Smart Shelf Grocers

SHOP ONLINE

Smart Grocery Shopping Interface

Shivay Seth

Acknowledgements

We would like to thank our computer science teacher, Paulina Ma'am, who helped us develop and expand this project through her suggestions and teachings. She constantly advised and assisted us wherever we needed help, allowing us to produce this project.

Secondly, we would also like to thank the YouTuber @Codemy.com who helped us understand the concepts of Tkinter and how to implement them in our code.

Overview of Python, Tkinter, and MySQL

Python is a versatile programming language known for its ease of use and adaptability across various applications. As an interpreted language, Python executes code line by line, providing immediate feedback and error correction, simplifying development for beginners. Its high-level nature abstracts hardware complexities, allowing developers to focus on problem-solving. Python is open-source, free to use, and has a clear syntax that enhances readability, with a rich standard library that simplifies tasks like file handling. Case sensitivity ensures variable names are treated distinctly, emphasizing naming consistency. Python's versatility extends to web development, where frameworks like Flask and Django make it easy to create dynamic websites.

Tkinter, Python's standard library for graphical user interfaces (GUIs), enables developers to design interactive applications with elements like buttons, labels, and text boxes. Its simplicity appeals to beginners, allowing them to create windows and add widgets easily. Tkinter supports a range of designs, from simple pop-ups to more complex applications, operating on an event-driven model where user actions trigger responses.

MySQL, an open-source relational database system, manages data using Structured Query Language (SQL). It organizes data into tables and integrates with Python through libraries like MySQL Connector, enabling direct database operations from Python code. Beginners can easily create databases, insert records, and query data, building a foundation for managing more complex applications. Python, Tkinter, and MySQL together form a powerful toolkit for developing diverse applications, ideal for those new to programming and data management.

Summary Of The Project

The **Smart Shelf Grocers** project is an innovative Python application designed for a seamless shopping experience, structured into three primary files: main.py, Login_Button.py, and shopping.py, each serving a distinct purpose.

1. main.py

The main.py file functions as the entry point for the application, providing a user-friendly greeting page. This page features a welcoming message, a visually appealing logo, and essential buttons for users to either continue to the shopping interface or exit the application. This file does not integrate database connections, focusing solely on user interaction at the initial stage.

2. Login_Button.py

In Login_Button.py, the application handles user authentication by verifying login credentials. It reads usernames and passwords from a credentials.csv file, securely storing user data. The system evaluates the strength of passwords, providing real-time feedback through a password length checker. When a password is entered, its length is evaluated, and the background color of the input field changes based on its validity. If the password meets the required length, it may turn green, indicating acceptability; otherwise, it turns red to signal that the password is too short. The login window offers options for both registration and login, enhancing user experience and ensuring that only registered users can access shopping features.

3. shopping.py

The core of the application lies in shopping.py, where users can browse various grocery items. It establishes a connection to a MySQL database, creating a dedicated database named smart_shelf_grocer if it does not already exist. Within this database, a transactions table is created to store details of each purchase, including the user's name, item names, quantities, and timestamps of transactions (date and time). The application allows users to view items with their corresponding images and prices, featuring a dynamic cart button that updates in real-time to reflect the number of items added. Users can add or remove items from their cart, and upon checkout, the system saves transaction data to the MySQL database for record-keeping.

The shopping experience is further enhanced with a search function, enabling users to filter items based on their queries. The application displays selected items in a cart summary, showing quantities and total prices before finalizing the transaction. Upon successful payment, a “Thank You” screen is presented, reinforcing a positive user experience.

Resources and Concepts used

1. Python Programming Language:
   - The entire project utilizes Python, enabling the implementation of features like user authentication and shopping cart management. For example, the logic to check if a username already exists during registration is written in Python.

2. Tkinter for GUI Development:
   - Tkinter is used to create the graphical interface. For instance, buttons for 'Login' and 'Register' are designed using Tkinter, allowing users to interact with the application.

3. CSV for Authentication:
   - The `credentials.csv` file is utilised to store usernames and passwords. When a user registers, their details are written to this file, allowing for quick authentication during login.

4. Password Strength Checker:
   - The system evaluates password strength dynamically; for example, the password entry field changes colour based on the password's length. A password that meets the minimum length criterion turns green, providing immediate feedback to users.

5. Dynamic Cart Updates:
   - The shopping interface includes a cart button that dynamically updates to show the number of items added. When a user adds a product, the cart icon reflects the new count, enhancing the user experience.

6. MySQL for Data Management:
   - MySQL is employed to manage transaction records. Each time a user makes a purchase, details like item name, quantity, user name, and transaction timestamp are stored in a MySQL database table, ensuring organised data management.

7. DateTime Module:
   - The `datetime` module is used to timestamp transactions accurately. For instance, when a purchase is made, the current date and time are recorded in the MySQL database, providing essential logs for future reference.

8. User-Defined Functions:
   - Functions are created for specific tasks, such as `add_to_cart()` and `authenticate_user()`. These functions streamline the code, making managing and updating individual functionalities easier.

9. Dictionaries:
- Dictionaries manage cart data effectively; for example, an item could be represented as a dictionary with keys for `name` and `quantity`. Lists containing these dictionaries allow for easy updates and retrieval of multiple items in the cart.

10. Error Handling:
- Error handling is implemented to manage invalid logins. For example, if a user enters incorrect credentials, an error message is displayed, prompting them to try again without crashing the application.

11. Global Variables:
- Global variables are used to store user session data, such as the current username. This allows different parts of the application to access the user information without needing to pass it through function parameters.

Requirements

- You must have pip installed
- The program must be downloaded from the link provided as a zipped folder (.zip) and extracted
- Pillow must be installed using the following command
    - $Pip install pillow
- Mysql-connector-python must be installed.
    - $pip install mysql-connector-python

Functions

File 1 - Main.py

Function 1: ‘exit_command’

```
def exit_command():
    root.destroy()
```

This function is called when the "Exit" button is clicked. It terminates the application by destroying the main Tkinter window.

Function 2: ‘continue_command’

```
def continue_command():
    root.destroy()
    import Login_Button
```

This function is called when the "Continue" button is clicked. It closes the current window and imports the Login_Button module.

File 2 - Login_Button.py

Parent Function 1: 'register'

```
def register():
    ...
```

This function initializes the registration process by creating a new window for user registration and contains sub-functions for saving credentials and updating password strength.

Sub-function 1.1: 'save_credentials'

```
def save_credentials():
    username = username_entry.get()
    password = password_entry.get()

    with open("credentials.csv", "r") as file:
        reader = csv.reader(file)
        for row in reader:
            if row and row[0] == username:
                messagebox.showerror("Registration Error", "Username already exists. Please
choose another.")
                return
    global logged_in_username
    logged_in_username = username

    with open("credentials.csv", "a", newline="") as file:
        writer = csv.writer(file)
        writer.writerow([username, password])

    messagebox.showinfo("Registration Successful", f"Registration successful for user:
{username}")
    register_window.destroy()
    loginpage.destroy()
    import shopping
```

This sub-function saves the user credentials (username and password) to a CSV file after checking if the username already exists. If registration is successful, it displays a confirmation message and closes the registration window.

Sub-function 1.2: ‘update_strength_bar(*args)’

```
def update_strength_bar(*args):
    password = password_var.get()

    if len(password) < 6:
        strength_bar.config(bg="red", text="Weak")
    elif len(password) < 10:
        strength_bar.config(bg="orange", text="Moderate")
    else:
        strength_bar.config(bg="green", text="Strong")
```

This sub-function updates the password strength indicator based on the length of the entered password. It changes the color and text of the strength bar to indicate whether the password is weak, moderate, or strong.

Parent Function 2: ‘login’

```
def login():
```

This function is called when the "Login" button is clicked. It opens a new window for user login and contains the sub-function check_credentials to verify user credentials.

Sub-Function 2.1: ‘check_credentials’

```
def check_credentials():
    entered_username = username_entry.get()
    entered_password = password_entry.get()

    with open("credentials.csv", "r") as file:
        reader = csv.reader(file)
        for row in reader:
            if row and row[0] == entered_username and row[1] == entered_password:
                messagebox.showinfo("Login Successful", "Welcome, {}".format(entered_username))

                global logged_in_username
                logged_in_username = entered_username

                loginpage.destroy()
                import shopping
                return

    messagebox.showerror("Login Failed", "Invalid username or password")
    username_entry.delete(0, END)
    password_entry.delete(0, END)
```

This sub-function retrieves the entered username and password, checks them against the records in the credentials.csv file, and shows a success message if they match. If they do not match, it shows an error message and clears the input fields.

File 3 - shopping.py

Parent Function 1: ‘create_transactions_table’

```
def create_database():
    mycursor.execute("SHOW DATABASES LIKE 'smart_shelf_grocer'")
    result = mycursor.fetchone()
    if not result:
        mycursor.execute("CREATE DATABASE smart_shelf_grocer")
        mycursor.execute("USE smart_shelf_grocer")
    else:
        mycursor.execute("USE smart_shelf_grocer")
```

This function checks if the database "smart_shelf_grocer" exists. If it doesn’t, it creates the database and uses it. Otherwise, it just switches to using it.

Parent Function 2: ‘create_transactions_table’

```
def create_transactions_table():
    mycursor.execute("""
    CREATE TABLE IF NOT EXISTS transactions (
        Trans_ID INT AUTO_INCREMENT ,
        username VARCHAR(255),
        item VARCHAR(255),
        quantity INT,
        total_price DECIMAL(10, 2),
        transaction_date DATE,
        transaction_time TIME,
        PRIMARY KEY(Trans_ID,username)

    )
    """)
    mydb.commit()
```

This function ensures that the transactions table is created in the database. The table will store transaction details, including username, items, quantity, total price, and date/time of the transaction.

Parent Function 3: ‘display_items’

```
def display_items(display_items):
    columns_per_row = 5
    current_row_frame = None

    for idx, item in enumerate(display_items):
        if idx % columns_per_row == 0:
            current_row_frame = ttk.Frame(items_frame)
            current_row_frame.pack(fill='x', pady=5)

        item_frame = create_item_frame(current_row_frame, item)
        item_frame.pack(side='left', padx=10, pady=5)
```

This function displays the available items for sale in rows, with a maximum of 5 items per row. It dynamically creates frames to hold the item information and calls a sub-function to display each item.

