```
# The School Cafeteria Management System
# Written by M.V.Harish Kumar - Grade 12 'A' on 05-09-2024
import mysql.connector as ms

# ********** Terminal Formatting codes and settings **********
cols = " " * 40

# Colour Codes
clrReset = '\033[00m'
clrBold = '\033[01m'
clrBlink = '\033[05m'
clrClear = '\033[2J\033[H'
clrRed = '\033[31m'
clrGreen = '\033[32m'
clrYellow = '\033[93m'

# ********** Logging Function **********
def log(kind='I', *args, sep=' '):
    kinds = {
            'I': ('INFO', clrBold, ''),
            'E': ('ERROR', clrRed, '\a'),
            'S': ('SUCCESS', clrGreen, ''),
            'W': ('WARNING', clrYellow, '\a')
    }
    k, c, a = kinds[kind]
    print(a, cols, "{}{}:".format(c,k), *args, clrReset, sep=sep)

# ********** Connect to Database **********
try:
    dbConnError = False
    conn = ms.connect(host="localhost", user="cafeAdmin", passwd="cafe@p$wd",
db="cafeteria")
except:
    dbConnError = True
    log('E', "\tServer Error Occured", "\tCan't connect to Database Server",
"\tPlease try again", sep='\n')

# ********** CMDLINE Printing and Formatting Helpers **********
def printBanner(*args, sep=' ', asStr=False):
    pStr = sep.join(args)
    border = cols + "+-{}-+".format('-'*len(pStr)) + "\n"
    if not asStr:
        pStr = clrBlink + clrBold + pStr + clrReset
    banStr = border + cols + "| {} |".format(pStr) + '\n' + border
    if not asStr:
        print(banStr)
    return banStr
```

```
def printTitle(titleStr):
    print(cols + '-' * len(titleStr))
    print(cols + clrBold + titleStr, clrReset)

def genTable(data, header=True, footer=False, colp=True):
    col_widths = [max(len(str(item)) for item in col) for col in zip(*data)]
    border = "+-" + "-+-".join("-" * width for width in col_widths) + "-+"
    table = [border]

    def format_row(row):
        return "| " + " | ".join(f"{str(item).ljust(width)}" for item, width
in zip(row, col_widths)) + " |"

    table.append(format_row(data[0]))
    if header:
        table.append(border.replace("-", "="))
    for row in data[1:-1]:
        table.append(format_row(row))
    if footer:
        table.append(border.replace("-", "="))
    if len(data) != 1:
        table.append(format_row(data[-1]))
    table.append(border)

    return "\n".join([cols+row if colp else row for row in table])

# ********** Functions to sanitize inputs **********
def validateInput(prompt, vals=None, itype=str):
    while True:
        try:
            data = input(prompt)
            if itype != str:
                data = itype("0"+data)
            if vals is None or data in vals:
                return itype(data)
            log('E', "Invalid input for given options. Please try again")
        except ValueError:
            log('E', "Invalid input typed. Please try again")

def inputLOV(prompt, options):
    print("\n", prompt)
    print(genTable([((i+1), options[i]) for i in range(len(options))], False))
    optId = validateInput(cols+clrBold+"ENTER AN OPTION: "+clrReset,
                          range(1, len(options)+1), int)
    return options[optId-1]

def inputDate(prompt):
    print(cols+prompt)
```

```
    d = validateInput(cols+"Enter date[1-31]: ", range(1, 32), int)
    m = validateInput(cols+"Enter month[1-12]: ", range(1, 13), int)
    y = validateInput(cols+"Enter year[YYYY]: ", range(2024, 2025), int)
    return "{}-{}-{}".format(y,m,d)

# ********** Function to generate Reports **********
def printReport(tagline, custData, repData, total=True):
    kind = validateInput("Do you want to print {} on screen[y/n]:
".format(tagline), "yn")
    if kind == "y":
        printBanner("The Velammal Cafeteria - {} Report".format(tagline))
        for k, v in custData.items():
            print(cols,"{}: {}".format(k, v))
