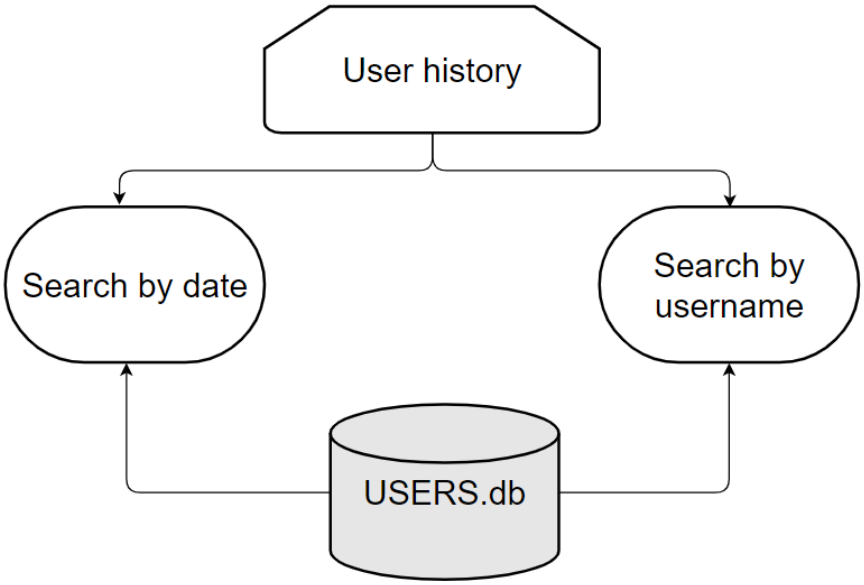
(3.2) MENU STRUCTURE:

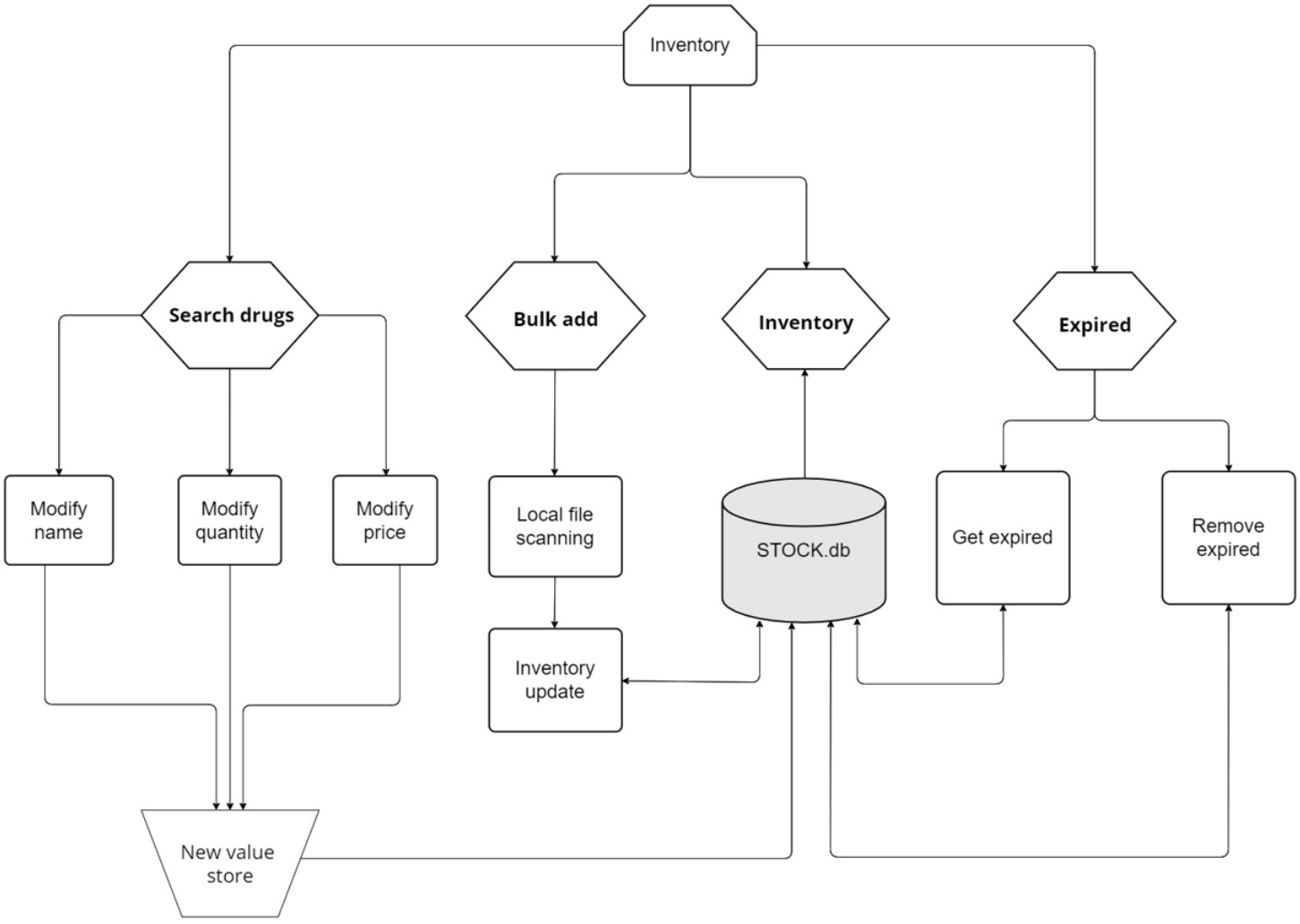


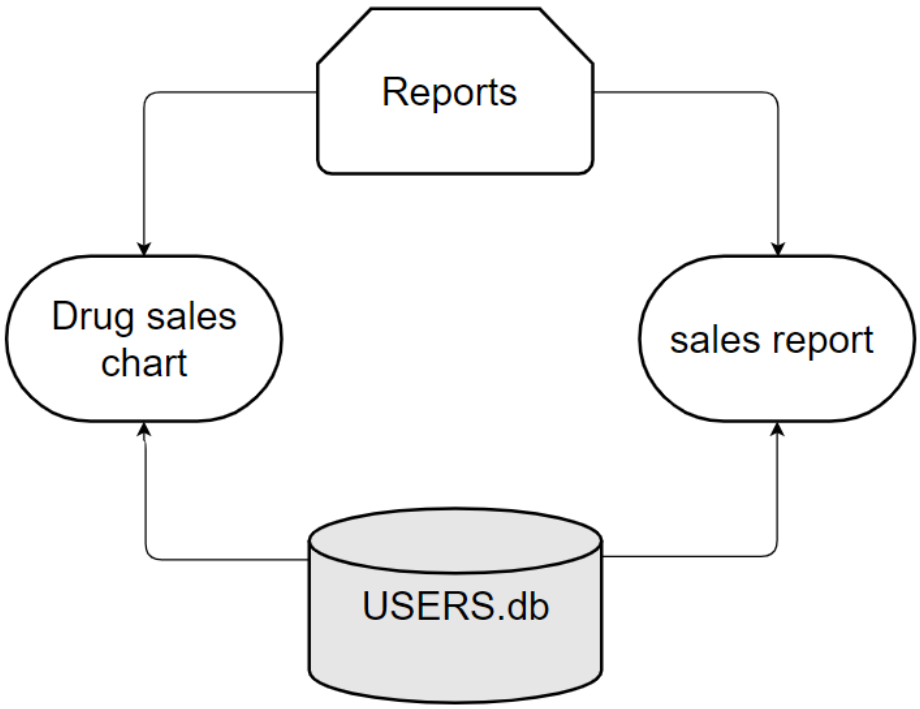
(3.3) DATA FLOW DIAGRAMS:

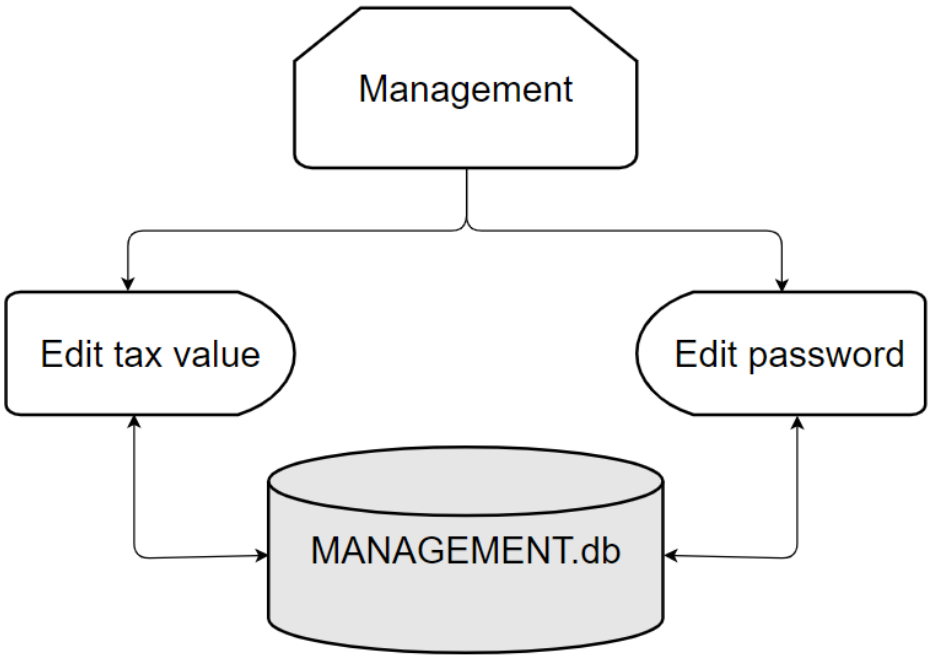



BIlling
Customer Name
Individual
Bulk
Drug Input
Add more
Order Modification Page
Oder pharmacy & Billing
STOCK.db
USERS.db
User Data Store


User history
Search by date
Search by username
USERS.db


Inventory
Search drugs
Bulk add
Inventory
Expired
Modify name
Modify quantity
Modify price
Local file scanning
STOCK.db
Get expired
Remove expired
Inventory update
New value store


Reports
Drug sales chart
sales report
USERS.db


Management
Edit tax value
Edit password
MANAGEMENT.db

(4) SYSTEM IMPLEMENTATION

(4.1) SOURCE CODE AND MODULE DESCRIPTION

```
═══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 [drugslist,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,printheader=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 printheader:
                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':'100114111119115115971121'},'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')

# modifed tabulate function for this use case
def tabulate(header,data,printheader=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 printheader:
        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]],
        ],
        printheader=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]],
        ],
        printheader=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]],
        ],
        printheader=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]]],
        ],
        printheader=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]],
        ],
        printheader=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]],
        ],
        printheader=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 chosing 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,printheader=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 printheader:
        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"]],
        ],
        printheader=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"]],
        ],
        printheader=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"])],
        ],
        printheader=False
    )
    pas = input("Input new password: ")

    dbfile['password'] = Encoder(pas)
    tabulate(
        "Option Service".split(),
        [
            ["New password",Decoder(dbfile["password"])],
        ],
        printheader=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,printheader=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 printheader:
        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:
```

```
            data.append([int(j) if j.strip().isnumeric() else j.strip() for j in i])
        if column_len == 5:
            for i in data:
                data_dict[i[0]] = [i[1],{i[3]:i[2]},i[4]]
        elif column_len == 3:
            for i in data:
                data_dict[i[0]] = [i[1],i[2]]
        else:
            data_dict = -1
    return (column_len,data_dict)

==ui.py==
import time
from os import system, name

import billingpage
from main import WriteDB
import inventorypage
import userhistorypage
import reportpage
import managementpage

# function to clear screen based on diff operating systems
def Clear():
    if name == 'nt': _ = system('cls')
    else: _ = system('clear')

# custom tabulation ui for printing tables
def tabulate(header,data,printheader=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 printheader:
        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)+'+')


# loading screen print function
def LoadingScreen():
    Prepend = ' '*8
    for i in range(58):
        print(f"""
{Prepend} ____   _____  ____       ____   _
{Prepend}/ ___| | ____||  _ \     |  _ \ | |__     __ _   _ __   _ __ ___     __ _
{Prepend}\___ \ |  _|  | | | |    | |_) || '_ \   / _` || '__|| '_ ` _ \   / _` |
{Prepend} ___) || |___ | |_| |    |  __/ | | | || (_| || |   | | | | | || (_| |
{Prepend}|____/ |_____||____/     |_|    |_| |_| \__,_||_|   |_| |_| |_| \__,_|\n\n""")
        print(f"{Prepend}LOADING: [{'#'*i+' '*(58-i)}]")
        time.sleep(0.05)
        Clear()

# main home page menu
# redirects to different sub-menus according to user input
def HomePage(db):
    def takeinput():
        x = input("> ")
        if x in "1 2 3 4 5 0".split():
            if x == '1':
                Clear()
                billingpage.FinalEditOption(db)
            if x == '2':
```

```
                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,printheader=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 printheader:
        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,printheader=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 printheader:
        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
```