Sub-function 3.1: ‘create_item_frame’

```
def create_item_frame(parent, item):
    frame = ttk.Frame(parent)
    item['count'] = tk.IntVar(value=0)

    # Load image
    image = Image.open(item["image"])
    image = image.resize((100, 100), Image.LANCZOS)
    photo = ImageTk.PhotoImage(image)

    label_image = ttk.Label(frame, image=photo)
    label_image.image = photo
    label_image.pack()

    label_title = ttk.Label(frame, text=item["title"], font=("Helvetica", 14))
    label_title.pack()

    label_price = ttk.Label(frame, text=f"₹{item['price']}", font=("Helvetica", 12, "bold"))
    label_price.pack()

    button_increase = ttk.Button(frame, text="+", command=lambda: increase(item))
    button_increase.pack(side="right")

    count_label = ttk.Label(frame, textvariable=item['count'])
    count_label.pack(side="right")

    button_decrease = ttk.Button(frame, text="-", command=lambda: decrease(item))
    button_decrease.pack(side="left")

    return frame
```

This sub-function creates a frame to display an individual item's details (image, title, price, and buttons to increase or decrease quantity).

Parent Function 4: 'update_cart_button'

```
def update_cart_button():
    total_items = sum(item['count'].get() for item in items)
    cart_button.config(text=f"Cart ({total_items})")
```

This function updates the cart button with the total number of items added to the cart.

Parent Function 5: 'increase'

```
def increase(item):
    item['count'].set(item['count'].get() + 1)
    update_cart_button()
```

This function increases the quantity of an item by one each time it is called and updates the cart button.

Parent Function 6: ‘decrease’

```
def decrease(item):
    count = item['count'].get()
    if count > 0:
        item['count'].set(count - 1)
    update_cart_button()
```

This function decreases the quantity of an item by one, ensuring that the count doesn't go below zero, and then updates the cart button.

Parent Function 7: ‘show_cart’

```
def show_cart():
    global root
    root.destroy()

    cart_window = tk.Tk()
    cart_window.title("Cart")
    cart_window.geometry("800x600")

    cart_frame = ttk.Frame(cart_window, padding=20)
    cart_frame.pack(expand=True)

    # Create labels for listboxes
    ttk.Label(cart_frame, text="Items", font=("Helvetica", 14, "bold")).grid(row=0,
column=0, padx=10, pady=5)
    ttk.Label(cart_frame, text="Quantity", font=("Helvetica", 14, "bold")).grid(row=0,
column=1, padx=10, pady=5)
    ttk.Label(cart_frame, text="Price", font=("Helvetica", 14, "bold")).grid(row=0,
column=2, padx=10, pady=5)

    listbox_items = tk.Listbox(cart_frame, font=("Helvetica", 14), height=15, width=20)
    listbox_quantity = tk.Listbox(cart_frame, font=("Helvetica", 14), height=15, width=10)
    listbox_total_price = tk.Listbox(cart_frame, font=("Helvetica", 14), height=15,
width=15)
```

```
    listbox_items.grid(row=1, column=0, padx=10, pady=5)
    listbox_quantity.grid(row=1, column=1, padx=10, pady=5)
    listbox_total_price.grid(row=1, column=2, padx=10, pady=5)

    total_items = sum(item['count'].get() for item in items)
    total_price = sum(item['count'].get() * item['price'] for item in items)

    for item in items:
        if item['count'].get() > 0:
            listbox_items.insert(tk.END, item['title'])
            listbox_quantity.insert(tk.END, item['count'].get())
            listbox_total_price.insert(tk.END, f"₹{item['count'].get() * item['price']}")

    total_items_label = ttk.Label(cart_frame, text=f"Total Items: {total_items}",
font=("Helvetica", 16, "bold"))
    total_items_label.grid(row=2, column=0, columnspan=2, pady=10)

    total_price_label = ttk.Label(cart_frame, text=f"Total Price: ₹{total_price}",
font=("Helvetica", 16, "bold"))
    total_price_label.grid(row=2, column=2, pady=10)

    proceed_button = ttk.Button(cart_frame, text="Proceed to Pay", command=lambda:
proceed_to_pay(cart_window))
    proceed_button.grid(row=3, column=2, sticky='e', pady=20)
```

This function displays the cart with all the items the user has selected, along with their quantities and total price. It also allows the user to proceed to payment.

Parent Function 8: ‘proceed_to_pay’

```
def proceed_to_pay(cart_window):
    cart_window.destroy()

    global logged_in_username

    for item in items:
        if item['count'].get() > 0:
            item_title = item['title']
            item_quantity = item['count'].get()
            total_item_price = item_quantity * item['price']
            transaction_date = datetime.now().date()
            transaction_time = datetime.now().time()

            mycursor.execute(
                "INSERT INTO transactions (username, item, quantity, total_price,
transaction_date, transaction_time) VALUES (%s, %s, %s, %s, %s, %s)",
                (logged_in_username, item_title, item_quantity, total_item_price,
transaction_date, transaction_time)
            )
            mydb.commit()

    def endit():
        thankyou.destroy()
        sys.exit('Program Ended')

    thankyou = tk.Tk()
    thankyou.geometry("830x830")
    thankyou.title("Thank you!")
    thankyou.configure(background='#ADD8E6')

    close_button = Button(thankyou, text="Close", command=endit, height=2, width=10)
    close_button.pack(side="bottom", padx=5, pady=10)

    global img
    global img_tk
    img = Image.open("images/smart shelf groceries.png")
    img = img.resize((600, 800), Image.BICUBIC)
    img_tk = ImageTk.PhotoImage(img)
    panel = Label(thankyou, image=img_tk, bg="#ADD8E6")
    panel.pack(side="top", fill="both", expand="yes")

    thankyou.mainloop()
```

This function processes the payment and inserts the transaction details into the MySQL database. Afterward, it shows a "Thank You" screen and ends the program when the user clicks "Close."

Python Files

File 1: main.py [File to run program]

```
#Importing tkinter
from tkinter import *
from tkinter import ttk
from PIL import Image, ImageTk

#Exit command to exit
def exit_command():
    root.destroy()

#Continue to next page
def continue_command():
    root.destroy()
    import Login_Button
root = Tk()
root.geometry("650x650")
root.title("Smart Shelf Groceries")
root.configure(background='#ADD8E6')

# Load and display the image
img = ImageTk.PhotoImage(Image.open("images/icon.png"))
panel = Label(root, image=img, bg='#ADD8E6')
```

```
panel.pack(side="top", fill="both", expand="yes")

# Create button frame
button_frame = Frame(root, bg='#ADD8E6')
button_frame.pack(side="bottom", fill="x", padx=8, pady=8)

# Continue button
continue_button = ttk.Button(button_frame, text="Continue",
command=continue_command,default="active")
continue_button.pack(side="right", padx=8)

# Exit button
exit_button = ttk.Button(button_frame, text="Exit",
command=exit_command)
exit_button.pack(side="right", padx=8)

root.mainloop()
```

File 2:Login_Button.py

```
#Importing tkinter
from tkinter import *
from tkinter import messagebox
from tkinter import ttk
import csv

loginpage = Tk()
loginpage.geometry("650x650")
loginpage.title("Main Window")
loginpage.configure(background='#ADD8E6')

#Command when user clicks on register
def register():
    def save_credentials():
        username = username_entry.get()
        password = password_entry.get()

        with open("credentials.csv", "r") as file:
            reader = csv.reader(file)
            for row in reader:
                if row and row[0] == username:
                    messagebox.showerror("Registration Error",
"Username already exists. Please choose another.")
```

```
                return
            global logged_in_username
            logged_in_username = username

            with open("credentials.csv", "a", newline="") as file:
                writer = csv.writer(file)
                writer.writerow([username, password])

            messagebox.showinfo("Registration Successful",
"Registration successful for user: {}".format(username))
            register_window.destroy()
            loginpage.destroy()
            import shopping

        def update_strength_bar(*args):
            password = password_var.get()

            if len(password) < 6:
                strength_bar.config(bg="red", text="Weak")
            elif len(password) < 10:
                strength_bar.config(bg="orange", text="Moderate")
            else:
                strength_bar.config(bg="green", text="Strong")

        global register_window
```

```
    register_window = Toplevel(loginpage)
    register_window.configure(background='#ADD8E6')
    register_window.title("Register")
    register_window.geometry("300x250")

    Label(register_window, text="Username:").pack(pady=10)
    username_entry = Entry(register_window)
    username_entry.pack()

    Label(register_window, text="Password:").pack(pady=10)
    password_var = StringVar()
    password_var.trace_add("write", update_strength_bar)
    password_entry = Entry(register_window, show="*",
textvariable=password_var)
    password_entry.pack()

    strength_bar = Label(register_window, text="Password
Strength", bg="gray")
    strength_bar.pack(pady=10)

    save_button = Button(register_window, text="Save",
command=save_credentials, font=("Helvetica", 14))
    save_button.pack(pady=10)
```

```
register_button = Button(loginpage, text="Register",
command=register, font=("Helvetica", 16))
register_button.pack(side=TOP, pady=(200, 10))

or_label = ttk.Label(loginpage, text="OR", font=("Helvetica",
14, "bold"))
or_label.pack(pady=20)

#Command when user clicks on login
def login():
    def check_credentials():
        entered_username = username_entry.get()
        entered_password = password_entry.get()

        with open("credentials.csv", "r") as file:
            reader = csv.reader(file)
            for row in reader:
                if row and row[0] == entered_username and row[1]
== entered_password:
                    messagebox.showinfo("Login Successful",
"Welcome, {}".format(entered_username))

                    global logged_in_username
                    logged_in_username = entered_username
                loginpage.destroy()
                import shopping
                return

        messagebox.showerror("Login Failed", "Invalid username
or password")

        username_entry.delete(0, END)
        password_entry.delete(0, END)

    global login_window
    login_window = Toplevel(loginpage)
    login_window.configure(background='#ADD8E6')
    login_window.title("Login")
    login_window.geometry("300x250")

    Label(login_window, text="Username:").pack(pady=10)
    username_entry = Entry(login_window)
    username_entry.pack()

    Label(login_window, text="Password:").pack(pady=10)
    password_entry = Entry(login_window, show="*")
    password_entry.pack()
    login_button = Button(login_window, text="Login",
command=check_credentials, font=("Helvetica", 16))
    login_button.pack(pady=10)

login_button = Button(loginpage, text="Login", command=login,
font=("Helvetica", 16))
login_button.pack(side=TOP, pady=(10, 200))

loginpage.mainloop()
```

