PYTHON FULL STACK COURSE MINIPROJECT

**STATIONARY SALES MANGEMENT SYSTEM**

RAJA M



Project Report submitted in Partial fulfilment requirement for the Proficient

Certificate Course

Done By

# RAJA M

Under the Guidance of Approved By

**VIDHYA SAKTHIVEL**



**About the Project**

This Stock Management System is a desktop application built using Python’s Tkinter library and MySQL as the backend database. The application opens with a **welcome page** that features a **login button**. Users can log in using their **Gmail and password**, which are validated against a user database. If the user does not have an account, they can register using the **Register Page**, which collects the same login details (Gmail and password) and stores them in the database. Once authenticated, users are redirected to the main stock management interface with tabs for stock management, memo creation, and reports.

**Project Prerequisites**

* **Tkinter**: Tkinter is Python's standard GUI library used to build the entire front-end interface of the project including tabs, buttons, labels, and input fields. It comes pre-installed with Python.
* **MySQL**: MySQL is the relational database used to store user details, stock items, sales data, and generated memos. You need to install MySQL Server and mysql-connector-python to connect it with the application.
* Install the required module using:

pip install mysql-connector-python

* Ensure that the MySQL server is running and a database is created with necessary tables before running the application.

**Tkinter Classes Used**

| **Class** | **Syntax** | **Description & Usage in Project** |
| --- | --- | --- |
| Tk() | root = Tk() | Main application window. |
| Toplevel() | win = Toplevel(root) | Creates pop-up windows for login and registration. |
| Label | Label(root, text="Email") | Displays text labels beside input fields. |
| Entry | Entry(root) | Text input fields used in forms (login, register, stock). |
| Button | Button(root, text="Login") | Triggers actions like login, registration, stock addition. |
| ttk.Notebook | tabs = ttk.Notebook(root) | Tabbed interface for Stocks, Memo, Report, and Settings. |
| ttk.Treeview | Treeview(root, columns=[...]) | Displays tabular data for stock and memos. |
| ttk.Combobox | Combobox(root, values=[...]) | Dropdown for selecting stock items in memo tab. |
| ttk.Scrollbar | Scrollbar(tree) | Adds scrolling support to table views. |
| ttk.Style() | style = ttk.Style() | Applied custom styles (e.g., red text for low stock). |
| LabelFrame | LabelFrame(root, text="Stock Info") | Grouped frame for form sections. |
| messagebox | messagebox.showinfo("Success", ...) | Shows alert/confirmation messages for validation and success actions. |

**Source Code**

**Welcome.py:**

from tkinter import \*

from PIL import Image, ImageTk

from project\_config import \*

from login import \*

w = Tk()

w.geometry('700x500')

w.title('Welcome Page')

login\_background\_image = Image.open('stationary.jpg').resize((700, 500))

login\_background\_image\_tk = ImageTk.PhotoImage(login\_background\_image)

label\_image = Label(w, image=login\_background\_image\_tk)

label\_image.image = login\_background\_image\_tk

label\_image.place(x=0, y=0)

welcome\_note = Label(w, text='''

Welcome to the

Zink stationary shop''',

                     font=CustomFontStyle(font\_size=25),

                     bg='white')

welcome\_note.place(x=30, y=190)

def show\_login\_page():

    w.withdraw()

    Login\_Page()

login\_btn = Button(w, text='Click to login',

                   bg=background\_color,

                   activebackground='white',

                   activeforeground='yellow',

                   command=show\_login\_page,

                   font=CustomFontStyle(font\_size=15))

login\_btn.place(x=100, y=350)

w.mainloop()

**Login.py:**

from tkinter import \*

from PIL import Image, ImageTk

from project\_config import \*

from register import Register\_Page

from main import \*

from model\_connection import \*

from tkinter import messagebox

def Login\_Page():

    w = Toplevel()

    w.geometry('700x500')

    w.title('Login Page')

    login\_background\_image = Image.open('login\_bg.jpg').resize((700, 500))

    login\_background\_image\_tk = ImageTk.PhotoImage(login\_background\_image)

    w.bg\_image = login\_background\_image\_tk

    label\_image = Label(w, image=login\_background\_image\_tk)

    label\_image.place(x=0, y=0)

    heading\_label = Label(w, text='Login',

                        font=CustomFontStyle(font\_size=30,

                                             weight='bold'),

                        background='white')

    heading\_label.place(x=200, y=100)

    gmail\_label = Label(w, text='Gmail:',

                        font=CustomFontStyle(font\_size=15),

                        background='white')

    gmail\_label.place(x=200, y=200)

    g\_entry = Entry(w,bg='white', font=CustomFontStyle(font\_size=15)

                    )

    g\_entry.place(x=350, y=200)

    password\_label = Label(w, text='Password:',

                        font=CustomFontStyle(font\_size=15),

                        background='white')

    password\_label.place(x=200, y=250)

    pass\_entry = Entry(w,bg='white',font=CustomFontStyle(font\_size=15))

    pass\_entry.place(x=350, y=250)

    def show\_main\_page():

        result = run\_query(f'select \* from users where gmail="{g\_entry.get()}" and password="{pass\_entry.get()}"')

        is\_user\_exist = result.\_\_len\_\_()>=1

        logged\_in\_user = None

        if is\_user\_exist:

            w.withdraw()

            logged\_in\_user = result[0]

            stock\_app(logged\_in\_user)

        else:

             messagebox.showerror("Error", "User doesn't exist please register (or) check the entered gmail and password is correct")

        return

    login\_btn = Button(w, text='Login',

                   bg=background\_color,

                   activebackground='white',

                   activeforeground='yellow',

                   command=show\_main\_page,

                   font=CustomFontStyle(font\_size=15))

    login\_btn.place(x=350, y=300)

    Or\_label = Label(w,text='OR', font=CustomFontStyle(

        font\_size=15, weight='bold'

    ),

    background='white')

    Or\_label.place(x=350, y=350)

    def show\_register():

        Register\_Page()

    register\_btn = Button(w, text='Register',

                   bg=background\_color,

                   activebackground='white',

                   activeforeground='yellow',

                   command=show\_register,

                   font=CustomFontStyle(font\_size=15))

    register\_btn.place(x=350, y=380)

    w.mainloop()

**Register.py:**

from tkinter import \*

from tkinter import messagebox

from PIL import Image, ImageTk

from project\_config import \*

from main import \*

from model\_connection import \*

def Register\_Page():

    w = Toplevel()

    w.geometry('700x500')

    w.title('Register Page')

    login\_background\_image = Image.open('login\_bg.jpg').resize((700, 500))

    login\_background\_image\_tk = ImageTk.PhotoImage(login\_background\_image)

    w.bg\_image = login\_background\_image\_tk  # Keep reference

    bg\_label = Label(w, image=login\_background\_image\_tk)

    bg\_label.place(x=0, y=0)

    form\_frame = Frame(w, bg="white")

    form\_frame.place(x=50, y=100)

    heading\_label = Label(form\_frame, text='Register',

                          font=CustomFontStyle(font\_size=30, weight='bold'),

                          bg='white')

    heading\_label.grid(row=0, column=0, columnspan=2, pady=20)

    gmail\_label = Label(form\_frame, text='Gmail:',

                        font=CustomFontStyle(font\_size=15),

                        bg='white')

    gmail\_label.grid(row=1, column=0, sticky='e', padx=10, pady=10)

    g\_entry = Entry(form\_frame, bg='white', font=CustomFontStyle(font\_size=15))

    g\_entry.grid(row=1, column=1, padx=10, pady=10)

    password\_label = Label(form\_frame, text='Password:',

                           font=CustomFontStyle(font\_size=15),

                           bg='white')

    password\_label.grid(row=2, column=0, sticky='e', padx=10, pady=10)

    pass\_entry = Entry(form\_frame, bg='white', font=CustomFontStyle(font\_size=15), show='\*')

    pass\_entry.grid(row=2, column=1, padx=10, pady=10)

    re\_pass\_label = Label(form\_frame, text='Re-enter Password:',

                          font=CustomFontStyle(font\_size=15),

                          bg='white')

    re\_pass\_label.grid(row=3, column=0, sticky='e', padx=10, pady=10)

    re\_pass\_entry = Entry(form\_frame, bg='white', font=CustomFontStyle(font\_size=15), show='\*')

    re\_pass\_entry.grid(row=3, column=1, padx=10, pady=10)

    def validate\_and\_submit():

        if pass\_entry.get() != re\_pass\_entry.get():

            messagebox.showerror("Error", "Passwords do not match!")

        else:

            gmail = g\_entry.get()

            password = pass\_entry.get()

            print(gmail, password)

            insert\_user\_to\_db = 'Insert Into Users(gmail, password) Values(%s,%s)'

            run\_query(insert\_user\_to\_db,(gmail, password))

            w.withdraw()

            messagebox.showinfo("Success", "Registration successful! Please login with that credential")

    login\_btn = Button(form\_frame, text='Sign up',

                       bg=background\_color,

                       activebackground='white',

                       activeforeground='yellow',

                       font=CustomFontStyle(font\_size=15),

                       command=validate\_and\_submit)

    login\_btn.grid(row=4, column=0, columnspan=2, pady=20)

    w.mainloop()

**Model\_connection.py:**

import mysql.connector

import sys

try:

    global con

    con=mysql.connector.connect(host='localhost',

                            user='root',

                            password='',

                            database='sales\_management\_db')

except:

    print('>>>>>>>>>>>',sys.exc\_info()[0],'>>>>>>>>>>>')

def run\_query(query='select \* from users',data\_to\_insert:tuple=None, update\_to\_db=False, reterival\_id=None,

              reterver\_table\_name='users'):

    table = con.cursor()

    result\_set = None

    updated\_user\_details = None

    print(data\_to\_insert)

    if data\_to\_insert != None:

        table.execute(query,data\_to\_insert)

        con.commit()

    elif update\_to\_db:

        table.execute(query)

        con.commit()

        if reterival\_id:

            table.execute(f'select \* from {reterver\_table\_name} where id={reterival\_id}')

            updated\_user\_details = table.fetchall()

        return updated\_user\_details

    else:

        table.execute(query)

        result\_set = table.fetchall()

    return result\_set

**Project\_config.py:**

from tkinter.font import Font

def CustomFontStyle(font\_family="Helvetica", font\_size=25,

                    weight='normal'):

    font\_style = Font(family=font\_family, size=font\_size,

                      weight=weight)

    return font\_style

background\_color = '#cff6f6'