        print(genTable(repData, footer=total))
    else:
        log('I', "Printing to file")
        bname = "{}_{}_{}.txt".format(tagline.replace(' ', ''),
custData.get('custCode', ''),
                                      custData['Date'].replace('-', ''))
        with open(bname, 'w') as billF:
            billF.write(printBanner("The Velammal Cafeteria - {}
Report".format(tagline), asStr=True)+'\n')
            for k, v in custData.items():
                billF.write(cols+"{}: {}\n".format(k, v))
            billF.write(genTable(repData, footer=total))
        log('S', "Succesfully printed", tagline, "report to", bname)

# ********** Function to Manage Staff Details **********
def staffMan(dbCur):
    options = ["Add new user", "Edit user details", "View users", "Delete
user"]
    header = [('ID', 'Name', 'Username', 'Password', 'Status')]
    opt = inputLOV("How would you like to manage staffs?", options)

    if opt == "Add new user":
        printTitle("Adding new user")
        name = input(cols+"Enter Staff name: ")
        uname = input(cols+"Enter userid for user: ")
        passwd = input(cols+"Enter password for user: ")
        dbCur.execute("SELECT IFNULL(MAX(id), 0)+1 FROM staff")
        dbCur.execute("INSERT INTO staff VALUES ({}, '{}', '{}', '{}', 'A')"\
                      .format(dbCur.fetchone()[0], name, uname, passwd))
        conn.commit()
        log('S', "Added new user sucessfully!")

    elif opt == "Edit user details":
        printTitle("Updating User")
        log('I', "Leaving a field empty will retain the previous value")
```

```
        dbCur.execute("SELECT * FROM staff")
        table = dbCur.fetchall()
        print(genTable(header + table))
        uid = validateInput(cols+"Enter the user id to edit: ", [x[0] for x in
table], int)
        for rec in table:
            if rec[0] == uid:
                data = rec
                break
        name = input(cols+"Enter Staff name[Default: {}]: ".format(data[1]))
        if not name:
            name = data[1]
        uname = input(cols+"Enter userid[Default: {}]: ".format(data[2]))
        if not uname:
            uname = data[2]
        passwd = input(cols+"Enter password[Default: {}]: ".format(data[3]))
        if not passwd:
            passwd = data[3]
        sts = validateInput(cols+"Enter Status[(A)ctive/(I)nactive][Default:
{}]: ".format(data[4]), "AI")
        if not sts:
            sts = data[4]
        dbCur.execute("UPDATE staff SET
name='{}',userid='{}',passwd='{}',status='{}'\
                      WHERE id = {}".format(name, uname, passwd, sts, uid))
        conn.commit()
        log('S', "Updated user sucessfully!")

    elif opt == "View users":
        opt = inputLOV("How would you like to view users?", ["All", "Search"])
        if opt == "All":
            _flagAll = True
            query = "SELECT * FROM staff"
        elif opt == "Search":
            _flagAll = False
            uid = validateInput(cols+"Enter the user id: ", None, int)
            query = "SELECT * FROM staff WHERE id = {}".format(uid)
        dbCur.execute(query)
        data = dbCur.fetchall()
        if data == [] and not _flagAll:
            log('I', "No such user with id {} found".format(uid))
        else:
            print(genTable(header + data))

    elif opt == "Delete user":
        printTitle("Deleting user")
        dbCur.execute("SELECT * FROM staff")
        data = dbCur.fetchall()
```

```
        print(genTable(header + data))
        uid = validateInput(cols+"Enter the user id to delete: ", [x[0] for x
in data], int)
        dbCur.execute("DELETE FROM staff WHERE id = {}".format(uid))
        conn.commit()
        log('S', "Deleted user sucessfully!")