File 3: shopping.py

```
import tkinter as tk
from tkinter import ttk, Button, Label
from PIL import Image, ImageTk
import sys

import mysql.connector
from datetime import datetime

from Login_Button import logged_in_username


mydb = mysql.connector.connect(
    host="localhost",       # Your MySQL host (usually localhost)
    user="root",            # Your MySQL username
    password="root"
)

mycursor = mydb.cursor()

# Function to create the database if it doesn't exist
def create_database():
    mycursor.execute("SHOW DATABASES LIKE
'smart_shelf_grocer'")
```

```
    result = mycursor.fetchone()
    if not result:
        mycursor.execute("CREATE DATABASE
smart_shelf_grocer")
        mycursor.execute("USE smart_shelf_grocer")
    else:
        mycursor.execute("USE smart_shelf_grocer")

# Call the function to check and create the database
create_database()


# Function to create the transactions table if it doesn't exist
def create_transactions_table():
    mycursor.execute("""
    CREATE TABLE IF NOT EXISTS transactions (
        Trans_ID INT AUTO_INCREMENT,
        username VARCHAR(255),
        item VARCHAR(255),
        quantity INT,
        total_price DECIMAL(10, 2),
        transaction_date DATE,
        transaction_time TIME,
       PRIMARY KEY(Trans_ID,username)
    )
    """)

    mydb.commit()

# Call the function to ensure table is created
create_transactions_table()



items = [
    {"title": "Apple", "price": 10, "image": r"images/apple.jpeg"},
    {"title": "Banana", "price": 20, "image":
r"images/banana.jpeg"},
    {"title": "Broccoli", "price": 30, "image":
r"images/broccoli.jpeg"},
    {"title": "Carrots", "price": 40, "image":
r"images/carrot.jpeg"},
    {"title": "Cucumbers", "price": 50, "image":
r"images/cucumber.jpeg"},
    {"title": "Grapes", "price": 60, "image":
r"images/grapes.jpeg"},
    {"title": "Orange", "price": 70, "image":
r"images/orange.jpeg"},
    {"title": "Pineapple", "price": 80, "image":
r"images/pineapple.jpeg"},
    {"title": "Strawberry", "price": 90, "image":
r"images/strawberry.jpeg"},
    {"title": "Tomato", "price": 100, "image":
r"images/tomato.jpeg"}
]


def create_item_frame(parent, item):
    frame = ttk.Frame(parent)
    item['count'] = tk.IntVar(value=0)

    # Load image
    image = Image.open(item["image"])
    image = image.resize((100, 100), Image.LANCZOS)
    photo = ImageTk.PhotoImage(image)

    label_image = ttk.Label(frame, image=photo)
    label_image.image = photo
    label_image.pack()

    label_title = ttk.Label(frame, text=item["title"],
font=("Helvetica", 14))
    label_title.pack()
    label_price = ttk.Label(frame, text=f"₹{item['price']}",
font=("Helvetica", 12, "bold"))
    label_price.pack()

    button_increase = ttk.Button(frame, text="+",
command=lambda: increase(item))
    button_increase.pack(side="right")

    count_label = ttk.Label(frame, textvariable=item['count'])
    count_label.pack(side="right")

    button_decrease = ttk.Button(frame, text="-",
command=lambda: decrease(item))
    button_decrease.pack(side="left")

    return frame

def increase(item):
    item['count'].set(item['count'].get() + 1)
    update_cart_button()

def decrease(item):
    count = item['count'].get()
    if count > 0:
        item['count'].set(count - 1)
    update_cart_button()

def update_cart_button():
    total_items = sum(item['count'].get() for item in items)
    cart_button.config(text=f"Cart ({total_items})")

def show_cart():
    global root
    root.destroy()  # This will close the main window

    cart_window = tk.Tk()
    cart_window.title("Cart")
    cart_window.geometry("800x600")

    cart_frame = ttk.Frame(cart_window, padding=20)
    cart_frame.pack(expand=True)

    # Create labels for listboxes
    ttk.Label(cart_frame, text="Items", font=("Helvetica", 14,
"bold")).grid(row=0, column=0, padx=10, pady=5)
    ttk.Label(cart_frame, text="Quantity", font=("Helvetica",
14, "bold")).grid(row=0, column=1, padx=10, pady=5)
    ttk.Label(cart_frame, text="Price", font=("Helvetica", 14,
"bold")).grid(row=0, column=2, padx=10, pady=5)
```

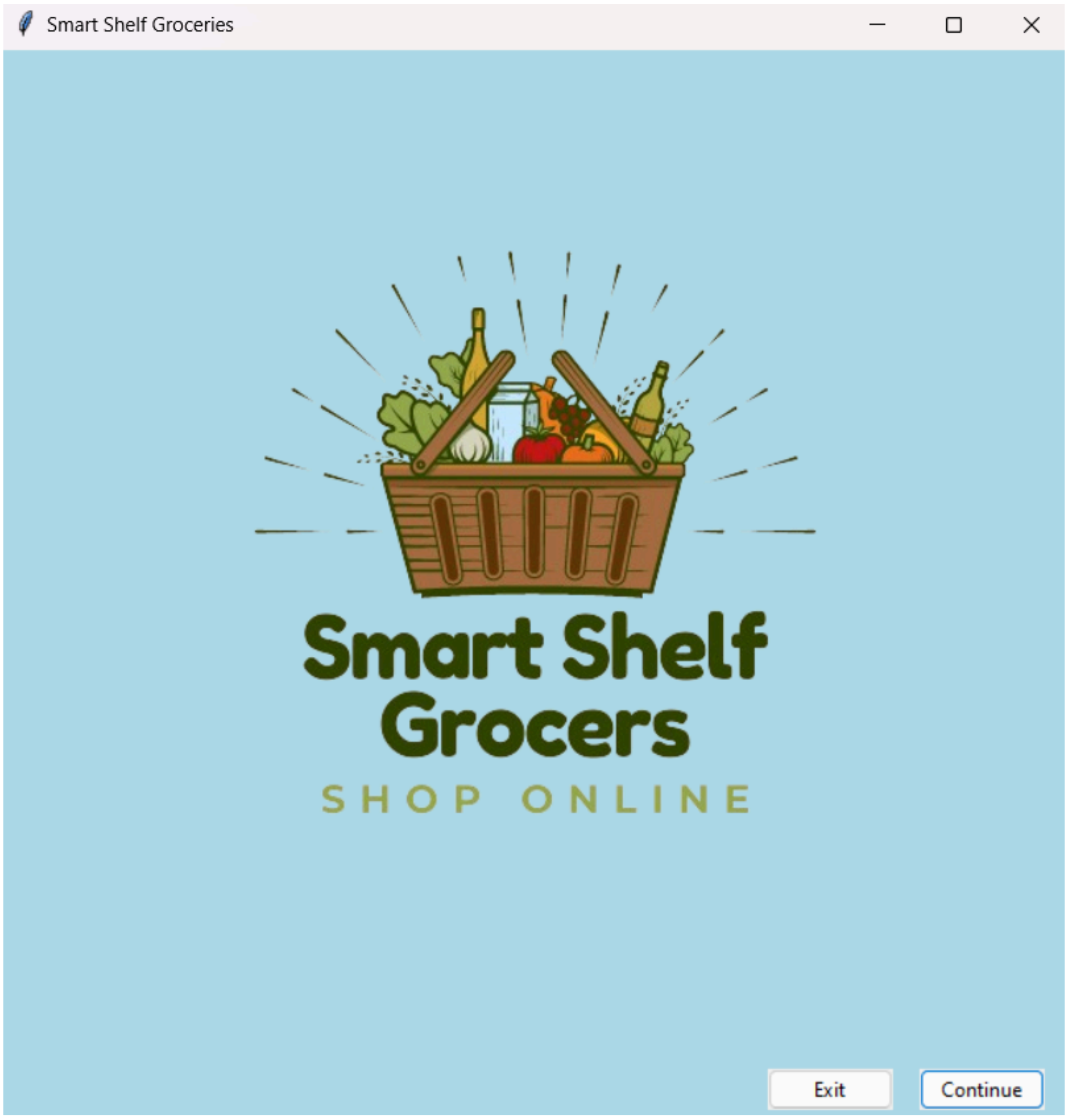
```
    listbox_items = tk.Listbox(cart_frame, font=("Helvetica",
14), height=15, width=20)
    listbox_quantity = tk.Listbox(cart_frame,
font=("Helvetica", 14), height=15, width=10)
    listbox_total_price = tk.Listbox(cart_frame,
font=("Helvetica", 14), height=15, width=15)

    listbox_items.grid(row=1, column=0, padx=10, pady=5)
    listbox_quantity.grid(row=1, column=1, padx=10, pady=5)
    listbox_total_price.grid(row=1, column=2, padx=10, pady=5)

    total_items = sum(item['count'].get() for item in items)
    total_price = sum(item['count'].get() * item['price'] for
item in items)

    for item in items:
        if item['count'].get() > 0:
            listbox_items.insert(tk.END, item['title'])
            listbox_quantity.insert(tk.END, item['count'].get())
            listbox_total_price.insert(tk.END,
f"₹{item['count'].get() * item['price']}")

    total_items_label = ttk.Label(cart_frame, text=f"Total
Items: {total_items}", font=("Helvetica", 16, "bold"))
```

```
    total_items_label.grid(row=2, column=0, columnspan=2,
pady=10)

    total_price_label = ttk.Label(cart_frame, text=f"Total
Price: ₹{total_price}", font=("Helvetica", 16, "bold"))
    total_price_label.grid(row=2, column=2, pady=10)

    proceed_button = ttk.Button(cart_frame, text="Proceed to
Pay", command=lambda: proceed_to_pay(cart_window))
    proceed_button.grid(row=3, column=2, sticky='e', pady=20)

def proceed_to_pay(cart_window):
    cart_window.destroy()

    # Assuming `logged_in_username` is passed or globally
accessible
    global logged_in_username

    # Insert the transaction details into the database
    for item in items:
        if item['count'].get() > 0:
            item_title = item['title']
            item_quantity = item['count'].get()
            total_item_price = item_quantity * item['price']
            transaction_date = datetime.now().date()
            transaction_time = datetime.now().time()

            # Insert transaction data into the MySQL database

            mycursor.execute(
                "INSERT INTO transactions (username, item,
quantity, total_price, transaction_date, transaction_time)
VALUES (%s, %s, %s, %s, %s, %s)",
                (logged_in_username, item_title, item_quantity,
total_item_price, transaction_date, transaction_time)
            )
            mydb.commit()

    def endit():
        thankyou.destroy()
        sys.exit('Program Ended')

    # Create the main Tkinter window for the "Thank You"
screen
    thankyou = tk.Tk()
    thankyou.geometry("830x830")
    thankyou.title("Thank you!")
    thankyou.configure(background='#ADD8E6')

    # Close button to end the program
    close_button = Button(thankyou, text="Close",
command=endit, height=2, width=10)
    close_button.pack(side="bottom", padx=5, pady=10)

    # Display a "Thank You" image
    global img
    global img_tk
    img = Image.open("images/smart shelf groceries.png")
    img = img.resize((600, 800), Image.BICUBIC)
    img_tk = ImageTk.PhotoImage(img)
    panel = Label(thankyou, image=img_tk, bg="#ADD8E6")
    panel.pack(side="top", fill="both", expand="yes")

    thankyou.mainloop()

def search_items(*args):
    search_query = search_var.get().lower()
    for widget in items_frame.winfo_children():
        widget.destroy()

    filtered_items = [item for item in items if search_query in
item["title"].lower()]
    display_items(filtered_items)

def display_items(display_items):
    columns_per_row = 5
    current_row_frame = None

    for idx, item in enumerate(display_items):
        if idx % columns_per_row == 0:
            current_row_frame = ttk.Frame(items_frame)
            current_row_frame.pack(fill='x', pady=5)

        item_frame = create_item_frame(current_row_frame,
item)
        item_frame.pack(side='left', padx=10, pady=5)

root = tk.Tk()
root.title("Shopping Window")
root.geometry("1200x1000")  # Change this line to set the
default size

main_frame = ttk.Frame(root, padding="20")
main_frame.pack(fill="both", expand=True)

title_label = ttk.Label(main_frame, text="Items for Sale",
font=("Helvetica", 18))
title_label.pack(pady=10)

search_cart_frame = ttk.Frame(main_frame)
```