**Main.py:**

import tkinter as tk

from tkinter import ttk, messagebox

from model\_connection import \*

from project\_config import \*

import uuid

from datetime import datetime, date

from PIL import Image, ImageTk

from reports.year\_report import generate\_year\_sales\_report

from reports.month\_report import generate\_monthly\_sales\_report

from weekly\_sales\_report import generate\_weekly\_sales\_report

#2025-02-23

#2025-04-10

stock\_data = []

def get\_updated\_stock():

    stock\_data.clear()  # Clear existing data to avoid duplicates

    result = run\_query('select \* from stock')

    if result:

        for i in result:

            stock\_data.append(i)

def stock\_app(logged\_in\_user):

    logged\_in\_user\_details = logged\_in\_user

    root = tk.Toplevel()

    root.title("Zink Stationary Stock Management System")

    root.geometry("1100x700")

    root.configure(bg='#e6f2ff')

    style = ttk.Style()

    style.theme\_use('clam')

    style.configure("Treeview", background="white", foreground="black",

                    rowheight=25, fieldbackground="white", font=('Helvetica', 10))

    style.configure("Treeview.Heading",

                background="#007acc",  # Blue background

                foreground="white",    # White text

                font=('Helvetica', 10, 'bold'))

    style.map("Treeview.Heading",

          relief=[('active', 'groove'), ('pressed', 'sunken')])

    style.configure("TNotebook.Tab",

                    background="#cce6ff",

                    foreground="black",

                    font=('Helvetica', 10, 'bold'),

                    padding=[10, 5])

    style.map("TNotebook.Tab",

              background=[("selected", "#007acc")],

              foreground=[("selected", "white")])

    header\_frame = tk.Frame(root, bg='#4da6ff')

    header\_frame.pack(fill='x')

    title\_label = tk.Label(header\_frame, text="Zink Stationary Stock Management System", font=CustomFontStyle(font\_size=15, weight='bold'),

                           bg='#4da6ff', fg='white')

    title\_label.pack(side='left', padx=20, pady=10)

    logged\_in\_user\_name = logged\_in\_user\_details[1].split('@')[0]

    user\_label = tk.Label(header\_frame, text=f"Logged in as: {logged\_in\_user\_name}", font=CustomFontStyle(font\_size=12),

                          bg='#4da6ff', fg='white')

    user\_label.pack(side='right', padx=20)

    notebook = ttk.Notebook(root)

    notebook.pack(expand=True, fill='both', padx=10, pady=10)

    # ======== Stocks Tab ========

    stocks\_tab = tk.Frame(notebook, bg='#e6f2ff')

    notebook.add(stocks\_tab, text='Stocks')

    tk.Label(stocks\_tab, text="Available Stocks", bg='#e6f2ff',

             font=CustomFontStyle(font\_size=16, weight='bold')).pack(pady=10)

    stock\_tree = ttk.Treeview(stocks\_tab, columns=("Item Code","Item", "Quantity", "Price", "Edit", "Delete"), show='headings', height=10)

    for col in ("Item Code","Item", "Quantity", "Price", "Edit", "Delete"):

        stock\_tree.heading(col, text=col)

        stock\_tree.column(col, width=100 if col not in ["Edit", "Delete"] else 60, anchor='center')

    stock\_tree.pack(padx=20, pady=10, fill='x')

    # Tag configurations for row colors

    stock\_tree.tag\_configure('alert', background='#EDA0A0')

    def refresh\_stock\_table():

        get\_updated\_stock()

        for row in stock\_tree.get\_children():

            stock\_tree.delete(row)

        for index, item in enumerate(stock\_data):

            tag = ''

            if item[2]<=10:

                tag='alert'

            print()

            stock\_tree.insert('', 'end', values=(\*item, '✏', '🗑'), tags=(tag,))

    def on\_row\_click(event):

        region = stock\_tree.identify("region", event.x, event.y)

        if region != "cell":

            return

        column = stock\_tree.identify\_column(event.x)

        row\_id = stock\_tree.identify\_row(event.y)

        if not row\_id:

            return

        values = stock\_tree.item(row\_id)['values']

        item\_code = values[0]

        if column == "#5":  # Edit column

            item\_name, quantity, price = values[1], values[2], values[3]

            item\_entry.delete(0, tk.END)

            quantity\_entry.delete(0, tk.END)

            price\_entry.delete(0, tk.END)

            item\_entry.insert(0, item\_name)

            quantity\_entry.insert(0, quantity)

            price\_entry.insert(0, price)

            # Change Add button to Update behavior

            def update\_stock():

                updated\_item = item\_entry.get()

                updated\_quantity = quantity\_entry.get()

                updated\_price = price\_entry.get()

                if updated\_item and updated\_quantity.isdigit() and updated\_price.replace('.', '', 1).isdigit():

                    query = f'UPDATE stock SET item\_name=%s, quantity=%s, price=%s WHERE id="{item\_code}"'

                    run\_query(query, data\_to\_insert=(updated\_item, int(updated\_quantity), float(updated\_price)))

                    refresh\_stock\_table()

                    update\_reports()

                    update\_item\_dropdown()

                    item\_entry.delete(0, tk.END)

                    quantity\_entry.delete(0, tk.END)

                    price\_entry.delete(0, tk.END)

                    update\_btn.config(text="Add Stock", command=add\_stock)

                    add\_frame.config(text='Add New Stock')

            update\_btn.config(text="Update Stock", command=update\_stock)

            add\_frame.config(text='Update Existing Stock')

        elif column == "#6":  # Delete column

            confirm = messagebox.askyesno("Confirm Delete", f"Delete item '{values[1]}'?")

            if confirm:

                run\_query(f'DELETE FROM stock WHERE id="{item\_code}"', update\_to\_db=True)

                refresh\_stock\_table()

                # update\_reports()

                # update\_item\_dropdown()

    stock\_tree.bind("<Button-1>", on\_row\_click)

    add\_frame = tk.LabelFrame(stocks\_tab, text="Add New Stock", font=('Helvetica', 14, 'bold'),

                              bg='white', padx=20, pady=20)

    add\_frame.pack(pady=20, padx=20, fill='x')

    tk.Label(add\_frame, text="Item Name:", bg='white', font=('Helvetica', 11)).grid(row=0, column=0, padx=10, pady=10)

    item\_entry = ttk.Entry(add\_frame, font=('Helvetica', 11), width=25)

    item\_entry.grid(row=0, column=1)

    tk.Label(add\_frame, text="Quantity:", bg='white', font=('Helvetica', 11)).grid(row=0, column=2, padx=10, pady=10)

    quantity\_entry = ttk.Entry(add\_frame, font=('Helvetica', 11), width=25)

    quantity\_entry.grid(row=0, column=3)

    tk.Label(add\_frame, text="Price:", bg='white', font=('Helvetica', 11)).grid(row=0, column=4, padx=10, pady=10)

    price\_entry = ttk.Entry(add\_frame, font=('Helvetica', 11), width=25)

    price\_entry.grid(row=0, column=5)

    def clear\_values\_to\_update():

        item\_entry.delete(0, tk.END)

        quantity\_entry.delete(0, tk.END)

        price\_entry.delete(0, tk.END)

        update\_btn.config(text='Add Stock')

        add\_frame.config(text='Add New Stock')

        return 0

    cancel\_btn = tk.Button(add\_frame, text="Cancel",

                       bg=background\_color,

                       activebackground='white',

                       activeforeground='yellow',

                       font=CustomFontStyle(font\_size=10),

                       command=clear\_values\_to\_update)

    cancel\_btn.grid(row=0, column=7, padx=10)

    def add\_stock():

        item = item\_entry.get()

        quantity = quantity\_entry.get()

        price = price\_entry.get()

        if item and quantity.isdigit() and price.replace('.', '', 1).isdigit():

            insert="Insert into stock(item\_name, quantity, price) values(%s,%s,%s)"

            run\_query(insert, data\_to\_insert=(item, int(quantity), float(price)))

            result = run\_query('select \* from stock')

            if len(stock\_data):

                stock\_data.clear()

            for i in result:

                stock\_data.append(i)

            item\_entry.delete(0, tk.END)

            quantity\_entry.delete(0, tk.END)

            price\_entry.delete(0, tk.END)

            refresh\_stock\_table()

            update\_reports()

            update\_item\_dropdown()

        else:

            messagebox.showerror("Invalid Input", "Enter valid name, quantity, and price.")

    update\_btn = tk.Button(add\_frame, text="Add Stock",

                       bg=background\_color,

                       activebackground='white',

                       activeforeground='yellow',

                       font=CustomFontStyle(font\_size=10),

                       command=add\_stock)

    update\_btn.grid(row=0, column=6, padx=10)

    # ======== Memo Tab ========

    memo\_tab = tk.Frame(notebook, bg='#e6f2ff')

    notebook.add(memo\_tab, text='Memo')

    memo\_frame = tk.LabelFrame(memo\_tab, text="Generate Cash Memo", font=('Helvetica', 14, 'bold'),

                           bg='white', padx=5, pady=5)

    memo\_frame.pack(pady=20, padx=20, fill='x')

    tk.Label(memo\_frame, text="Select Item:", bg='white', font=('Helvetica', 11)

             ).grid(row=0, column=0, sticky='w', padx=(5, 2), pady=5)

    selected\_item = tk.StringVar()

    item\_dropdown = ttk.Combobox(memo\_frame, textvariable=selected\_item, font=('Helvetica', 11),

                             width=20, state='readonly')

    item\_dropdown.grid(row=0, column=1, sticky='w', padx=(2, 5), pady=5)

    tk.Label(memo\_frame, text="Quantity:", bg='white', font=('Helvetica', 11)

    ).grid(row=0, column=2, sticky='w', padx=(5, 2), pady=5)

    quantity\_input = ttk.Entry(memo\_frame, font=('Helvetica', 11), width=10)

    quantity\_input.grid(row=0, column=3, sticky='w', padx=(2, 5), pady=5)

    purchase\_items = []

    def add\_to\_purchase\_list():

        item\_name = selected\_item.get()

        qty\_str = quantity\_input.get()

        if not item\_name or not qty\_str.isdigit():

            messagebox.showerror("Invalid Input", "Please select item and enter valid quantity.")

            return

        qty = int(qty\_str)

        index = 0

        for item in stock\_data:

            if item[1] == item\_name:

                if qty > item[2]:

                    messagebox.showerror("Stock Error", "Not enough stock available.")