(4.2) SCREENSHOTS

```
Type the option you want to choose and press Enter [0-5]:
+------------------------+
| Option | Service       |
+========================+
| 1      | Billing       |
+------------------------+
| 2      | Inventory     |
+------------------------+
| 3      | Management    |
+------------------------+
| 4      | User History  |
+------------------------+
| 5      | Reports       |
+------------------------+
| 0      | Quit          |
+------------------------+
❯
```

```
Customer name: Pranav
Bulk order (y/n): y
Place bulk order list file inside the FILES directory
5 Results found:
+--------------------------------+
| SNo. | File                    |
+================================+
|    1 | temp_a.csv              |
+--------------------------------+
|    2 | bigdata_bulkremove.csv  |
+--------------------------------+
|    3 | temp_d.csv              |
+--------------------------------+
|    4 | NEW_STOCK.csv           |
+--------------------------------+
|    5 | bigdata_bulkadd.csv     |
+--------------------------------+
Enter your choice [1-5]:
```

```
Place bulk import file inside the FILES directory
5 Results found:
+-------------------------------+
| SNo. | File                   |
+===============================+
|    1 | temp_a.csv             |
+-------------------------------+
|    2 | bigdata_bulkremove.csv |
+-------------------------------+
|    3 | temp_d.csv             |
+-------------------------------+
|    4 | NEW_STOCK.csv          |
+-------------------------------+
|    5 | bigdata_bulkadd.csv    |
+-------------------------------+
Enter your choice [1-5]:


1 Results Found:
+---------------------------------------+
| SNo. | ID   | Name        | Available |
+=======================================+
|    1 | I001 | Paracetamol |        10 |
+---------------------------------------+
Enter your choice [1-1]:1
You chose Paracetamol
Enter quantity: 10
Add more entries (y/n):


Order summary:
+---------------------------------------+
| Sno. | ID   | Name        | Quantity |
+=======================================+
|    1 | I001 | Paracetamol |       10 |
|    2 | I010 | Vinblastine |       10 |
|    3 | I007 | Marinol     |        8 |
+---------------------------------------+
Do you want to make any changes (y/n): y
Which entry you want to change [1-3]: 1
Enter quantity: 9
Changed qty of Paracetamol from 10 to 9
Do you want to make any more changes (y/n):
```

```
+=================================================================+
|                           SED Pharma                            |
|                           ~~~~~~~~~~                            |
| Customer: Pranav                          2022-01-18 / 21:20:29 |
+-----------------------------------------------------------------+
| SNo | ID   | Item        | Expiry  | Quantity | Rate | Net Price |
+=================================================================+
|   1 | I001 | Paracetamol | 11-2021 |        9 |  375 |      3375 |
+-----------------------------------------------------------------+
|   2 | I010 | Vinblastine | 11-2021 |       10 |  600 |      6000 |
+-----------------------------------------------------------------+
|   3 | I007 | Marinol     | 11-2021 |        8 |  100 |       800 |
+=================================================================+
| Total cost before Tax:                                Rs 10175.0 |
| Tax [12%]:                                             Rs 1221.0 |
| Total cost after Tax:                                 Rs 11396.0 |
+=================================================================+
Press Enter to go to main menu

Type the option you want to choose and press Enter [0-3]:
+--------------------------+
| Option | Service         |
+==========================+
| 1      | Search and Edit |
+--------------------------+
| 2      | View Inventory  |
+--------------------------+
| 3      | Bulk add        |
+--------------------------+
| 4      | Expired         |