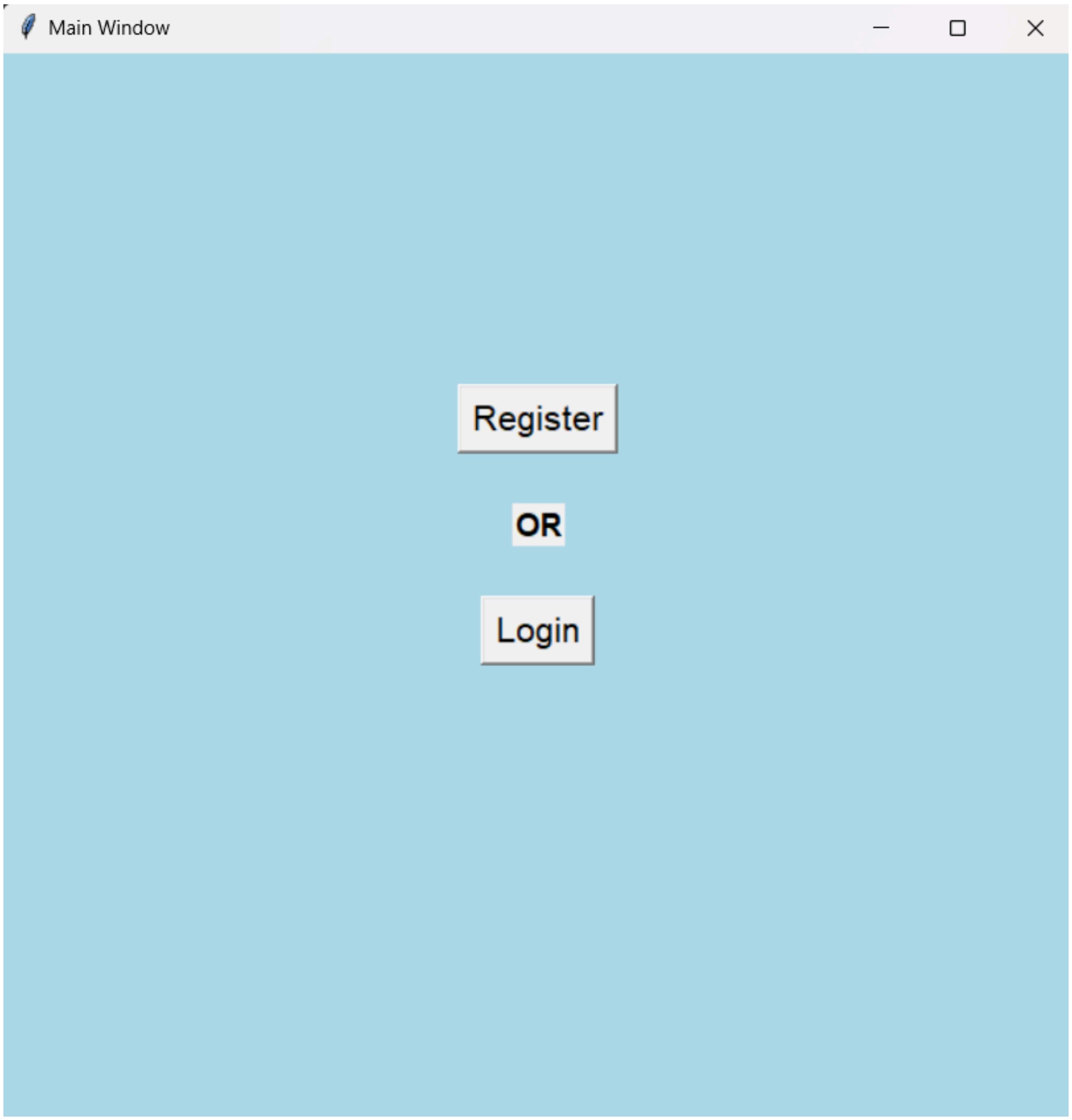
```
search_cart_frame.pack(pady=10)

search_var = tk.StringVar()
search_var.trace_add("write", search_items)

search_bar = ttk.Entry(search_cart_frame,
textvariable=search_var, width=50)
search_bar.pack(side="left", padx=(0, 10))

cart_button = ttk.Button(search_cart_frame, text="Cart (0)",
command=show_cart)
cart_button.pack(side="left")

canvas = tk.Canvas(main_frame)
canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

items_frame = ttk.Frame(canvas)
canvas.create_window((0, 0), window=items_frame,
anchor="nw")

display_items(items)

root.mainloop()
```

Output Screens

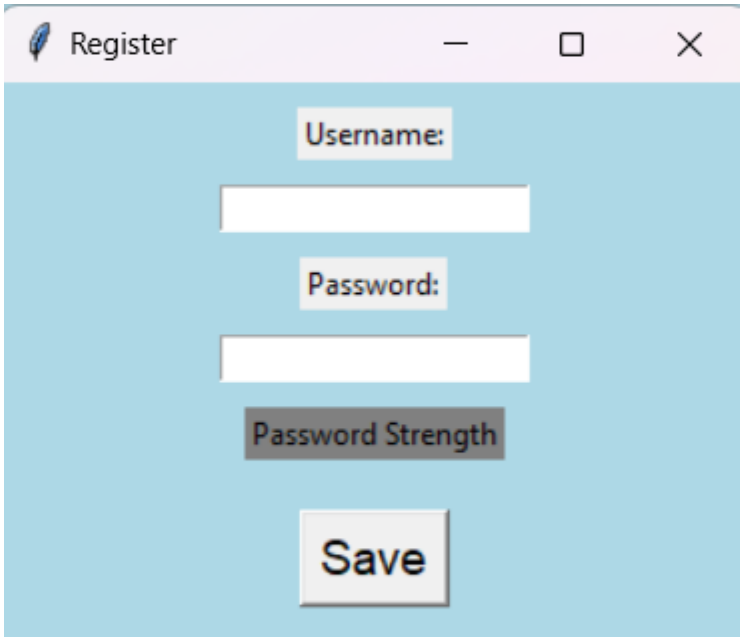
Screen 1: Displays our logo, along with a continue button to move forward and exit button to exit.



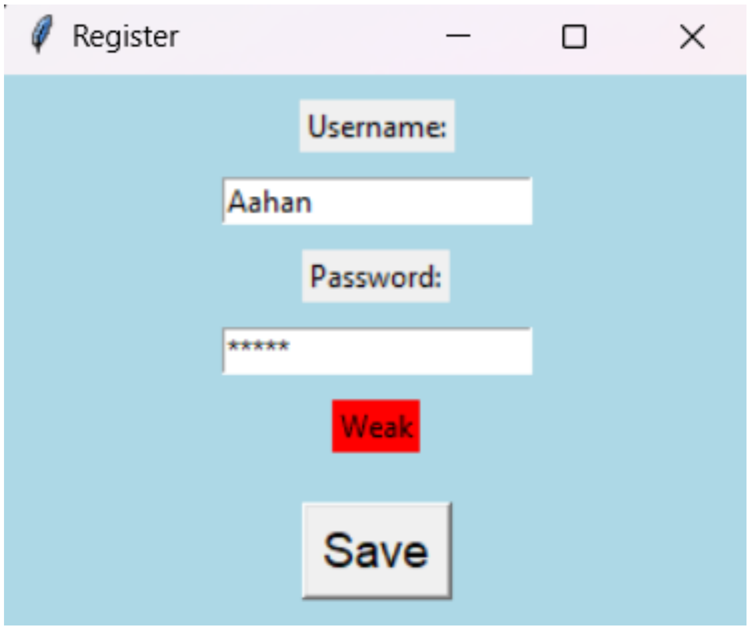
Screen 2: Displays two buttons, register and login.