# ********** Function to Manage Customer Details **********
def custMan(dbCur):
    options = ["Add new customer", "Edit customer details", "View customers",
"Delete customer"]
    header = [('custCode', 'Name', 'Type', 'Status')]
    kindSwitch = {'S': 'Student', 'T': 'Staff'}
    opt = inputLOV("How would you like to manage customers?", options)

    if opt == "Add new customer":
        printTitle("Adding new customer")
        name = input(cols+"Enter name of the customer: ")
        ckind = validateInput(cols+"Enter customer kind[(S)tudent/s(T)aff]: ",
"ST")
        dbCur.execute("SELECT IFNULL(MAX(custCode),0)+1 FROM customer")
        dbCur.execute("INSERT INTO customer VALUES ({}, '{}', '{}', 'A')"\
                      .format(dbCur.fetchone()[0], name, kindSwitch[ckind]))
        conn.commit()
        log('S', "Added new customer sucessfully!")

    elif opt == "Edit customer details":
        printTitle("Updating customer")
        log('I', "Leaving a field empty will retain the previous value")
        dbCur.execute("SELECT * FROM customer")
        table = dbCur.fetchall()
        print(genTable(header + table))
        cid = validateInput(cols+"Enter the custCode to edit: ", [x[0] for x
in table], int)
        for rec in table:
            if rec[0] == cid:
                data = rec
                break
        name = input(cols+"Enter name[Default: {}]: ".format(data[1]))
        if not name:
            name = data[1]
        ckind = validateInput(cols+"Enter customer
kind[(S)tudent/s(T)aff][Default: {}]: "\
                              .format(data[2]), "TS")
        ckind = kindSwitch.get(ckind, data[2])
        sts = validateInput(cols+"Enter Status[(A)ctive/(I)nactive][Default:
{}]: ".format(data[3]), "AI")
        if not sts:
```

```
            sts = data[3]
        dbCur.execute("UPDATE customer SET
name='{}',custType='{}',status='{}'\
                      WHERE custCode = {}".format(name, ckind, sts, cid))
        conn.commit()
        log('S', "Updated customer sucessfully!")

    elif opt == "View customers":
        opt = inputLOV("How would you like to view customers?", ["All",
"Search"])
        if opt == "All":
            _flagAll = True
            query = "SELECT * FROM customer"
        elif opt == "Search":
            _flagAll = False
            cid = validateInput(cols+"Enter the custCode: ", None, int)
            query = "SELECT * FROM customer WHERE custCode = {}".format(cid)
        dbCur.execute(query)
        data = dbCur.fetchall()
        if data == [] and not _flagAll:
            log('I', "No such customer with code {} found".format(cid))
        else:
            print(genTable(header + data))

    elif opt == "Delete customer":
        printTitle("Deleting customer")
        dbCur.execute("SELECT * FROM customer")
        data = dbCur.fetchall()
        print(genTable(header + data))
        cid = validateInput("Enter the custCode to delete: ", [x[0] for x in
data], int)
        dbCur.execute("DELETE FROM customer WHERE custCode = {}".format(cid))
        conn.commit()
        log('S', "Deleted user sucessfully!")

# ********** Function to Manage Menu items **********
def menuMan(dbCur):
    options = ["Add item", "View items", "Update rate", "Delete item"]
    header = [("itemCode", "Item Name", "Rate")]
    opt = inputLOV("Choose an operation", options)

    if opt == "Add item":
        printTitle("Adding new menu Item")
        name = input(cols+"Enter name of menu item: ")
        rate = validateInput(cols+"Enter rate: ", None, float)
        dbCur.execute("SELECT IFNULL(MAX(itemCode),0)+1 FROM items")
        dbCur.execute("INSERT INTO items VALUES ({}, '{}',
{})".format(dbCur.fetchone()[0], name, rate))
```

```
        conn.commit()
        log('S', "Added new menu item successfully!")

    elif opt == "View items":
        dbCur.execute("SELECT * FROM items")
        print(genTable(header + dbCur.fetchall()))

    elif opt == "Update rate":
        printTitle("Updating rate")
        dbCur.execute("SELECT * FROM items")
        table = dbCur.fetchall()
        print(genTable(header + table))
        icd = validateInput(cols+"Enter the itemCode to edit: ", [x[0] for x
in table], int)
        rt = validateInput(cols+"Enter new rate: ", None, float)
        dbCur.execute("UPDATE items SET rate = {} where itemCode =
{}".format(rt,icd))
        conn.commit()
        log('S', "Updated rate sucessfully!")