                    return

                total = qty \* item[3]

                purchase\_items.append([index,item\_name, item[3], qty, total])

                index+=1

                refresh\_purchase\_table()

                return

    def refresh\_purchase\_table():

        memo\_tree.delete(\*memo\_tree.get\_children())

        for idx, (\_, item, price, qty, total) in enumerate(purchase\_items, start=1):

            memo\_tree.insert('', 'end', values=(idx, item, f"₹{price:.2f}", qty, f"₹{total:.2f}",'✏', '🗑'))

    def generate\_memo():

        if not purchase\_items:

            messagebox.showwarning("No Items", "Add at least one item to generate memo.")

            return

        memo\_id = int(str(uuid.uuid4().int)[:5])

        total\_amount = sum(item[4] for item in purchase\_items)

        memo\_text = f"========== CASH MEMO {memo\_id} ==========\n"

        for \_,item, price, qty, total in purchase\_items:

            insert\_query="Insert into sales(memo\_id,item\_name,price,quantity,total\_amount) values(%s, %s, %s, %s, %s)"

            run\_query(insert\_query, data\_to\_insert=(memo\_id,item,price,qty,total))

            run\_query(f'update stock set quantity=quantity-{qty} where item\_name="{item}"',

                      update\_to\_db=True)

            print(item,qty,price,total)

            memo\_text += f"{item:15} x {qty:<3} @ ₹{price:<7.2f} = ₹{total:.2f}\n"

        memo\_text += "----------------------------------------\n"

        memo\_text += f"TOTAL BILL: ₹{total\_amount:.2f}\n"

        memo\_text += "========================================\n"

        messagebox.showinfo("Memo Generated", memo\_text)

        purchase\_items.clear()

        memo\_edit\_frame.pack\_forget()

        refresh\_purchase\_table()

    tk.Button(memo\_frame, text="Add items to purchase list",

          bg=background\_color,

          activebackground='white',

          activeforeground='yellow',

          command=add\_to\_purchase\_list).grid(row=0, column=4, padx=(10, 5), pady=5)

    tk.Button(memo\_frame, text="Generate Memo",

          bg=background\_color,

          activebackground='white',

          activeforeground='yellow',

          command=generate\_memo).grid(row=0, column=5, padx=(5, 10), pady=5)

    # Purchase Items Table

    memo\_tree = ttk.Treeview(memo\_tab, columns=("S.No", "Item", "Price", "Qty", "Total",'Edit','Delete'), show='headings')

    for col in ("S.No", "Item", "Price", "Qty", "Total",'Edit','Delete'):

        memo\_tree.heading(col, text=col)

        memo\_tree.column(col, width=100 if col not in ["Edit", "Delete"] else 60, anchor='center')

    memo\_tree.pack(padx=20, pady=20, fill='x')

    def update\_item\_dropdown():

        item\_dropdown['values'] = [item[1] for item in stock\_data]

    memo\_edit\_frame = tk.LabelFrame(memo\_tab, text="Edit Purchase Items", font=('Helvetica', 14, 'bold'),

                           bg='white', padx=5, pady=5)

    item\_index\_want\_updated\_quantity = 0

    def on\_row\_click(event):

        region = memo\_tree.identify("region", event.x, event.y)

        if region != "cell":

            return

        column = memo\_tree.identify\_column(event.x)

        row\_id = memo\_tree.identify\_row(event.y)

        if not row\_id:

            return

        values = memo\_tree.item(row\_id)['values']

        item\_code = values[0]

        if column == "#6":  # Edit column

            nonlocal item\_index\_want\_updated\_quantity

            item\_index\_want\_updated\_quantity = values[0]-1

            if not edited\_quantity.get().strip() == '':

                edited\_quantity.delete(0,tk.END)

            edited\_quantity.insert(0,values[3])

            memo\_edit\_frame.pack(padx=10, pady=10, fill='x')

        elif column == "#7":  # Delete column

            confirm = messagebox.askyesno("Confirm Delete", f"Delete item '{values[1]}'?")

            if confirm:

                purchase\_items.pop(item\_code-1)

                refresh\_purchase\_table()

    print(purchase\_items)

    memo\_tree.bind("<Button-1>", on\_row\_click)

    tk.Label(memo\_edit\_frame, text="Quantity:", bg='white', font=('Helvetica', 11)

    ).grid(row=0, column=0)

    edited\_quantity = ttk.Entry(memo\_edit\_frame, font=('Helvetica', 11), width=10)

    edited\_quantity.grid(row=0, column=1, sticky='w', padx=(2, 5), pady=5)

    def change\_quantity():

        e\_quan = edited\_quantity.get()

        purchase\_items[item\_index\_want\_updated\_quantity][3] = int(e\_quan)

        purchase\_items[item\_index\_want\_updated\_quantity][4] = purchase\_items[item\_index\_want\_updated\_quantity][2] \* int(e\_quan)

        print(item\_index\_want\_updated\_quantity)

        print(purchase\_items[item\_index\_want\_updated\_quantity])

        refresh\_purchase\_table()

        memo\_edit\_frame.place\_forget()

    def cancel\_quantity\_edit():

        memo\_edit\_frame.pack\_forget()

        edited\_quantity.delete(0, tk.END)

    tk.Button(memo\_edit\_frame, text="Change quantity",

          bg=background\_color,

          activebackground='white',

          activeforeground='yellow',

          command=change\_quantity).grid(row=0, column=4, padx=(10, 5), pady=5)

    tk.Button(memo\_edit\_frame, text="Cancel change",

          bg=background\_color,

          activebackground='white',

          activeforeground='yellow',

          command=cancel\_quantity\_edit).grid(row=0, column=5, padx=(10, 5), pady=5)

    # ======== Report Tab ========

    report\_tab = tk.Frame(notebook, bg='#e6f2ff')

    notebook.add(report\_tab, text='Report')

    report\_frame = tk.LabelFrame(report\_tab, text="Inventory Report", font=('Helvetica', 14, 'bold'),

                                 bg='white', padx=20, pady=20)

    report\_frame.pack(pady=30, padx=30, fill='x')

    total\_items\_label = tk.Label(report\_frame, text="Total Items: 0", font=('Helvetica', 12), bg='white')

    total\_items\_label.pack(pady=10)

    total\_value\_label = tk.Label(report\_frame, text="Total Inventory Value: ₹0", font=('Helvetica', 12), bg='white')

    total\_value\_label.pack(pady=10)

    report\_sales = tk.LabelFrame(report\_tab, text="Sales Report", font=('Helvetica', 14, 'bold'),

                                 bg='white', padx=20, pady=20)

    report\_sales.pack(pady=30, padx=30, fill='x')

    def generate\_sales\_report():

        generate\_year\_sales\_report()

        generate\_monthly\_sales\_report()

        generate\_weekly\_sales\_report()

    import os

    if os.path.exists('year\_sales.jpg') and os.path.exists('month\_sales.jpg') and os.path.exists('weekly\_sales\_report.jpg'):

        year\_sales\_chart = Image.open('year\_sales.jpg').resize((450, 300))

        year\_sales\_chart\_tk = ImageTk.PhotoImage(year\_sales\_chart)

        report\_sales.year\_image = year\_sales\_chart\_tk

        month\_sales\_chart = Image.open('month\_sales.jpg').resize((450, 300))

        month\_sales\_chart\_tk = ImageTk.PhotoImage(month\_sales\_chart)

        report\_sales.month\_image = month\_sales\_chart\_tk

        weekly\_sales\_chart = Image.open('weekly\_sales\_report.jpg').resize((450, 300))

        weekly\_sales\_chart\_tk = ImageTk.PhotoImage(weekly\_sales\_chart)

        report\_sales.week\_image = weekly\_sales\_chart\_tk

         # Display Yearly Sales Chart

        year\_label = tk.Label(report\_sales, image=report\_sales.year\_image, bg='white')

        year\_label.pack(side='left', padx=10, pady=10)

        # Display Monthly Sales Chart

        month\_label = tk.Label(report\_sales, image=report\_sales.month\_image, bg='white')

        month\_label.pack(side='left', padx=10, pady=10)

        # Display Weekly Sales Chart

        week\_label = tk.Label(report\_sales, image=report\_sales.week\_image, bg='white')

        week\_label.pack(side='left', padx=10, pady=10)

    else:

        tk.Label(report\_sales, text="Couldn't generate report",

                 font=CustomFontStyle(font\_size=10, weight='bold')).pack()

    def update\_reports():

        total\_items = len(stock\_data)

        total\_value = sum(qty \* price for \_, \_, qty, price in stock\_data)

        total\_items\_label.config(text=f"Total Items: {total\_items}")

        total\_value\_label.config(text=f"Total Inventory Value: ₹{total\_value:.2f}")

    # ======== Settings Tab ========

    settings\_tab = tk.Frame(notebook, bg='#e6f2ff')

    notebook.add(settings\_tab, text='Settings')

    settings\_frame = tk.LabelFrame(settings\_tab, text="User Settings", font=('Helvetica', 14, 'bold'),

                                   bg='white', padx=20, pady=20)

    settings\_frame.pack(pady=40, padx=40, fill='x')

    tk.Label(settings\_frame, text="Gmail:", font=('Helvetica', 11), bg='white').grid(row=0, column=0, padx=10, pady=10)

    name\_entry = ttk.Entry(settings\_frame, font=('Helvetica', 11), width=30)

    name\_entry.insert(0, logged\_in\_user\_details[1])

    name\_entry.grid(row=0, column=1)

    tk.Label(settings\_frame, text="Password:", font=('Helvetica', 11), bg='white').grid(row=1, column=0, padx=10, pady=10)

    email\_entry = ttk.Entry(settings\_frame, font=('Helvetica', 11), width=30)

    email\_entry.insert(0, logged\_in\_user\_details[2])

    email\_entry.grid(row=1, column=1)

    def save\_settings():

        nonlocal logged\_in\_user\_details

        new\_email = name\_entry.get().strip()

        new\_password = email\_entry.get().strip()

        if new\_email and new\_password:

            is\_name\_changed = new\_email != logged\_in\_user\_details[1]

            is\_password\_changed = new\_password != logged\_in\_user\_details[2]

            message\_label = ''

            if is\_name\_changed or is\_password\_changed:

                if is\_name\_changed:

                    result = run\_query(f'Update Users Set gmail="{new\_email}" where id={logged\_in\_user\_details[0]}', update\_to\_db=True,