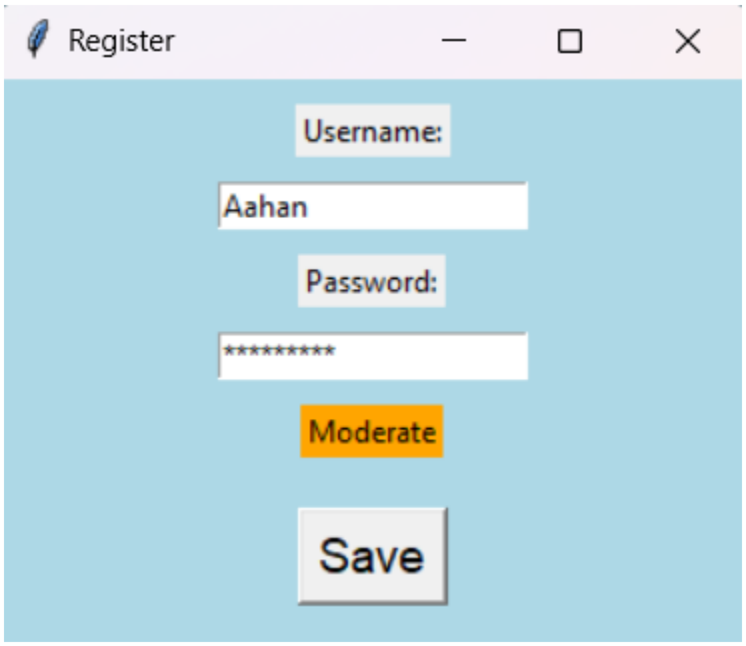
Screen 3: If user clicks on register, the following window pops up:



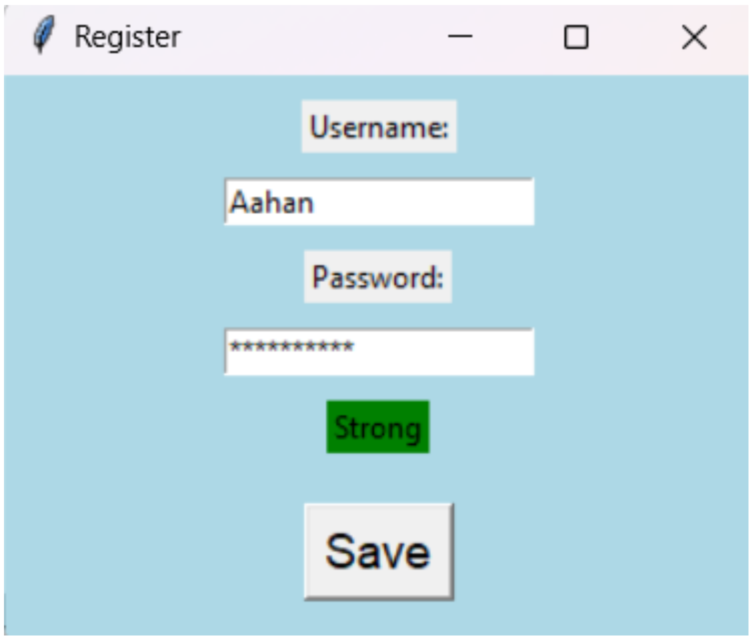
Password strength bar turns red and displays 'Weak' if password is less than 6 characters:



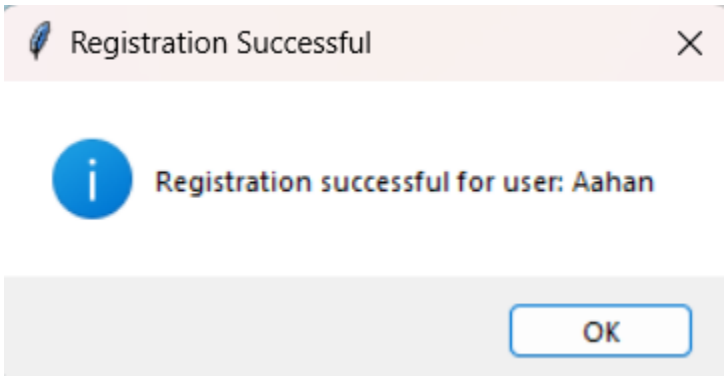
Password strength bar turns orange and displays 'Moderate' if password is less than 9 characters:



Password strength bar turns green and displays 'Moderate' if password is more than 9 characters:

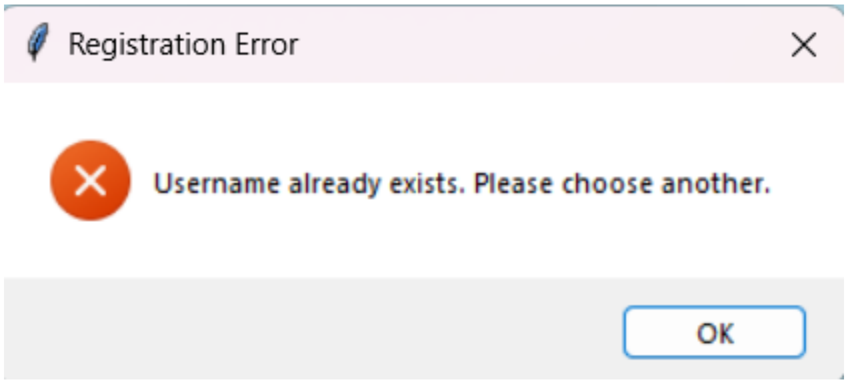


Screen 4: When the user clicks on save, a message box is displayed and screen 9 is displayed.



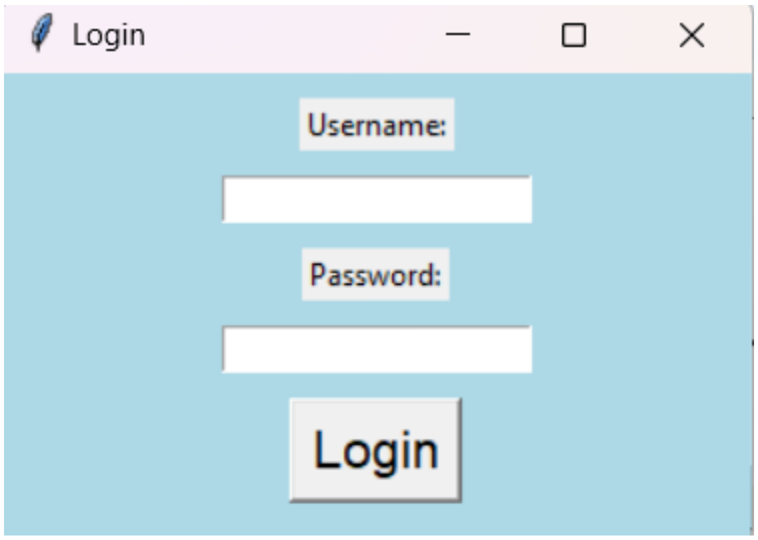
Registration successful for user: Aahan

Screen 5: When the user tries to register with a username already present

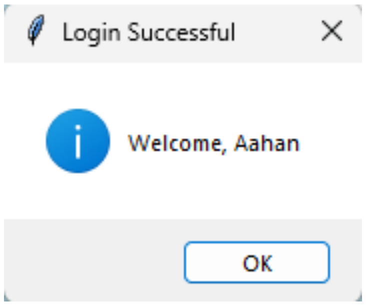


Username already exists. Please choose another.

Screen 6: If the user clicks on login, the following window pops up:

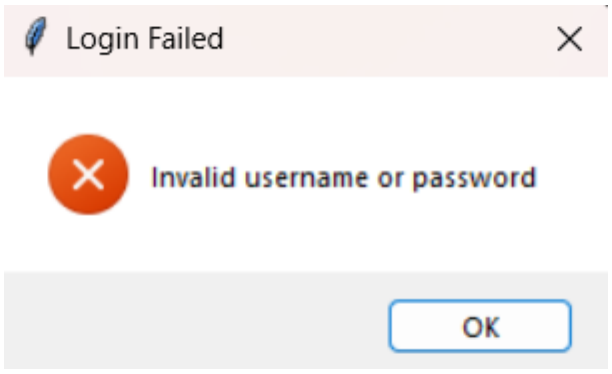


Screen 7: If a user logs in successfully:



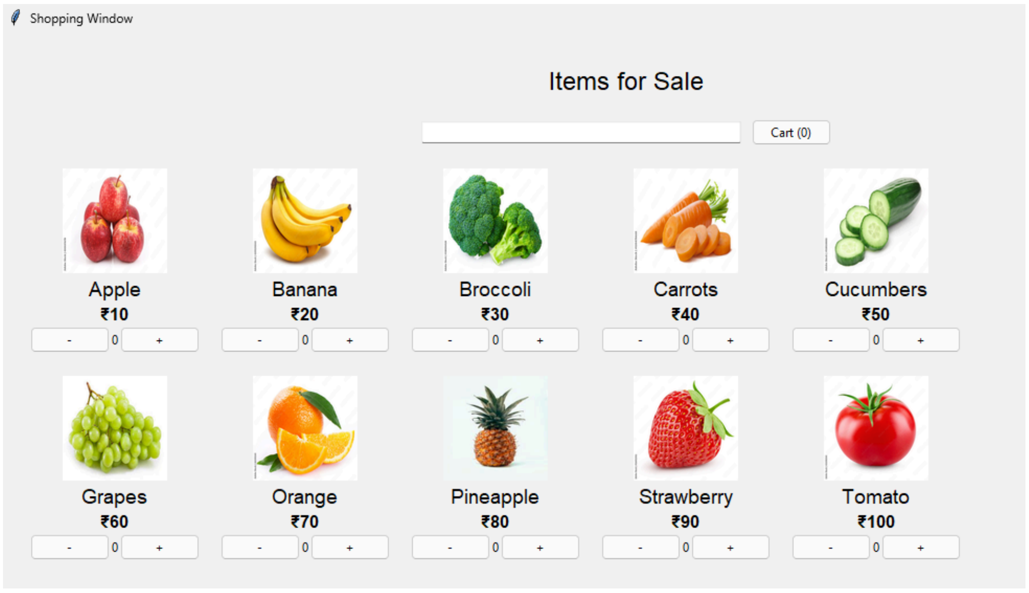
Welcome, Aahan

Screen 8: If a user enters a username or password that is incorrect:

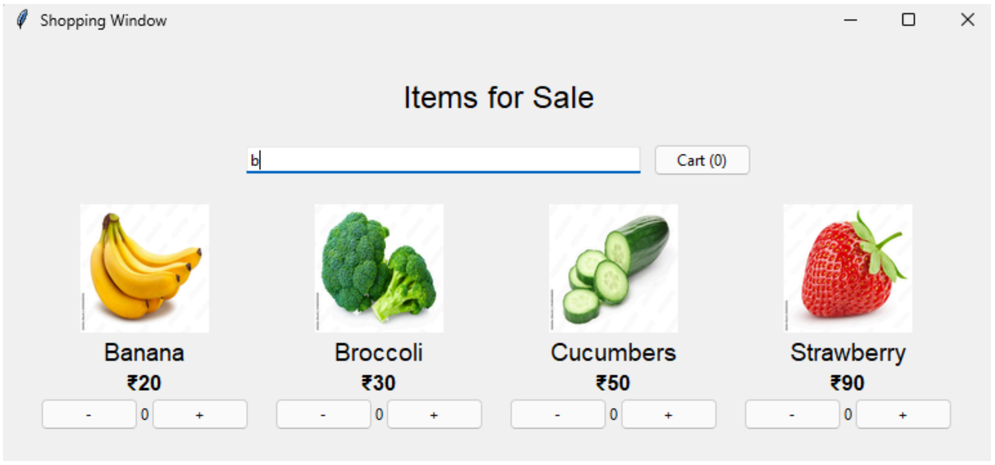


Invalid username or password

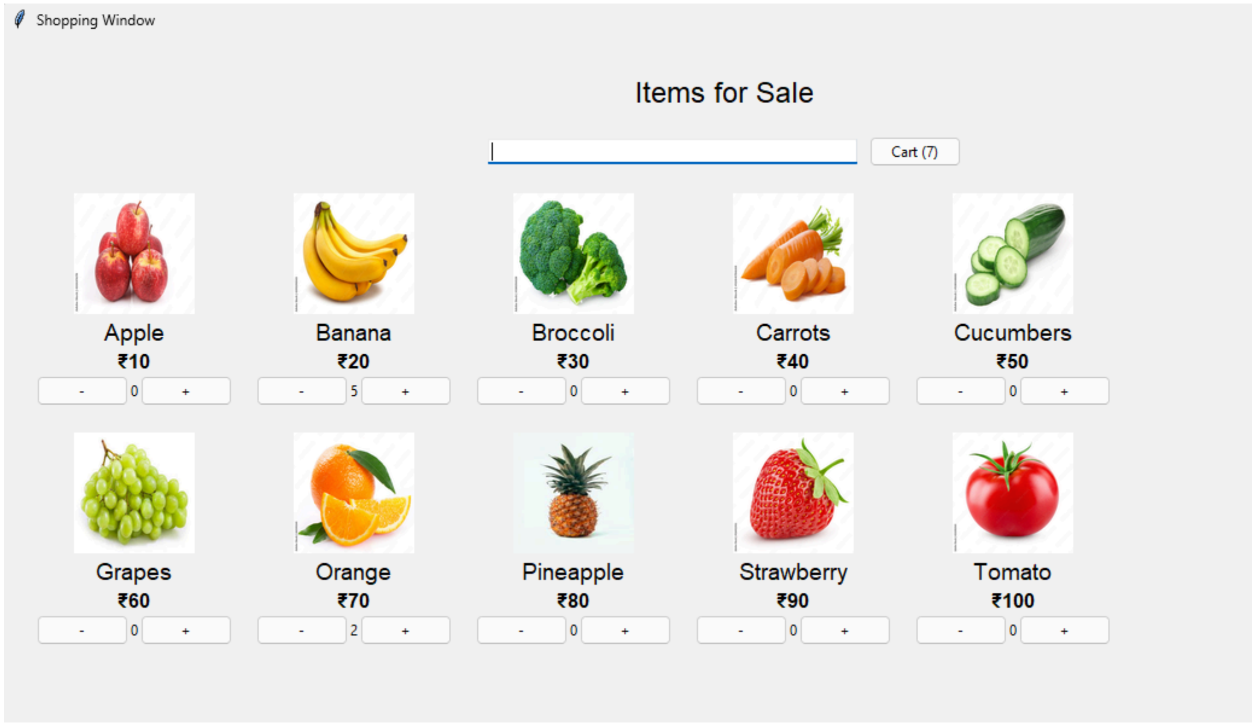
Screen 9: Shopping Window



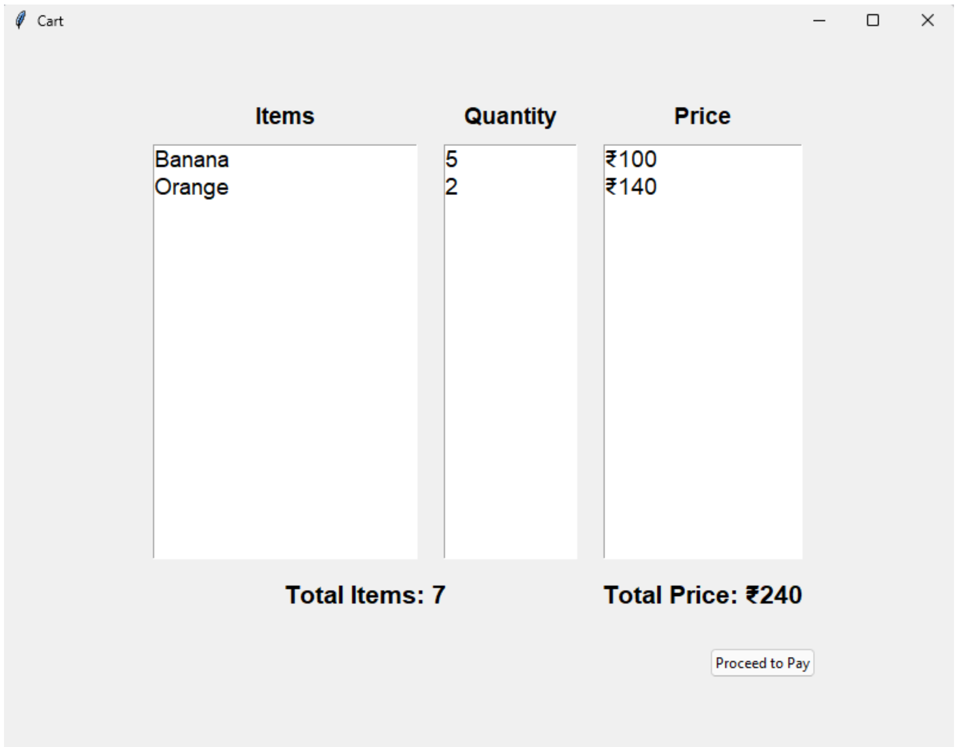
To use the following window, click on the '+' or '-' buttons to increase or decrease the quantity of the required item respectively. The number of items will be simultaneously updated in the cart button near the search bar. The search bar on top is automatic, which changes the screen to display only those items which contain the letter typed. For example, if you type 'b', all the items which contain 'b' in them will be displayed.



Screen 10: After selecting the items (5 Bananas and 2 Oranges), the Cart button dynamically gets updated with the total number of items selected (7 in this case).