    elif opt == "Delete item":
        printTitle("Deleting menu item")
        dbCur.execute("SELECT * FROM items")
        data = dbCur.fetchall()
        print(genTable(header + data))
        icd = validateInput(cols+"Enter the itemCode to delete: ", [x[0] for x
in data], int)
        dbCur.execute("DELETE FROM items WHERE itemCode = {}".format(icd))
        conn.commit()
        log('S', "Deleted item sucessfully!")

# ********** Function to input Receipt Data **********
def getReceiptData(dataTable):
    data = []
    opt = "y"
    while opt.lower() == 'y':
        print(genTable(dataTable))
        icd = validateInput(cols+"Enter itemCode: ", [x[0] for x in
dataTable[1:]], int)
        qty = validateInput(cols+"Enter quantity: ", None, int)
        data.append((icd, qty))
        opt = validateInput("Do you want to continue[y/n]: ", "yn")
    return data

# ********** Function to Manage Daily Stock items **********
def stockMan(dbCur):
    options = ["Add receipt", "View receipt", "Update quantity", "Delete
item"]
```

```
    header = [("itemCode", "Item Name", "Quantity")]
    opt = inputLOV("Choose an operation", options)

    if opt == "Add receipt":
        printTitle("Adding new receipt")
        dbCur.execute("SELECT * FROM items")
        dbCur.executemany("INSERT INTO dailyStock VALUES (%s, current_date(),
%s)",
                          getReceiptData(header + dbCur.fetchall()))
        conn.commit()
        log('S', "Added receipt successfully!")

    elif opt == "View receipt":
        dbCur.execute("SELECT i.itemCode, i.itemName, s.quantity FROM
dailyStock s, items i\
                      WHERE i.itemCode = s.itemCode AND s.receiptDate =
current_date();")
        print(genTable(header + dbCur.fetchall()))

    elif opt == "Update quantity":
        printTitle("Updating quantity")
        dbCur.execute("SELECT i.itemCode, i.itemName, s.quantity FROM
dailyStock s, items i\
                      WHERE i.itemCode = s.itemCode AND s.receiptDate =
current_date();")
        table = dbCur.fetchall()
        print(genTable(header + table))
        icd = validateInput(cols+"Enter the itemCode to update: ", [x[0] for x
in table], int)
        qty = validateInput(cols+"Enter new quantity: ", None, int)
        dbCur.execute("UPDATE dailyStock SET quantity = {} WHERE itemCode =
{}\
                      AND receiptDate = current_date()".format(qty,icd))
        conn.commit()
        log('S', "Updated quantity sucessfully!")

    elif opt == "Delete item":
        printTitle("Deleting receipt item")
        dbCur.execute("SELECT i.itemCode, i.itemName, s.quantity FROM
dailyStock s, items i\
                      WHERE i.itemCode = s.itemCode AND s.receiptDate =
current_date();")
        table = dbCur.fetchall()
        print(genTable(header + table))
        icd = validateInput(cols+"Enter the itemCode to delete: ", [x[0] for x
in table], int)
        dbCur.execute("DELETE FROM dailyStock WHERE receiptDate =
current_date()\
```

```
                        AND itemCode = {}".format(icd))
        conn.commit()
        log('S', "Deleted item sucessfully!")