+--------------------------+
| 0      | Go back         |
+--------------------------+
❯
Type the option you want to edit and press Enter [0-3]:
+------------------+
| Option | Value   |
+==================+
| 1      | Name    |
+------------------+
| 2      | Qty     |
+------------------+
| 3      | Price   |
+------------------+
| 0      | Go back |
+------------------+
❯
```

```
+=============================================+
|                  Expired                    |
+---------------------------------------------+
| ID   | Name                 | Expiry  | Qty |
+=============================================+
| I004 | Heparin              | 11-2021 | 40  |
+---------------------------------------------+
| I007 | Marinol              | 11-2021 | 12  |
+---------------------------------------------+
| I001 | Dolo                 | 11-2021 | 11  |
+---------------------------------------------+
| I011 | Fentanyl             | 11-2021 | 90  |
+---------------------------------------------+
| I015 | Acetylsalicylic acid | 11-2021 | 10  |
+---------------------------------------------+
| I010 | Vinblastine          | 11-2021 | 10  |
+=============================================+
Proceed to removal [y/n]:
```

```
Type the option you want to choose and press Enter [0-2]:
+---------------------------------+
| Option | Service                |
+=================================+
| 1      | Edit tax value         |
+---------------------------------+
| 2      | Change access password |
+---------------------------------+
| 0      | Go back                |
+---------------------------------+
Type the option you want to choose and press Enter [0-2]:
+--------------------------+
| Option | Service         |
+==========================+
| 1      | Search by User  |
+--------------------------+
| 2      | Search by Date  |
+--------------------------+
| 0      | Go back         |
+--------------------------+
Type the option you want to choose and press Enter [0-2]:
+---------------------------+
| Option | Service          |
+===========================+
| 1      | Drug sales chart |
+---------------------------+
| 2      | Sales Report     |
+---------------------------+
| 0      | Go back          |
+---------------------------+
```

(5) SYSTEM TESTING

Software Testing is an empirical investigation conducted to provide stakeholders with information about the quality of the product, with respect to the context in which it is intended to operate. Software Testing also provides an objective, independent view of the software to allow the management to appreciate and understand the risks during the implementation of the software. The aim of the system testing process was to determine all defects in our project. The program was subjected to a series of trial operations with test inputs and various observations were made and based on these observations, changes were made and again tested for better results. Our Project went through two levels of testing Unit testing and Integration testing.

(5.1) UNIT TESTING:

Unit testing was undertaken when a module has been created and successfully reviewed. In order to test a single module, we need to provide a complete working environment.

(5.2) INTEGRATION TESTING:

After integrating the entire modules developed, we performed various checks by providing different set of test input. The primary objective is to test all the modules in order to ensure that no errors are occurring when one module invokes the other module.

(6) CONCLUSION

The software for Pharmacy is found to be working efficiently. The software appears very flexible since it is menu driven with user-friendly screens. No Formal programming knowledge is required for the user. Also, the user is not burdened with data storing and data retrieval procedures as both are done internally. The user is able to manipulate data easily.

(7) REFERENCES

Learning Computer Science by Sumita Arora

Python Documentation

Python Standard Library

THANK YOU