                                       reterival\_id=logged\_in\_user\_details[0])

                    logged\_in\_user\_details = result[0]

                    message\_label += 'Gmail '

                if is\_password\_changed:

                    result = run\_query(f'Update Users Set password="{new\_password}" where id={logged\_in\_user\_details[0]}', update\_to\_db=True,

                                       reterival\_id=logged\_in\_user\_details[0])

                    logged\_in\_user\_details = result[0]

                    message\_label += 'Password '

                new\_name = logged\_in\_user\_details[1].split('@')[0]

                user\_label.config(text=f"Logged in as: {new\_name}")

                messagebox.showinfo("Success", f"User {message\_label} updated successfully.")

        else:

            messagebox.showerror("Invalid Input", "Both fields are required.")

    tk.Button(settings\_frame, text="Save Changes", command=save\_settings,

               bg=background\_color,

                   activebackground='white',

                   activeforeground='yellow',

                   ).grid(row=2, column=1, pady=20)

    # refresh\_stock\_table()

    # #update\_reports()

    # update\_item\_dropdown()

    def on\_tab\_change(event):

        notebook = event.widget

        selected\_tab = notebook.select()

        tab\_index = notebook.index(selected\_tab)

        if tab\_index==0:

            refresh\_stock\_table()

        elif tab\_index==1:

            update\_item\_dropdown()

        elif tab\_index==2:

            refresh\_stock\_table()

            update\_reports()

            generate\_sales\_report()

    notebook.bind("<<NotebookTabChanged>>", on\_tab\_change)

    root.mainloop()

#stock\_app((1,'user','pass','pass'))

**Project Overview**

This project is a **Stock Management System** designed to help users manage inventory, generate sales memos, and analyze reports. It supports stock tracking, invoice creation, real-time sales updates, and user authentication. All interactions are performed via a user-friendly GUI built with Tkinter and data is stored in MySQL.

**Technologies Used in the Project**

* **Python 3.x** – Programming language for backend logic and GUI.
* **Tkinter** – GUI library for creating desktop interfaces.
* **MySQL** – Relational database for storing user, stock, and sales data.
* **ttk (Themed Tkinter)** – Enhanced widgets for better GUI appearance.
* **uuid, datetime, random** – For unique ID generation and time-stamping.

**Project Functionality**

**A. Login Flow**

* The application starts at the **welcome screen** with a "Login" button.
* Clicking "Login" opens a new window for user login (email & password).
* If the user doesn't exist, they can click "Register" to open a registration form and store credentials in the database.

**B. Dashboard Tabs (After Login)**

Once authenticated, the user accesses a tabbed interface with three main tab

**Functional Overview (Login to Settings Tab)**

**A. Login/Register**

* Login requires **Gmail and password**.
* Register window collects the same fields and stores them in the MySQL database.
* Validations include:
  + Non-empty fields,
  + Proper email format (contains '@'),
  + Password minimum length check.

**B. Stock Tab**

* Add, Edit, or Delete stock items.
* Each stock entry includes **name**, **quantity**, and **price**.
* A Treeview table displays current stock.
* **Stock items with quantity < 10 are automatically highlighted in red** to signal low inventory.
* Validations:
  + Name should not be blank.
  + Quantity and price must be valid positive numbers.

**C. Memo Tab**

* Allows user to:
  + Select stock items,
  + Enter quantity sold,
  + Generate a memo with itemized breakdown.
* Finalized memos deduct quantity from stock.
* Validations include:
  + Checking if entered quantity is available in stock,
  + Preventing duplicate item selection.

**D. Report Tab**

Displays both Inventory and Sales performance using **charts**:

* **Inventory Report**:
  + Shows total number of items and total stock value (₹).
* **Sales Report**:
  + **Yearly Sales** – Total sales amount by year.
  + **Monthly Sales** – Bar chart showing monthly sales trend for 2025.
  + **Weekly Sales** – Weekly breakdown showing performance spikes.

These charts are generated using **Matplotlib** and updated dynamically from the database.

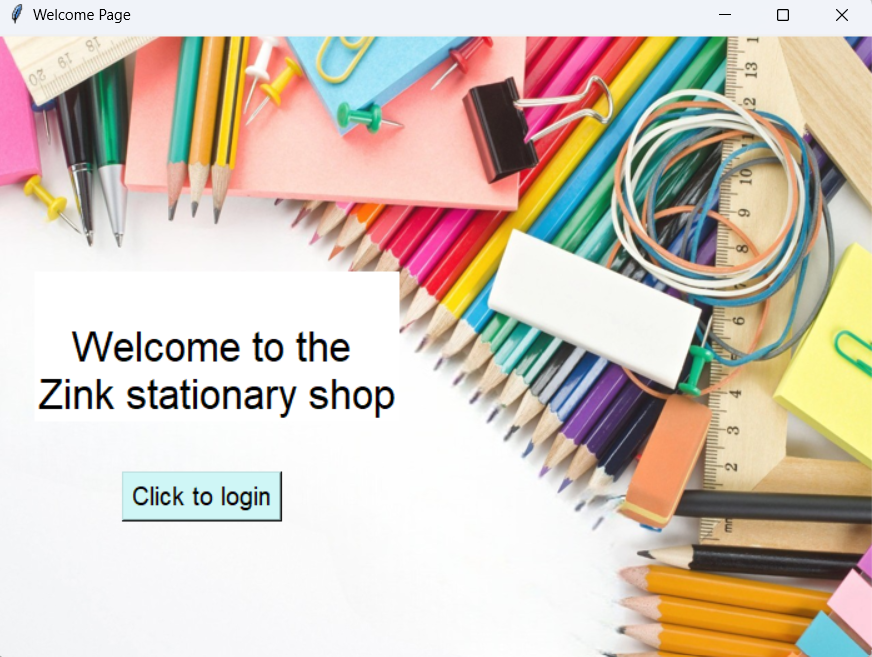
**E. Settings Tab**

* Displays the currently logged-in user.
* Allows user to **update their email and password** with proper validation:
  + Email format check,
  + Password length and confirmation match.

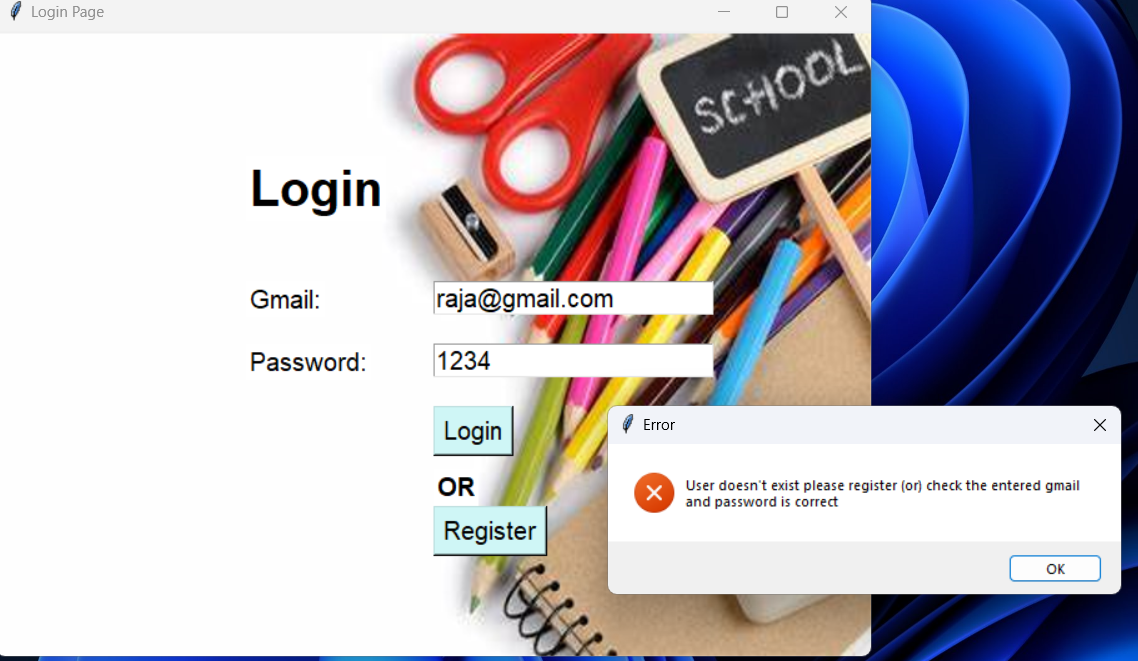
🔒 *Note: No logout or account deletion feature is currently available in the Settings tab.*