def addBill(dbCur, tokId, query):
    log('I', "Current Token id is", tokId)
    opt = "y"
    while opt.lower() == 'y':
        itemCodes = []
        dbCur.execute("SELECT i.itemCode, i.itemName, s.quantity FROM
dailyStock s, items i\
                        WHERE i.itemCode = s.itemCode AND s.receiptDate =
current_date();")
        header = [("itemCode", "itemName", "quantity")]
        table = dbCur.fetchall()
        print(genTable(header + table))
        icd = validateInput(cols+"Enter itemCode: ", [x[0] for x in table],
int)
        while True:
            qty = validateInput(cols+"Enter quantity: ", None, int)
            for rec in table:
                if rec[0] == icd:
                    threshold = rec[2]
                    break
            if qty <= threshold:
                dbCur.execute("UPDATE dailyStock SET quantity = quantity - {}
WHERE itemCode = {}\
                                AND receiptDate = current_date()".format(qty,
icd))
                if icd in itemCodes:
                    dbCur.execute("UPDATE sales SET qty = qty + {} WHERE tDate
= current_date()\
                                    AND tokenId = {} AND itemCode =
{}".format(tokId, icd))
                else:
                    itemCodes.append(icd)
                    dbCur.execute(query % (icd, qty))
                break
            log('E', "Only", threshold, "unit(s) is available, enter another
value")
        opt = validateInput("Do you want to continue[y/n]: ", "yn")

# ********** Function to Manage Daily Sales **********
def salesMan(dbCur):
    options = ["Add bill", "Update bill", "Delete bill"]
    itemHdr = [("itemCode", "Item Name", "Quantity")]
    custHdr = [("custCode", "Name", "Type")]
    billTokHdr = [('tokenId', 'Customer Name')]
```

```
    opt = inputLOV("Choose an operation", options)

    if opt == "Add bill":
        printTitle("Adding new bill")
        custData = {}
        dbCur.execute("SELECT * FROM customer")
        table = dbCur.fetchall()
        print(genTable(custHdr + table))
        custData['custCode'] = validateInput(cols+"Enter the custCode: ",
[x[0] for x in table], int)
        for rec in table:
            if rec[0] == custData['custCode']:
                custData['Name'] = rec[1]
        dbCur.execute("SELECT IFNULL(MAX(tokenId),0)+1 FROM sales WHERE tDate
= current_date()")
        custData['Token ID'] = dbCur.fetchone()[0]
        query = "INSERT INTO sales VALUES({}, current_date(), {}, %s, %s)" \
                .format(custData['Token ID'], custData['custCode'])
        addBill(dbCur, custData['Token ID'], query)
        conn.commit()
        log('S', "Added bill successfully!")

        pBill = validateInput("Do you want to print bill[y/n]: ", "yn")
        if pBill == 'y':
            dbCur.execute("SELECT current_date()")
            custData['Date'] = str(dbCur.fetchone()[0])
            dbCur.execute("SELECT i.itemName, s.qty, s.qty*i.rate FROM items
i, sales s\
                           WHERE i.itemCode = s.itemCode AND s.custCode = {}\
                           AND s.tokenId = {} AND s.tDate = current_date()"\