Screen 11: After user clicks on the cart button

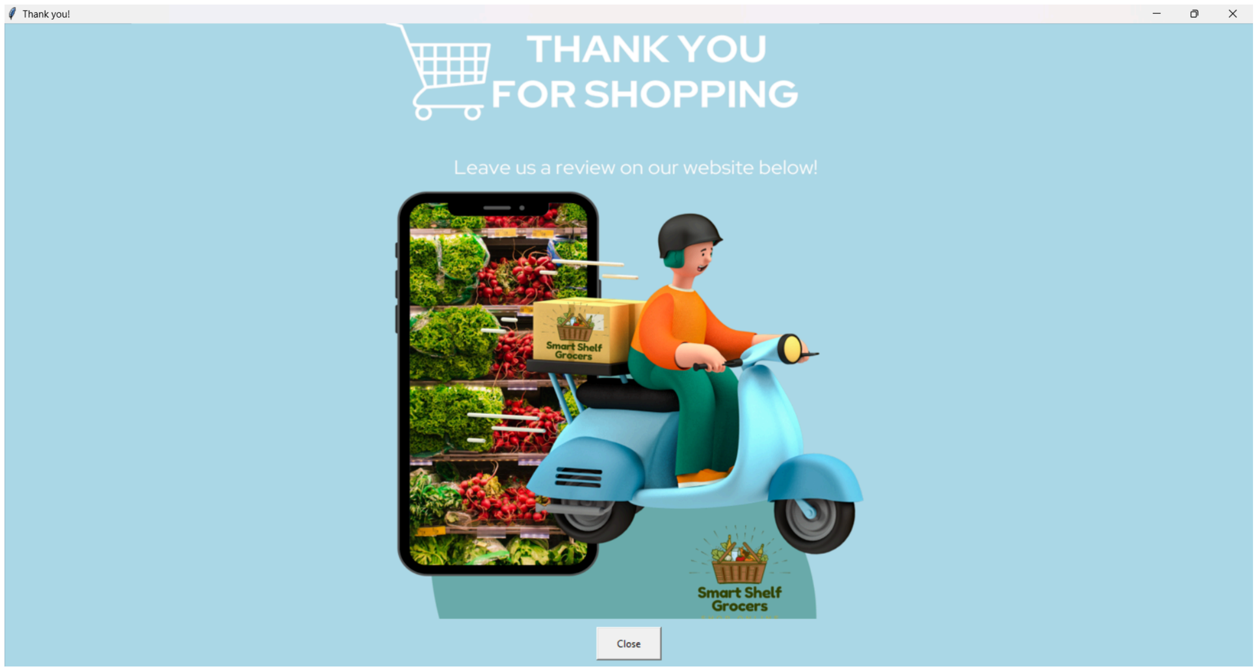


Items	Quantity	Price
Banana	5	₹100
Orange	2	₹140

Total Items: 7

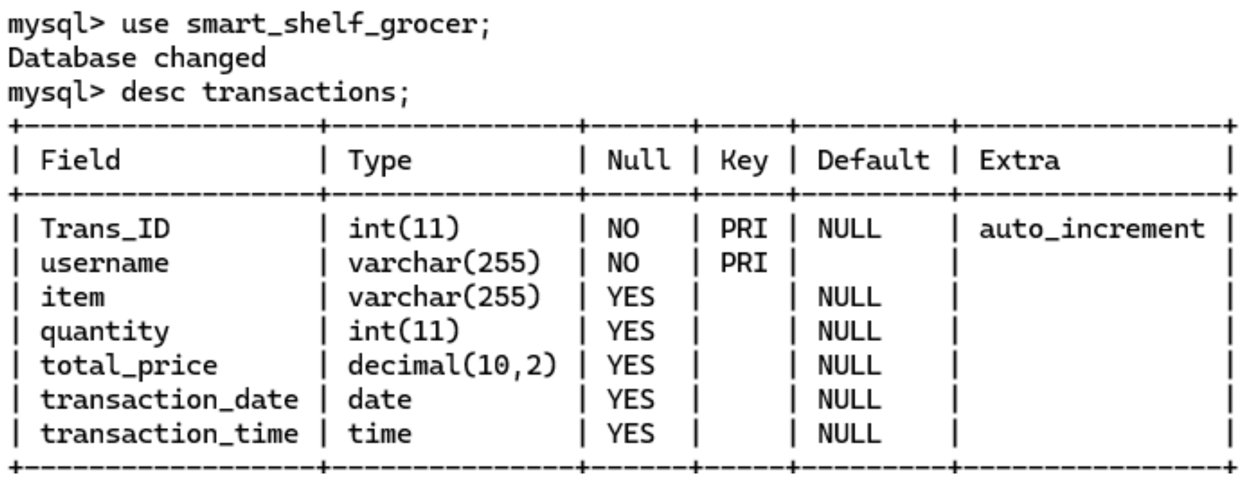
Total Price: ₹240

Screen 12: Thank you page after user clicks on proceed to pay



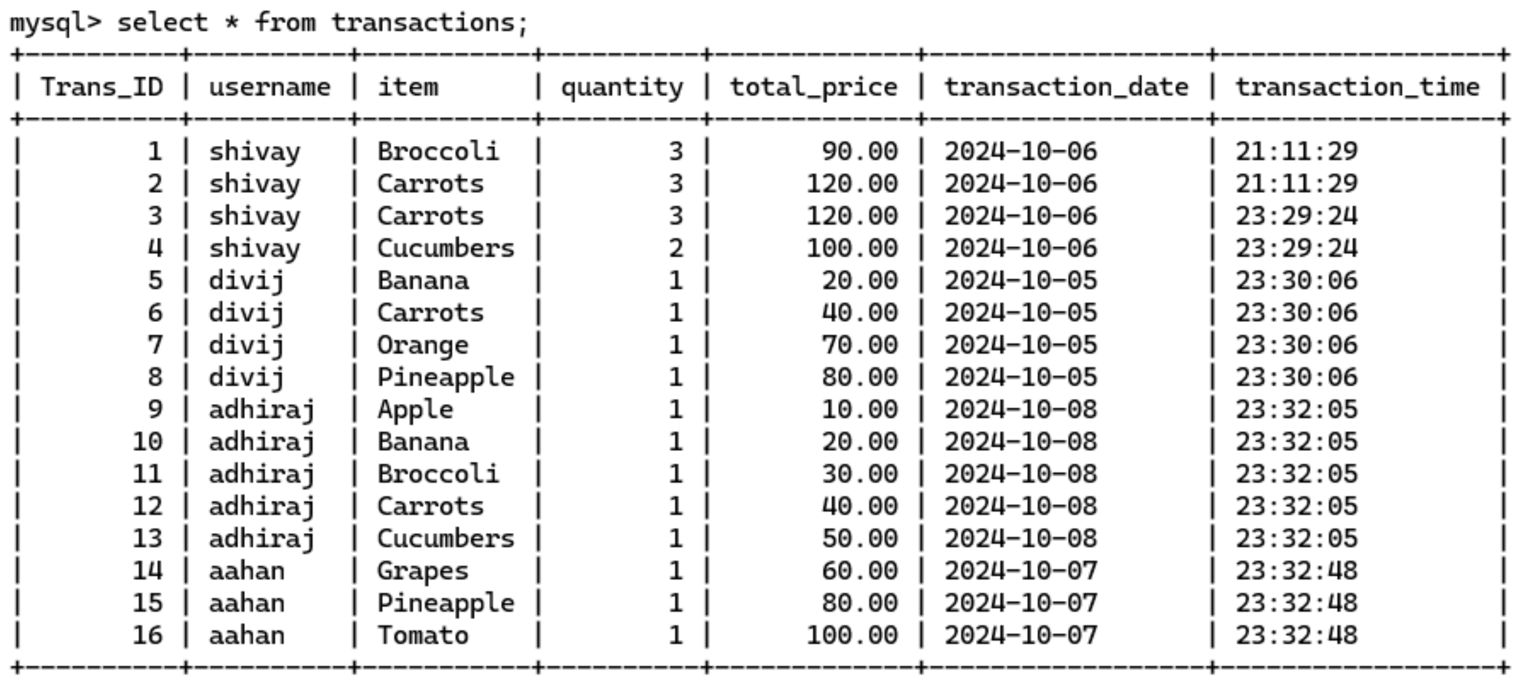
MySQL-CSV Files Integration

1. Structure of Transactions table



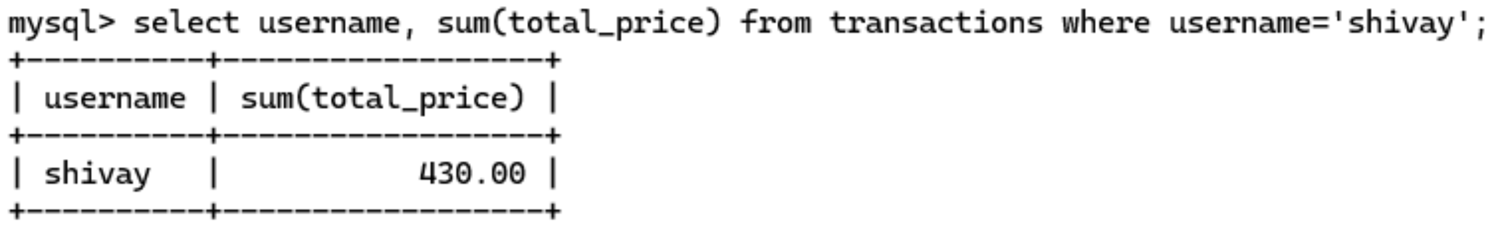
```
mysql> use smart_shelf_grocer;
Database changed
mysql> desc transactions;
+------------------+---------------+------+-----+---------+----------------+
| Field            | Type          | Null | Key | Default | Extra          |
+------------------+---------------+------+-----+---------+----------------+
| Trans_ID         | int(11)       | NO   | PRI | NULL    | auto_increment |
| username         | varchar(255)  | NO   | PRI |         |                |
| item             | varchar(255)  | YES  |     | NULL    |                |
| quantity         | int(11)       | YES  |     | NULL    |                |
| total_price      | decimal(10,2) | YES  |     | NULL    |                |
| transaction_date | date          | YES  |     | NULL    |                |
| transaction_time | time          | YES  |     | NULL    |                |
+------------------+---------------+------+-----+---------+----------------+
```

2. Viewing transactions logged in table



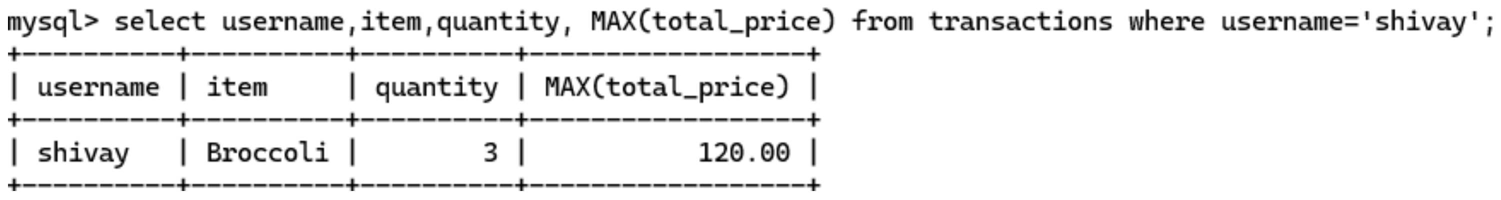
```
mysql> select * from transactions;
+----------+----------+-----------+----------+-------------+------------------+------------------+
| Trans_ID | username | item      | quantity | total_price | transaction_date | transaction_time |
+----------+----------+-----------+----------+-------------+------------------+------------------+
|        1 | shivay   | Broccoli  |        3 |       90.00 | 2024-10-06       | 21:11:29         |
|        2 | shivay   | Carrots   |        3 |      120.00 | 2024-10-06       | 21:11:29         |
|        3 | shivay   | Carrots   |        3 |      120.00 | 2024-10-06       | 23:29:24         |
|        4 | shivay   | Cucumbers |        2 |      100.00 | 2024-10-06       | 23:29:24         |
|        5 | divij    | Banana    |        1 |       20.00 | 2024-10-05       | 23:30:06         |
|        6 | divij    | Carrots   |        1 |       40.00 | 2024-10-05       | 23:30:06         |
|        7 | divij    | Orange    |        1 |       70.00 | 2024-10-05       | 23:30:06         |
|        8 | divij    | Pineapple |        1 |       80.00 | 2024-10-05       | 23:30:06         |
|        9 | adhiraj  | Apple     |        1 |       10.00 | 2024-10-08       | 23:32:05         |
|       10 | adhiraj  | Banana    |        1 |       20.00 | 2024-10-08       | 23:32:05         |
|       11 | adhiraj  | Broccoli  |        1 |       30.00 | 2024-10-08       | 23:32:05         |
|       12 | adhiraj  | Carrots   |        1 |       40.00 | 2024-10-08       | 23:32:05         |
|       13 | adhiraj  | Cucumbers |        1 |       50.00 | 2024-10-08       | 23:32:05         |
|       14 | aahan    | Grapes    |        1 |       60.00 | 2024-10-07       | 23:32:48         |
|       15 | aahan    | Pineapple |        1 |       80.00 | 2024-10-07       | 23:32:48         |
|       16 | aahan    | Tomato    |        1 |      100.00 | 2024-10-07       | 23:32:48         |
+----------+----------+-----------+----------+-------------+------------------+------------------+
```