**FIGURE:**

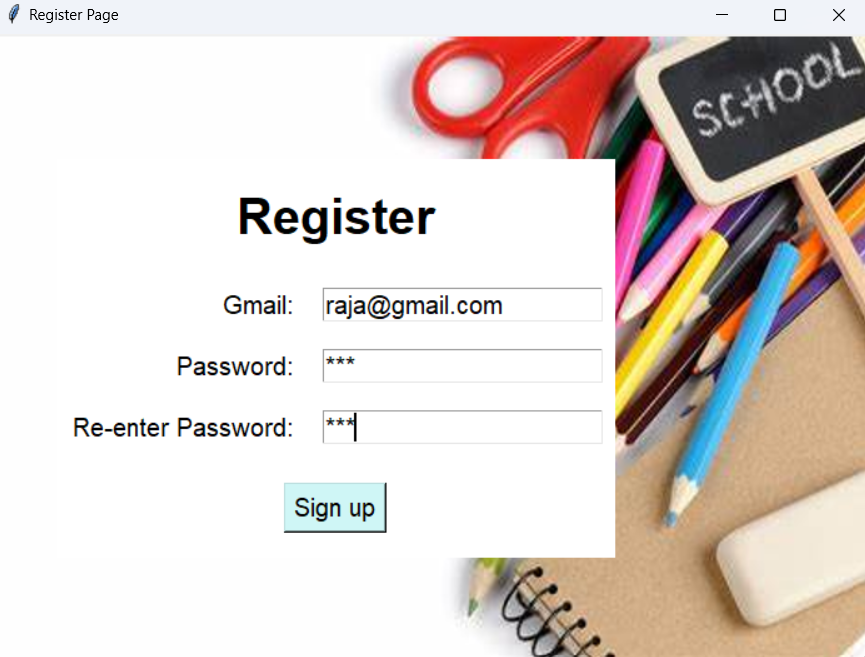
**WELCOME WINDOW:**

****

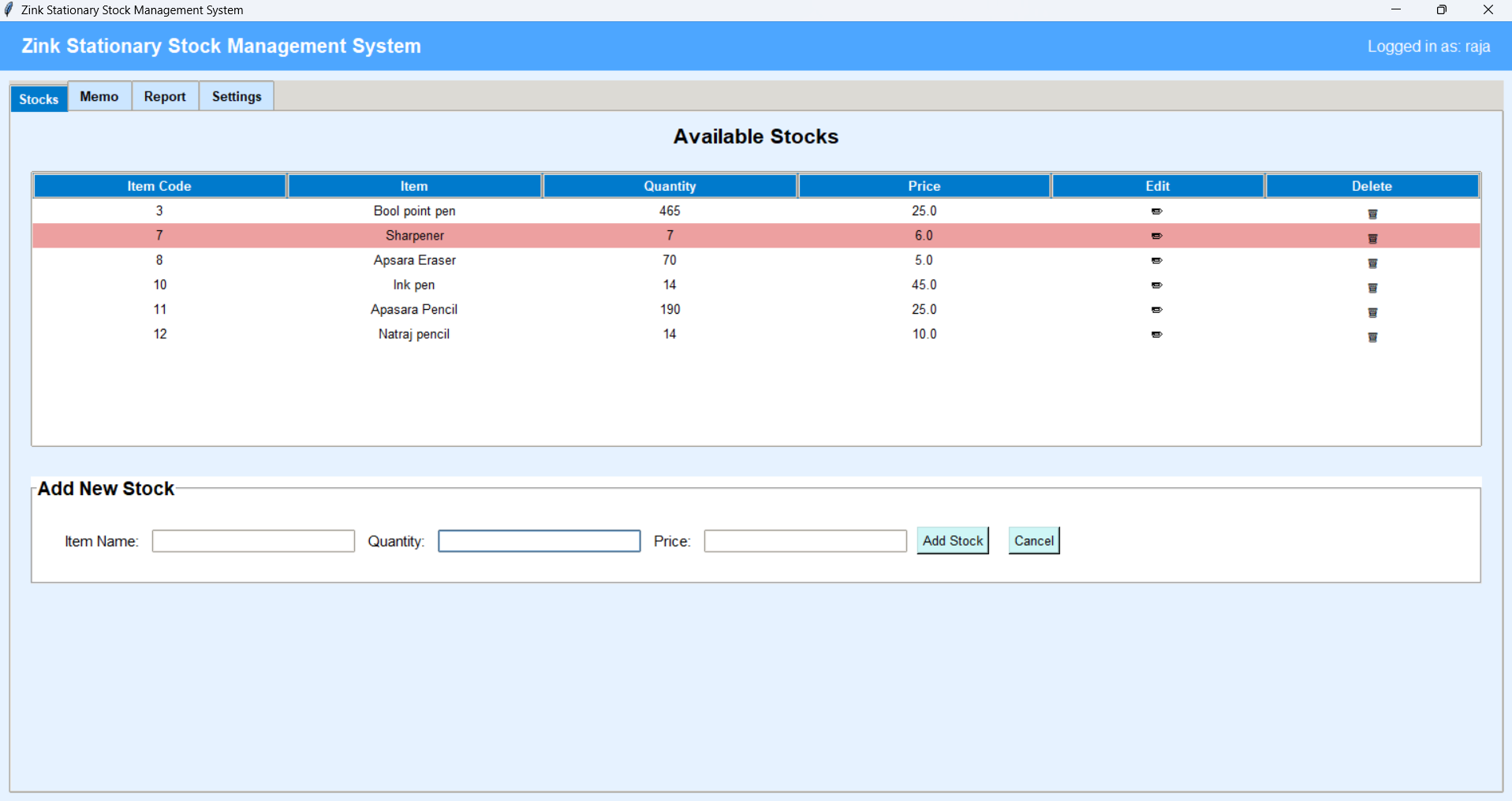
**LOIGN WINDOW:**

****

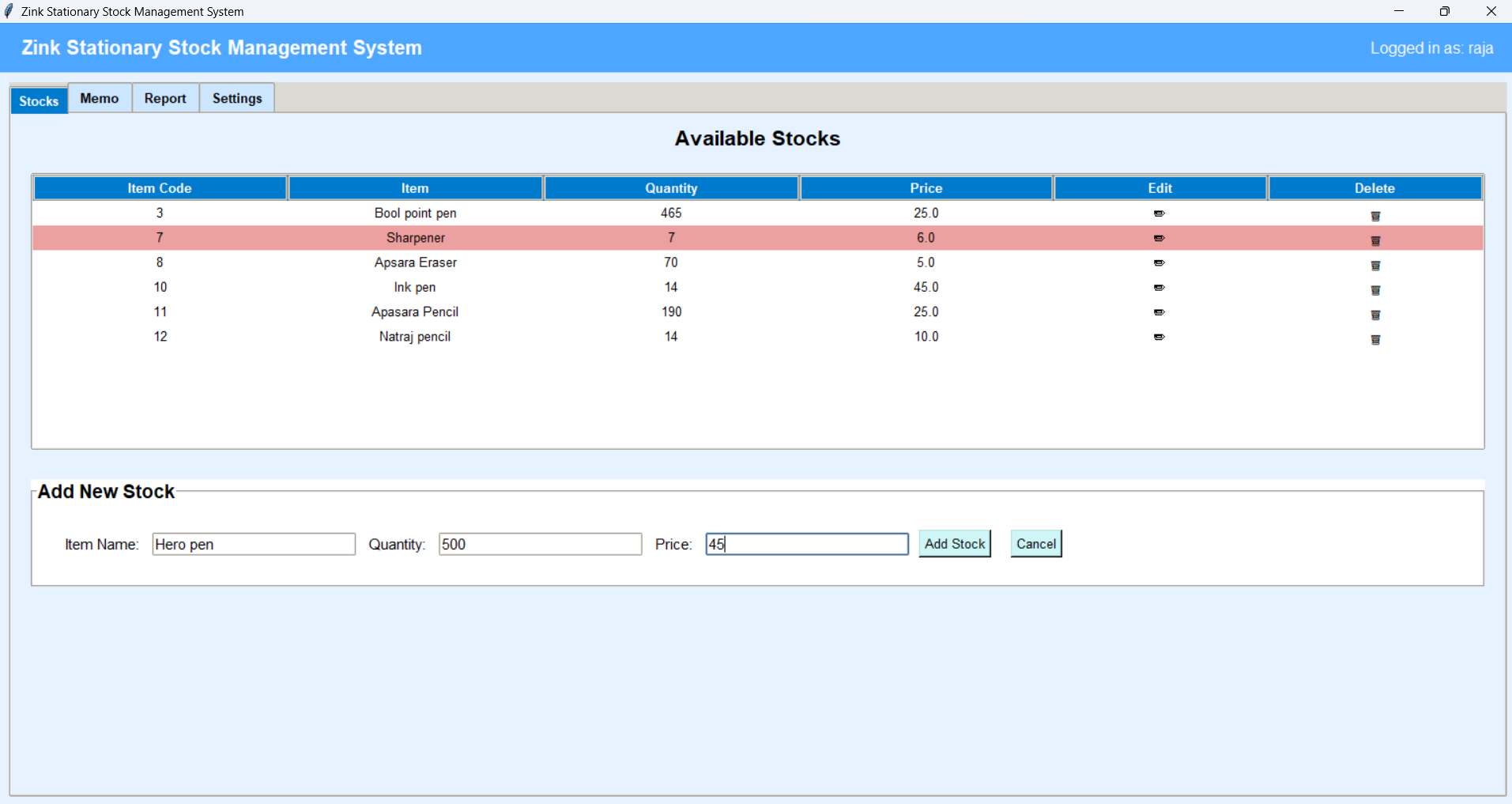
**REGISTER WINDOW:**

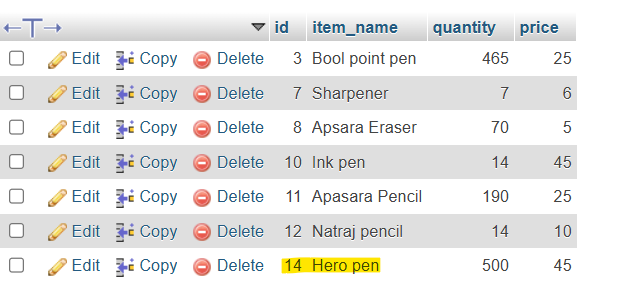
****

**MAIN WINDOW:**

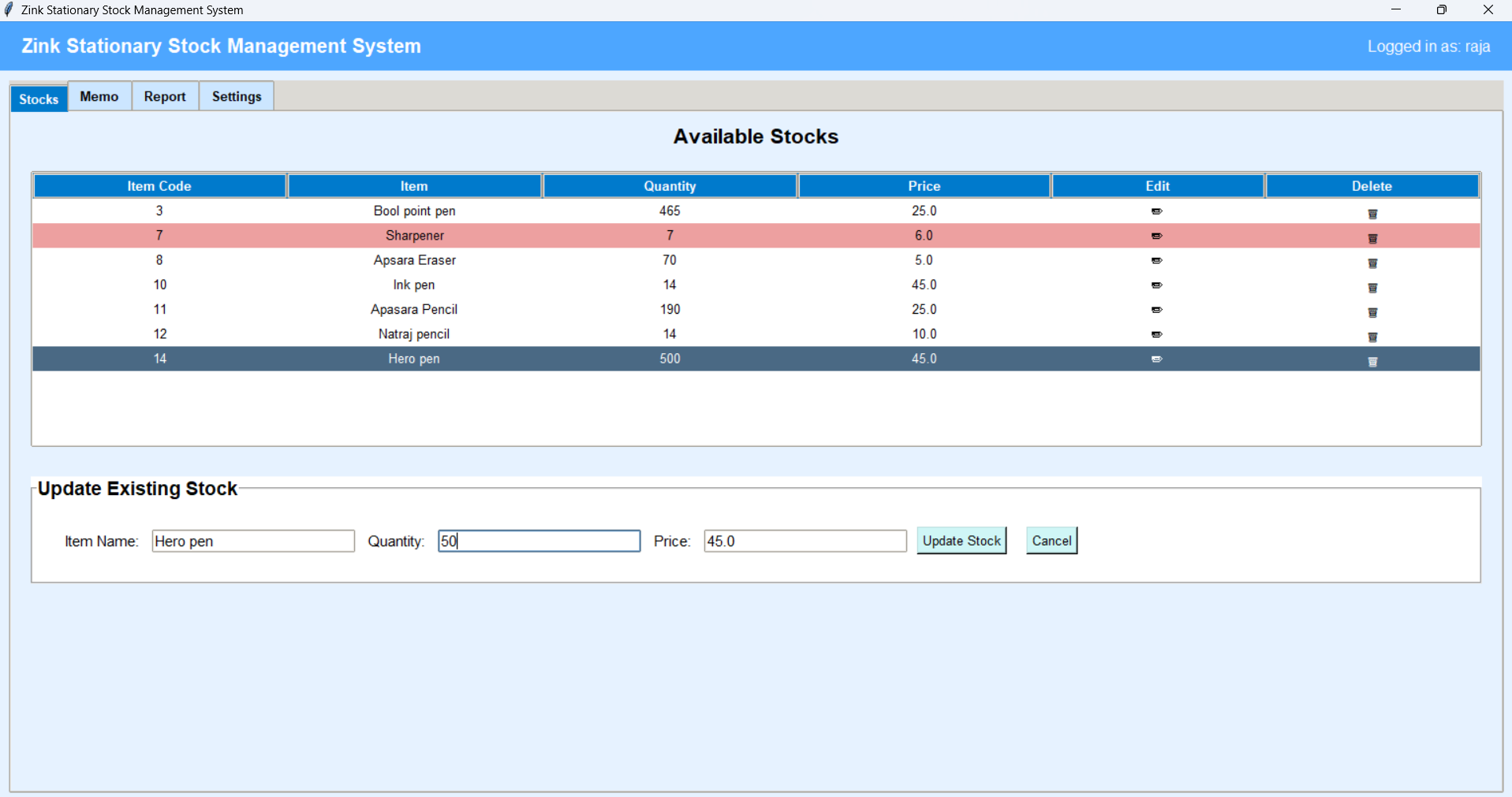
****

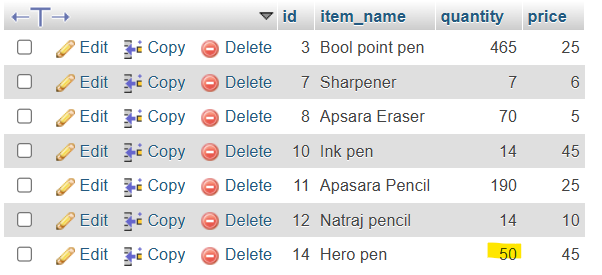
**STOCK INSERT:**

****

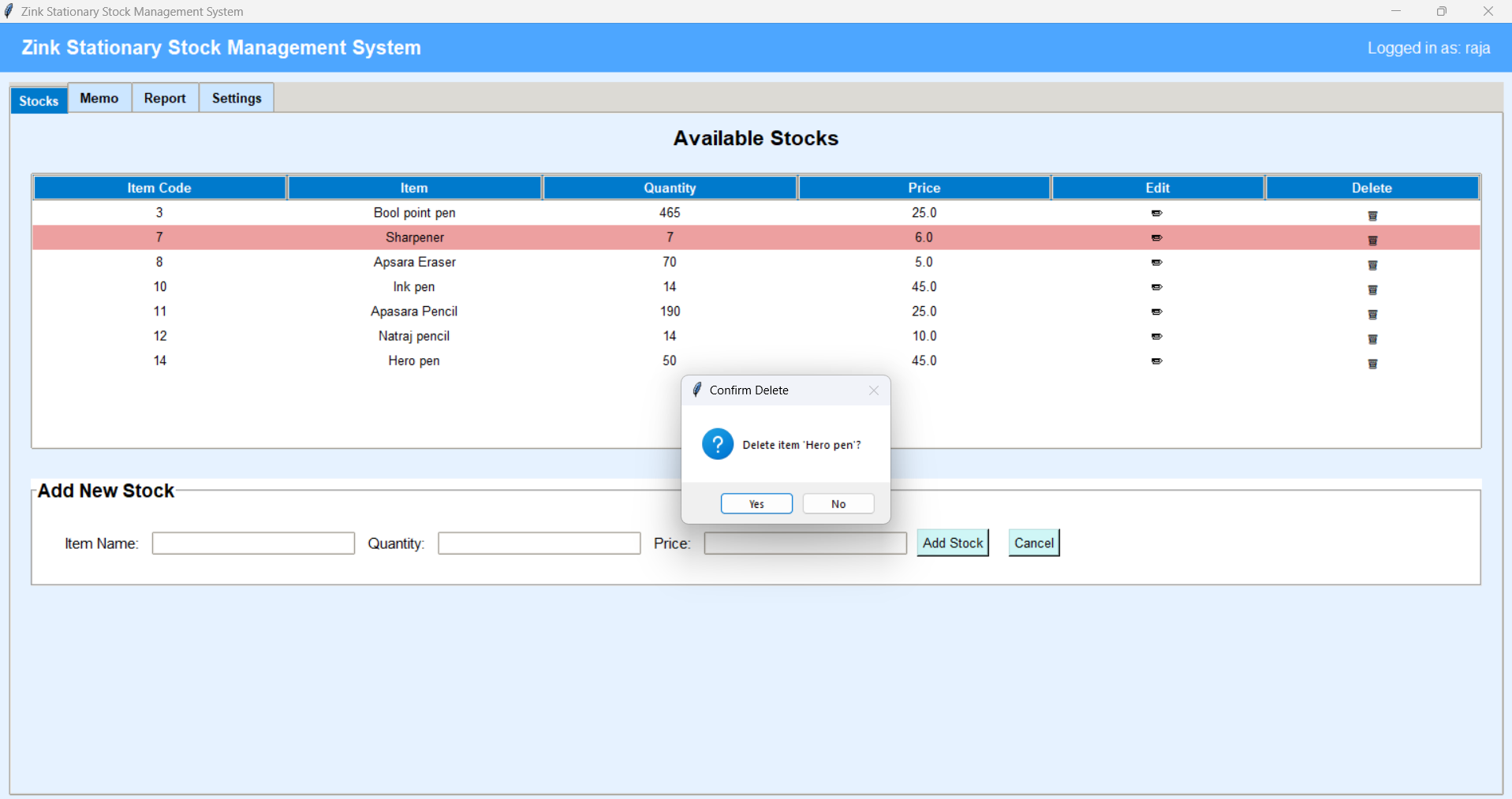
****

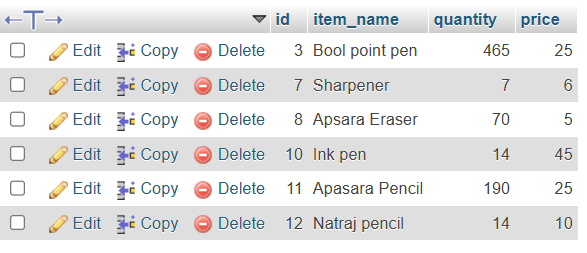