                           .format(custData['custCode'], custData['Token ID']))
            billData, total = dbCur.fetchall(), 0
            for sno in range(len(billData)):
                total += billData[sno][2]
                billData[sno] = (sno+1,) + billData[sno]
            billData = [('SNO', 'Item', 'Quantity', 'Price')] + billData +
[('', '', 'Total', total)]
            printReport('Bill', custData, billData)

    elif opt == "Update bill":
        printTitle("Updating bill")
        dbCur.execute("SELECT DISTINCT s.tokenId, c.name FROM sales s,
customer c\
                      WHERE s.custCode = c.custCode AND s.tDate =
current_date()")
        table = dbCur.fetchall()
        print(genTable(billTokHdr + table))
```

```
        tid = validateInput(cols+"Enter the tokenId to update: ", [x[0] for x
in table], int)
        dbCur.execute("SELECT s.itemcode, i.itemName, s.qty FROM items i,
sales s\
                      WHERE i.itemCode = s.itemCode AND s.tokenId = {}\
                      AND s.tDate = current_date()".format(tid))
        billData = dbCur.fetchall()
        print(genTable(itemHdr + billData))
        icd = validateInput(cols+"Enter the itemCode to update: ", [x[0] for x
in billData], int)
        newQty = validateInput(cols+"Enter the Decrease in quantity: ", None,
int)
        for rec in billData:
            if rec[0] == icd:
                prevQty = rec[2]
                break
        if newQty <= prevQty:
            dbCur.execute("UPDATE dailyStock SET quantity = quantity + {}
WHERE itemCode = {}\
                          AND receiptDate = current_date()".format(newQty,
icd))
            if newQty == prevQty:
                dbCur.execute("DELETE FROM sales WHERE itemCode = {} AND tDate
= current_date()\
                              AND tokenId = {}".format(icd, tid))
            else:
                dbCur.execute("UPDATE sales SET qty = qty - {} WHERE itemCode
= {}\
                              AND tDate = current_date() AND tokenId =
{}".format(newQty, icd, tid))
        else:
            log('W', 'Only decrease in quantity is supported. Add a new bill
to increase quantity')
        conn.commit()
        log('S', "Updated Bill Successfully")

    elif opt == "Delete bill":
        printTitle("Deleting bill")
        dbCur.execute("SELECT DISTINCT s.tokenId, c.name FROM sales s,
customer c\
                      WHERE s.custCode = c.custCode AND s.tDate =
current_date()")
        data = dbCur.fetchall()
        print(genTable(billTokHdr + data))
        tid = validateInput(cols+"Enter the tokenId of bill to delete: ",
[x[0] for x in data], int)
        dbCur.execute("UPDATE sales s, dailyStock d SET d.quantity =
d.quantity + s.qty WHERE\
```

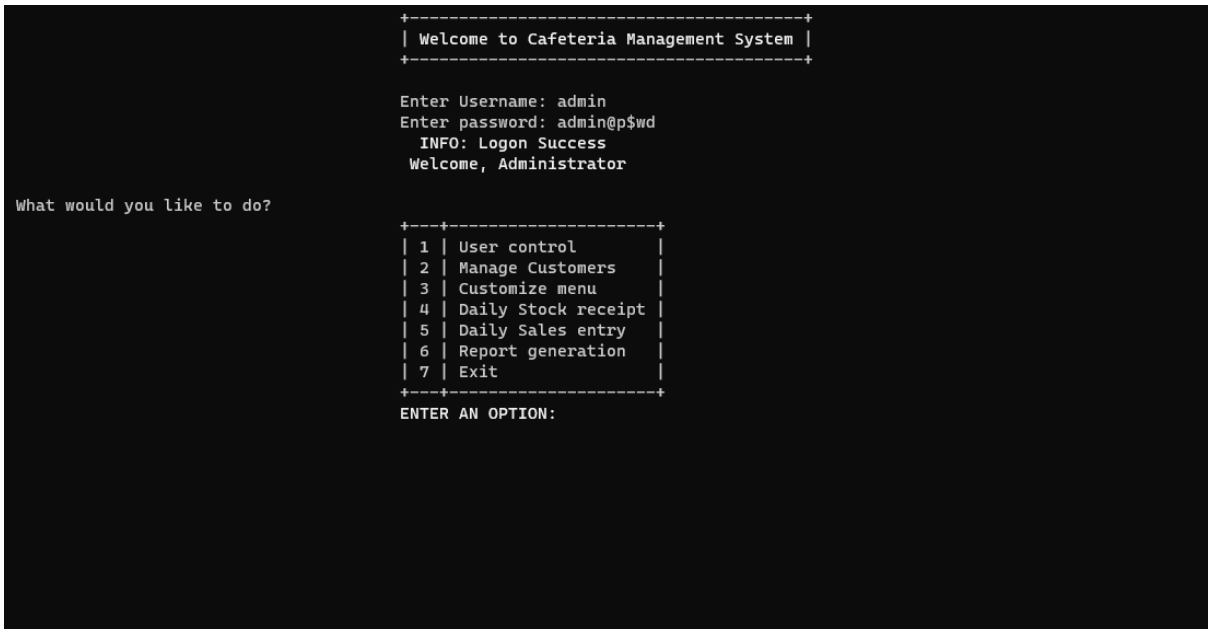
```
                        s.itemCode = d.itemCode AND s.tokenId = {} AND s.tDate =
current_date()\
                        AND d.receiptDate = current_date()".format(tid))
        dbCur.execute("DELETE FROM sales WHERE tDate = current_date() AND
tokenId = {}".format(tid))
        conn.commit()
        log('S', "Deleted bill sucessfully!")