3. Finding the sum of all purchases made by user ‘shivay’



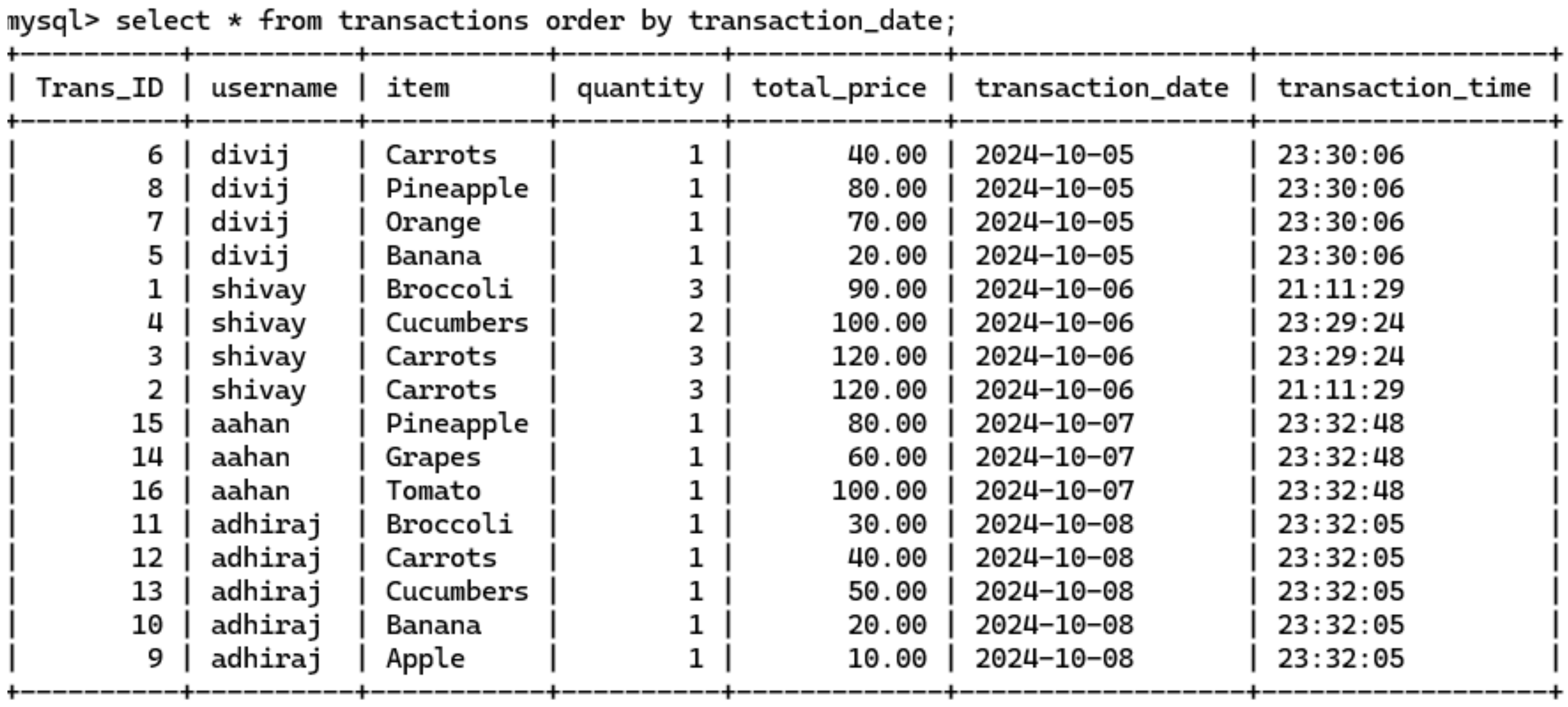
```
mysql> select username, sum(total_price) from transactions where username='shivay';
+----------+------------------+
| username | sum(total_price) |
+----------+------------------+
| shivay   |           430.00 |
+----------+------------------+
```

4. Finding the max total price of an item purchased by user



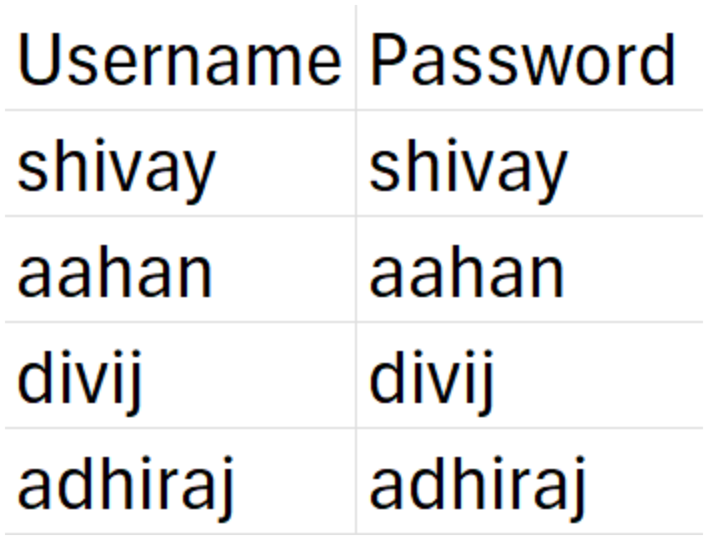
```
mysql> select username,item,quantity, MAX(total_price) from transactions where username='shivay';
+----------+----------+----------+------------------+
| username | item     | quantity | MAX(total_price) |
+----------+----------+----------+------------------+
| shivay   | Broccoli |        3 |           120.00 |
+----------+----------+----------+------------------+
```

5. Ordering the transactions by transaction date



```
mysql> select * from transactions order by transaction_date;
+----------+----------+-----------+----------+-------------+------------------+------------------+
| Trans_ID | username | item      | quantity | total_price | transaction_date | transaction_time |
+----------+----------+-----------+----------+-------------+------------------+------------------+
|        6 | divij    | Carrots   |        1 |       40.00 | 2024-10-05       | 23:30:06         |
|        8 | divij    | Pineapple |        1 |       80.00 | 2024-10-05       | 23:30:06         |
|        7 | divij    | Orange    |        1 |       70.00 | 2024-10-05       | 23:30:06         |
|        5 | divij    | Banana    |        1 |       20.00 | 2024-10-05       | 23:30:06         |
|        1 | shivay   | Broccoli  |        3 |       90.00 | 2024-10-06       | 21:11:29         |
|        4 | shivay   | Cucumbers |        2 |      100.00 | 2024-10-06       | 23:29:24         |
|        3 | shivay   | Carrots   |        3 |      120.00 | 2024-10-06       | 23:29:24         |
|        2 | shivay   | Carrots   |        3 |      120.00 | 2024-10-06       | 21:11:29         |
|       15 | aahan    | Pineapple |        1 |       80.00 | 2024-10-07       | 23:32:48         |
|       14 | aahan    | Grapes    |        1 |       60.00 | 2024-10-07       | 23:32:48         |
|       16 | aahan    | Tomato    |        1 |      100.00 | 2024-10-07       | 23:32:48         |
|       11 | adhiraj  | Broccoli  |        1 |       30.00 | 2024-10-08       | 23:32:05         |
|       12 | adhiraj  | Carrots   |        1 |       40.00 | 2024-10-08       | 23:32:05         |
|       13 | adhiraj  | Cucumbers |        1 |       50.00 | 2024-10-08       | 23:32:05         |
|       10 | adhiraj  | Banana    |        1 |       20.00 | 2024-10-08       | 23:32:05         |
|        9 | adhiraj  | Apple     |        1 |       10.00 | 2024-10-08       | 23:32:05         |
+----------+----------+-----------+----------+-------------+------------------+------------------+
```

In the CSV file ‘credentials.csv’-



Username	Password
shivay	shivay
aahan	aahan
divij	divij
adhiraj	adhiraj

Limitations and Future Scope

The future scope of this project is immense. There are many features that we still need to add, due to reasons such as lack of time and inexperience in programming in Tkinter. Below we have listed down the limitations faced while doing this project along with its potential in the future: -

1. Adding a feature for adding payment options is also an idea.
2. We would like to add an update feature, which gives the user an option to change their username or password.
3. Currently, the search bar can only handle simple queries. It could be improved to filter products by prices etc.
4. While transactions are logged, the system does not track inventory levels, which may lead to issues if items go out of stock.

Bibliography and Link to Project

1. https://www.geeksforgeeks.org/python-tkinter-tutorial/
2. https://www.w3schools.in/python/gui-programming
3. https://www.pythontutorial.net/tkinter/
4. https://www.javatpoint.com/python-tkinter
5. https://www.tutorialspoint.com/python/python_gui_programming.htm/
6. https://docs.python.org/3/library/index.html
7. https://www.youtube.com/@Codemycom
8. https://www.canva.com/
9. https://srikakulamads.com/what-is-online-grocery-shopping/
10. https://www.pexels.com/

Link to Project:

https://drive.google.com/drive/folders/1Cfp7ZQ8n5sGftSTrZf4eSN3vB1aSUD6P?usp=sharing