**STOCK UPDATE:**

****

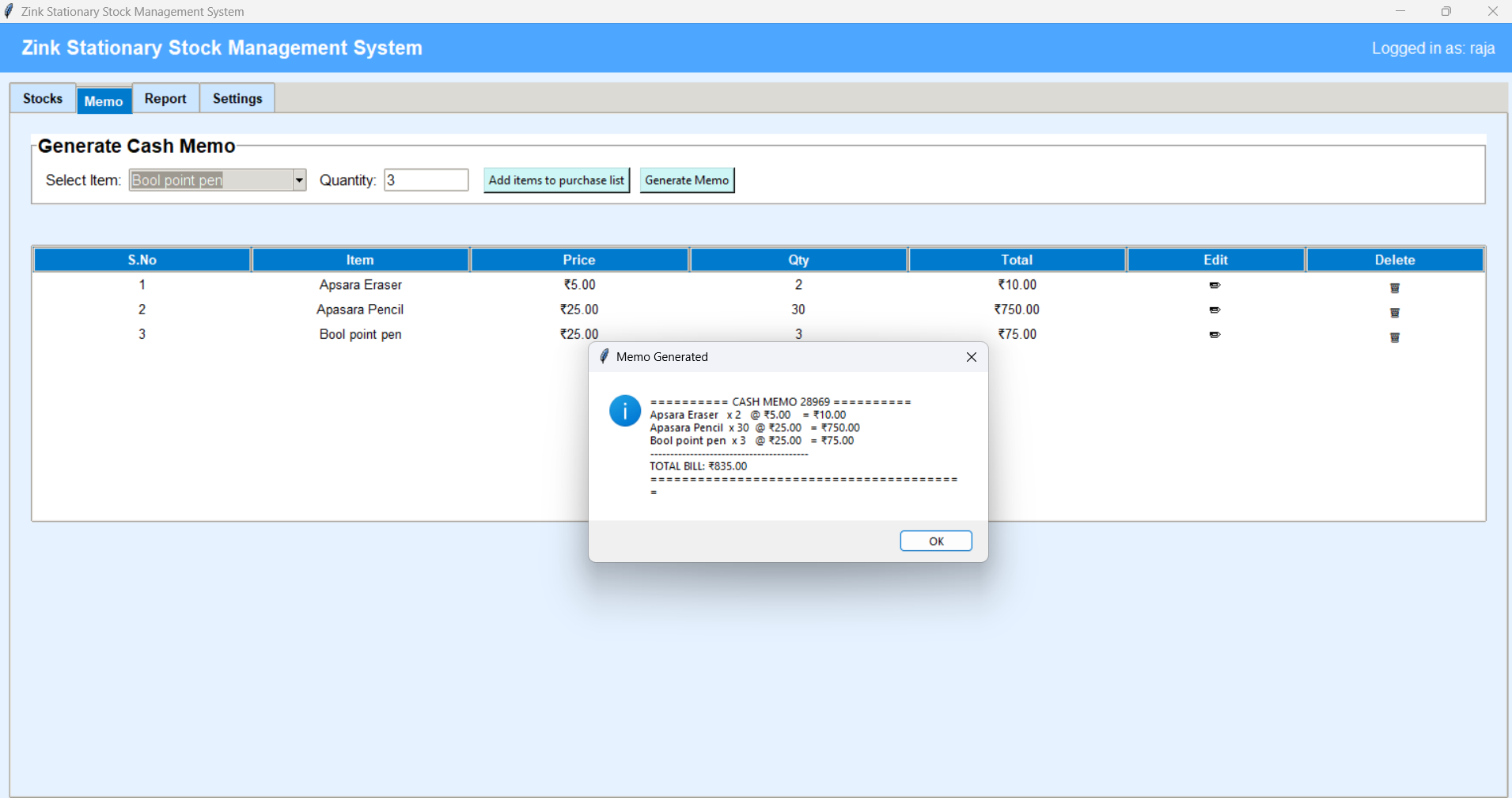
****

**STOCK DELETE:**

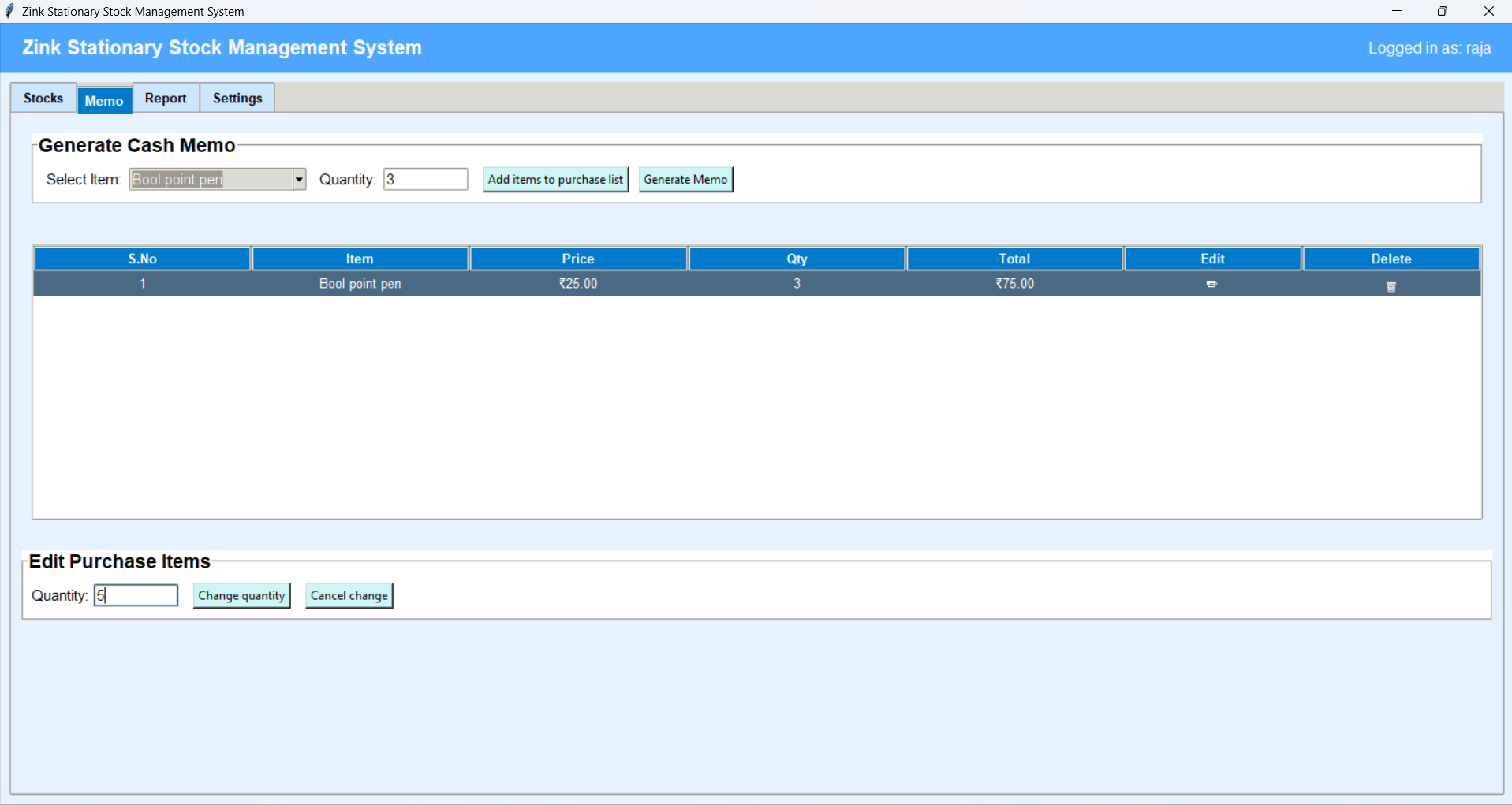
****

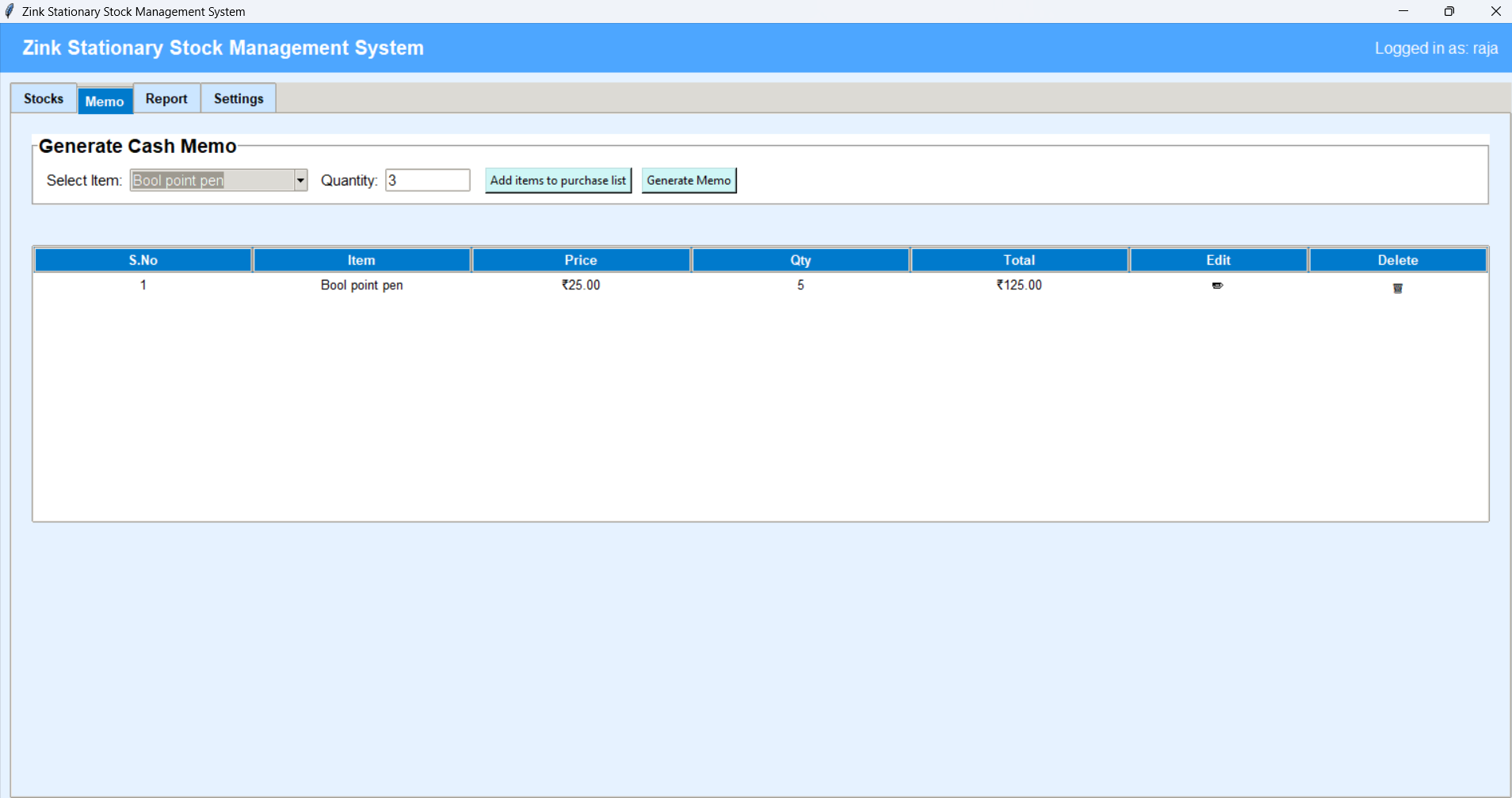
****

**MEMO WINDOW(Generate memo):**

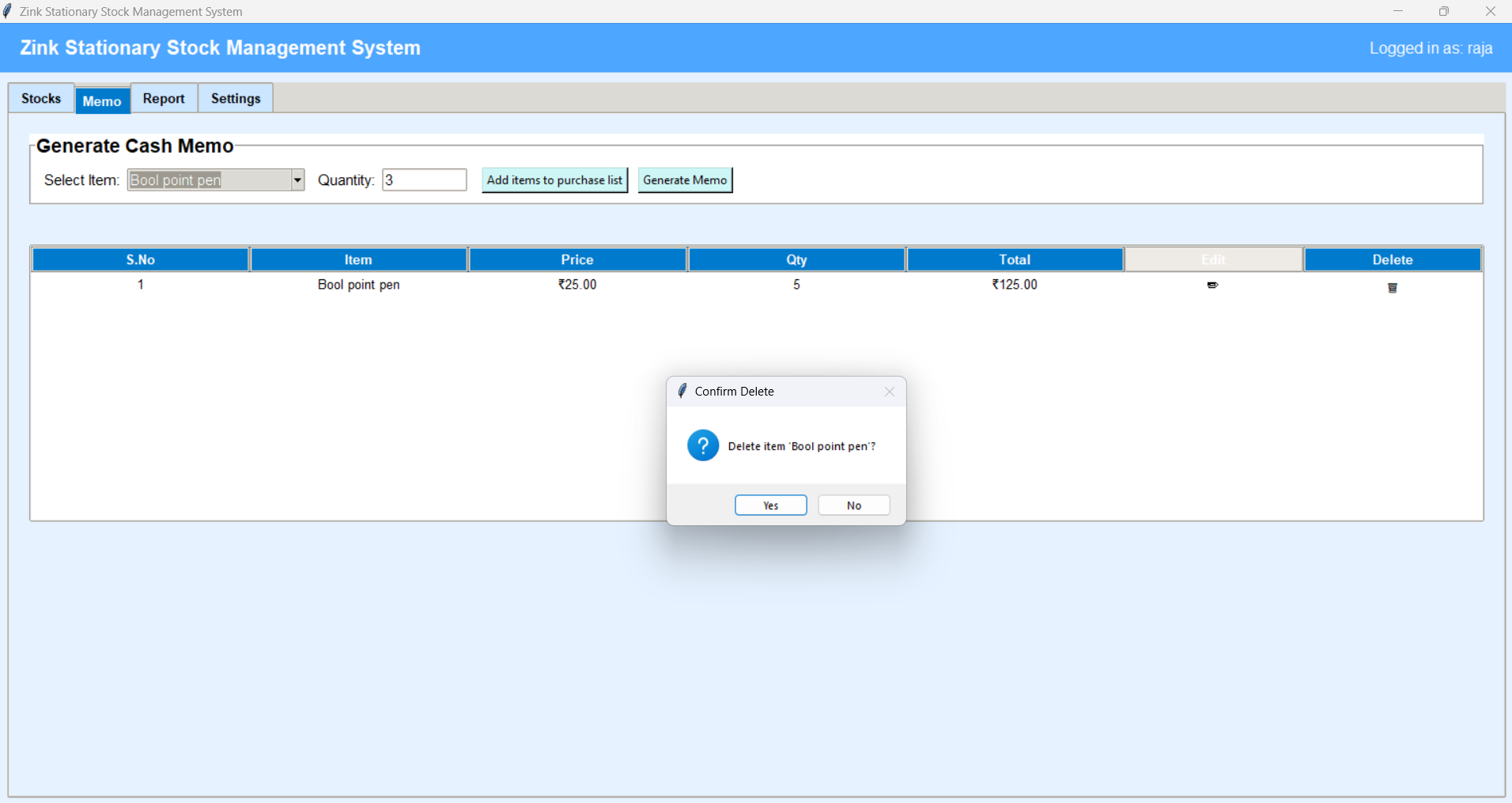
****

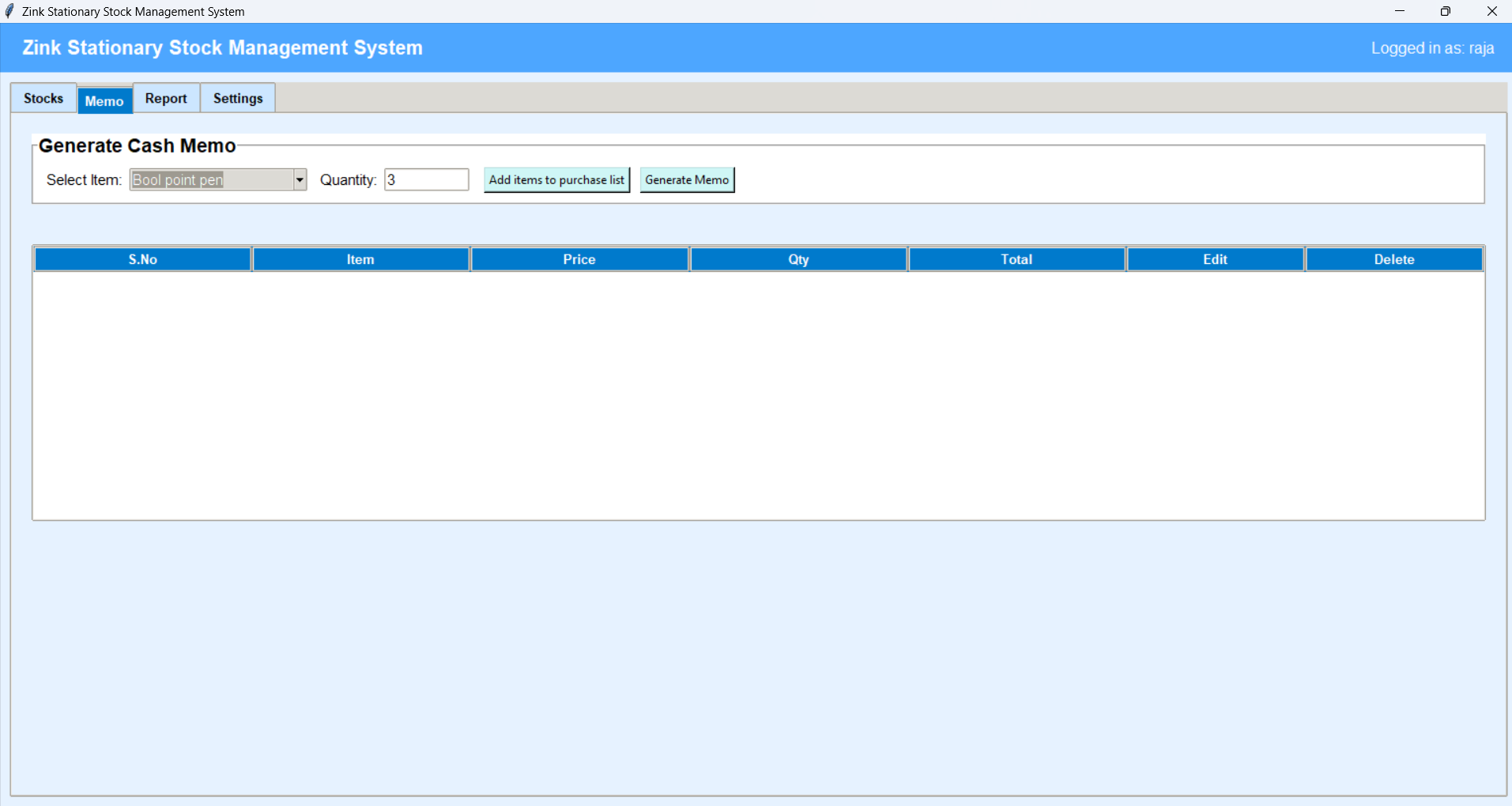
**UPDATE QUANTITY OF PURCHASE ITEM**

****

****

**DELETE PURCHASE ITEM**

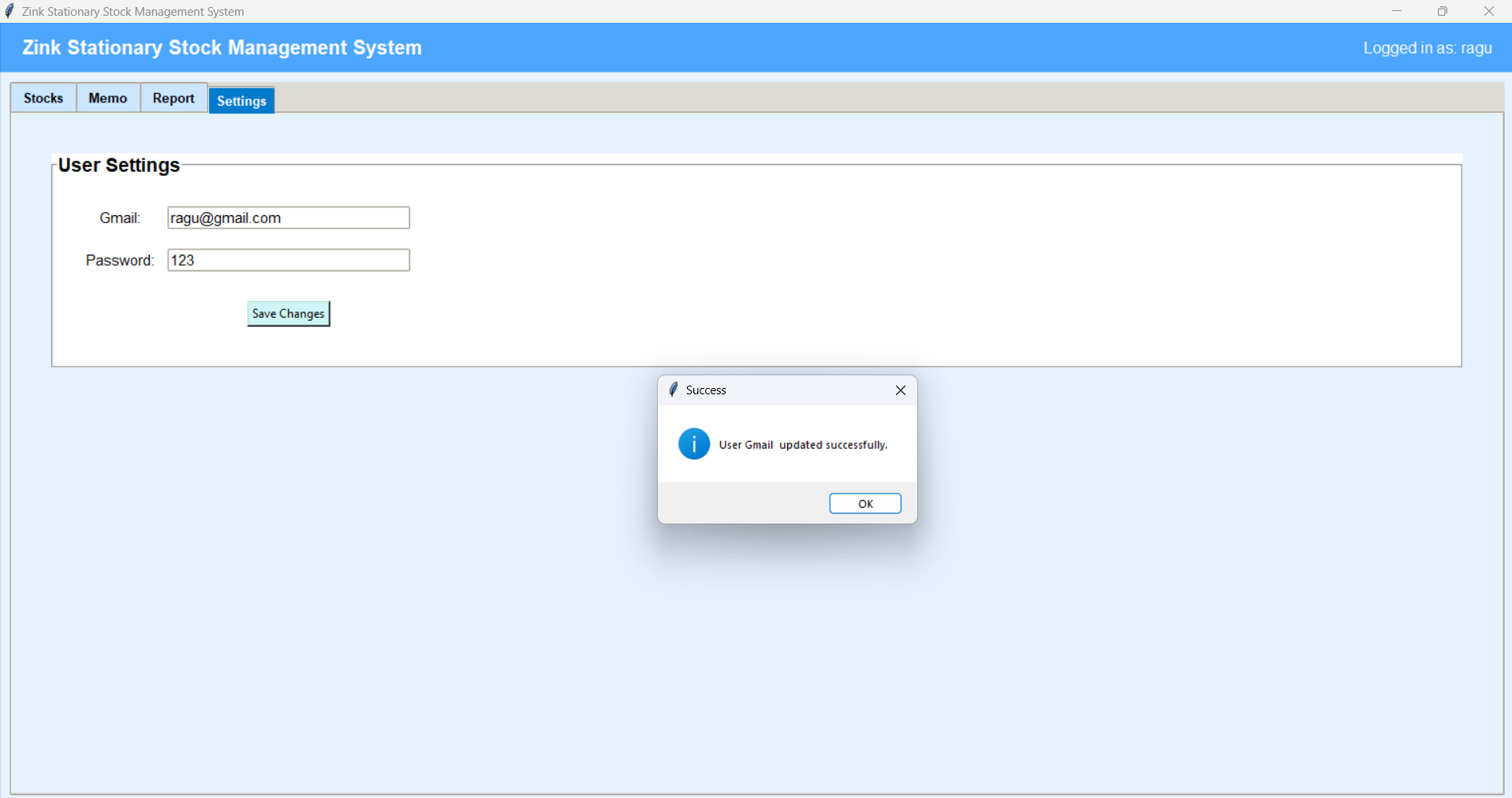
****

****

**REPORT WINDOW**

****

**SETTINGS WINDOW(Update user mail id)**

****

**Conclusion**

The Stock Management System project successfully demonstrates how a desktop application can be used to efficiently manage inventory, handle sales transactions, and analyze business performance using graphical reports. Built with Python’s Tkinter for the user interface and MySQL for backend storage, the system offers a user-friendly experience for both technical and non-technical users.

Key features such as user authentication, low-stock alerts, memo generation, and dynamic sales charts ensure that essential business operations are streamlined and transparent. With input validations and real-time updates, the system reduces manual errors and improves decision-making efficiency.

Overall, this project serves as a practical solution for small businesses or shops looking to digitize their inventory and sales process with a reliable and easy-to-use application. Future enhancements like exporting reports, adding user roles, and enabling cloud database support can further improve its scalability and utility.