# ********** Function to Manage Reports **********
def repMan(dbCur):
    options = ["Bill History", "Sales Reconcilation", "Stock Receipt"]
    opt = inputLOV("Enter your choice: ", options)
    dt = inputDate("Date of History")

    if opt == "Bill History":
        dbCur.execute("SELECT DISTINCT s.tokenId, c.custCode, c.name FROM
sales s, customer c\
                        WHERE tDate = '{}' AND c.custCode =
s.custCode".format(dt))
        data = dbCur.fetchall()
        if data == []:
            log('E', 'No records found on date:', dt)
        else:
            custData = {'Date': dt}
            print(genTable([('Token ID', 'custCode', 'Customer Name')] +
data))
            custData['Token ID'] = validateInput("Enter token id: ", [x[0] for
x in data], int)
            for row in data:
                if row[0] == custData['Token ID']:
                    custData['custCode'] = row[1]
                    custData['Customer Name'] = row[2]
            dbCur.execute("SELECT i.itemName, s.qty, s.qty*i.rate\
                            FROM items i, sales s WHERE i.itemCode = s.itemCode
AND\
                            s.custCode = {} AND s.tokenId = {} AND s.tDate =
'{}'"\
                            .format(custData['custCode'], custData['Token ID'],
dt))
            billData, total = dbCur.fetchall(), 0
            for sno in range(len(billData)):
                total += billData[sno][2]
                billData[sno] = (sno+1,) + billData[sno]
            billData = [('SNO', 'Item', 'Quantity', 'Price')] + billData +
[('', '', 'Total', total)]
            printReport('Bill', custData, billData, total)

    elif opt == "Sales Reconcilation":
```

```
        dbCur.execute("SELECT s.itemCode, i.itemName, d.quantity+SUM(s.qty),
SUM(s.qty),\
                      d.quantity FROM sales s, items i, dailyStock d\
                      WHERE s.itemCode = i.itemCode AND d.itemCode =
s.itemCode\
                      AND s.tDate = '{0}' AND d.receiptDate = '{0}'\
                      GROUP BY s.itemCode, d.itemCode, d.quantity".format(dt))
        salesData = dbCur.fetchall()
        if salesData == []:
            log('E', 'No records found on date:', dt)
        else:
            for sno in range(len(salesData)):
                salesData[sno] = (sno+1,) + salesData[sno]
            salesData = [('SNO', 'itemCode', 'Name', 'Quantity imported',
                          'Quantity sold', 'Remaining')] + salesData
            printReport('Sales Reconcilation', {'Date': dt}, salesData,
total=False)

    elif opt == "Stock Receipt":
        dbCur.execute("SELECT i.itemCode, i.itemName, s.quantity FROM
dailyStock s, items i \
                       WHERE i.itemCode = s.itemCode AND s.receiptDate =
'{}'".format(dt))
        receiptData = dbCur.fetchall()
        if receiptData == []:
            log('E', 'No records found on date:', dt)
        else:
            total = 0
            for sno in range(len(receiptData)):
                total += receiptData[sno][2]
                receiptData[sno] = (sno+1,) + receiptData[sno]
            receiptData = [('SNO', 'itemCode', 'Name', 'Quantity')] +
receiptData
            printReport('Stock Receipt', {'Date': dt}, receiptData,
total=False)

# ********** Main entry function for menu **********
def mainMenu(dbCur, userData):
    while True:
        options = ["User control", "Manage Customers", "Customize menu",
"Daily Stock receipt",
                   "Daily Sales entry", "Report generation", "Exit"]
        opt = inputLOV("What would you like to do?", options)
        if opt == "User control":
            if not userData['isAdmin']:
                log('E', "User", userData['name'], "doesn't have rights to
manage users!")
            else:
```

```
                staffMan(dbCur)
            elif opt == "Manage Customers":
                if not userData['isAdmin']:
                    log('E', "User", userData['name'], "doesn't have rights to
manage customers!")
                else:
                    custMan(dbCur)
            elif opt == "Customize menu":
                menuMan(dbCur)
            elif opt == "Daily Stock receipt":
                stockMan(dbCur)
            elif opt == "Daily Sales entry":
                salesMan(dbCur)
            elif opt == "Report generation":
                repMan(dbCur)
            elif opt == "Exit":
                print(cols, f"Logging out user: {userData['name']}...")
                break

        log('S', "Successfully logged out!")
        print(cols, "Thank you")

if not dbConnError and conn.is_connected():
        cur = conn.cursor()

        print(clrClear, end="")
        printBanner("Welcome to Cafeteria Management System")
        uname = input(cols + "Enter Username: ")
        pswd = input(cols + "Enter password: ")
        cur.execute(f"SELECT name, passwd FROM staff WHERE userId = '{uname}' AND
status = 'A'")
        data = cur.fetchall()
        if data != []:
            if data[0][1] == pswd:
                userData = {"name": data[0][0], "username": uname, "isAdmin":
data[0][0] == "Administrator"}
                log('I', 'Logon Success')
                print(cols, clrBold, f"\bWelcome, {userData['name']}", clrReset)
                try:
                    mainMenu(cur, userData)
                except KeyboardInterrupt:
                    log('W', "User Interrupted Operation. Exiting...")
                except Exception as e:
                    log('E', "FATAL ERROR OCCURED")
                    log('E', "The error message was:")
                    #raise e
                    log('E', str(e))
            else:
```

```
            log('E', "Invalid password for user:", uname)
    else:
        log('E', "No such user in database:", uname)

input("Press enter to continue...")
if not dbConnError:
    conn.commit()
    conn.close()
```

OUTPUT:



```
| 5 | Daily Sales entry   |
| 6 | Report generation   |
| 7 | Exit                |
+---+---------------------+
ENTER AN OPTION: 3

Choose an operation

+---+-------------+
| 1 | Add item    |
| 2 | View items  |
| 3 | Update rate |
| 4 | Delete item |
+---+-------------+
ENTER AN OPTION: 2
+----------+----------------+-------+
| itemCode | Item Name      | Rate  |
+==========+================+=======+
| 1        | Coffee         | 15.00 |
| 2        | Tea            | 10.00 |
| 3        | Mineral Water  | 10.00 |
| 4        | Buttermilk     | 20.00 |
| 5        | Grape juice    | 20.00 |
| 6        | Veg pulao      | 50.00 |
| 7        | Veg briyani    | 50.00 |
| 8        | Meals          | 40.00 |
| 9        | Sarbath        | 20.00 |
| 10       | Veg puffs      | 20.00 |
| 11       | Egg puffs      | 30.00 |
| 12       | Veg Sandwich   | 35.00 |
| 13       | Vadai          | 15.00 |
| 14       | Mini samosa    | 20.00 |
| 15       | Veg roll       | 15.00 |
| 16       | Chicken roll   | 20.00 |
| 17       | Veg Burger     | 50.00 |
| 18       | Donut          | 35.00 |
```

```
| 11       | Egg puffs      | 30.00 |
| 12       | Veg Sandwich   | 35.00 |
| 13       | Vadai          | 15.00 |
| 14       | Mini samosa    | 20.00 |
| 15       | Veg roll       | 15.00 |
| 16       | Chicken roll   | 20.00 |
| 17       | Veg Burger     | 50.00 |
| 18       | Donut          | 35.00 |
| 19       | Bread omelette | 55.00 |
| 20       | Sambar rice    | 50.00 |
| 21       | Corn puffs     | 20.00 |
| 22       | Cream bun      | 20.00 |
| 23       | Chips          | 20.00 |
| 24       | Milkshake      | 35.00 |
| 25       | Skittles       | 10.00 |
| 26       | Kitkat         | 25.00 |
| 27       | Munch (large)  | 20.00 |
+----------+----------------+-------+

What would you like to do?

+---+--------------------+
| 1 | User control       |
| 2 | Manage Customers   |
| 3 | Customize menu     |
| 4 | Daily Stock receipt |
| 5 | Daily Sales entry  |
| 6 | Report generation  |
| 7 | Exit               |
+---+--------------------+
ENTER AN OPTION: 7
 Logging out user: Administrator...
  SUCCESS: Successfully logged out!
 Thank you
Press enter to continue...
```