Central Michigan University

BIS 698

*Information Systems Project*



*Inventory Management System*

*BIS 698 Information Systems Project 22458798 (Hybrid)*

*(Spring 2024)*

*Dr.. Javad Norourzi Nia*

Submission Date: 04/22/2024

**Group 3**

Subham Tiwari

Utsav Jaiswal

Taran Sai Uppalapati

Sai Abhishek Reddy Agasanur

Devi Anish Korada

Chaitanya Goud Nayanoola

# Contents

* Introduction
* System Overview
* System Architecture
* Features and Functionalities
* User Interface
* Database Design
* System Process
* Code Snippets

1. **Introduction**

The IMS will feature a user-friendly interface accessible to the stationary store owner, employees, and floor manager. Key functionalities include Different user roles (e.g., Floor Managers, employee) may have different levels of access to system functionalities. Changes made to system data are logged and tracked for auditing purposes. Employees can customize settings and preferences according to their requirements. The system provides reports, and these reports can be exported to excel for monitoring business performance and making informed decisions. The system will help the store with the managing their processes, ensuring accuracy and operational efficiency.

1. **System Overview:**

Utilizing Python for the backend, the project ensures efficient development and seamless integration. The frontend will be crafted with tkinter , providing a user-friendly interface. MySQL, a flexible NoSQL database, will manage data, ensuring a powerful and scalable solution for the Inventory Management System.

1. **System Architecture:**

A diagram of a company

Description automatically generated

1. **Features and Functionalities:**

Authentication Process - Features:

1. Secure Access: Employees are required to provide their registered email address and password to access the system.

2. Verification: The system verifies the provided credentials against the stored user database to ensure authenticity.

3. Error Handling: Failed authentication attempts trigger appropriate error messages, maintaining system security and preventing unauthorized access.

Main Menu Navigation - Features:

1. Centralized Hub: The Main Menu page serves as the central hub for accessing various system functionalities.

2. Intuitive Navigation: Employees can easily navigate between different modules, such as Inventory Management, Vendor Management, Customer Sales, and Password Update.

3. Structured Interface: Each menu option corresponds to specific functionalities within the system, providing a structured and user-friendly interface for efficient operation.

Inventory Management Process-Features:

1. Product Management: Employees can save, add, update, delete, select, find, clear for products within the inventory.

2. Stock Level Monitoring: Employees can make use of Export to excel feature to check the stock report more thoroughly

3. Data Integrity: Changes made to inventory records are logged and tracked to ensure data integrity and accountability.

Vendor Management Process- Features:

1. Vendor Information: Employees can access detailed information about vendors, including contact details and order history.

2. Order Placement: Employees can place orders for inventory items, look up products by their names and specify quantities and assigning vendors for procurement

Customer Sales Process - Features:

1. Product Selection: Employees can search for products, select items for purchase, and specify quantities.

2. Shopping Cart: Selected items are added to the shopping cart, where Employees can review and modify their order before checkout.

3. Order Processing: The system calculates total prices, applies discounts or promotions, and generates invoices for customers.

Overall System Features:

1. Role-Based Access Control: Different user roles (e.g., Floor Managers, employee) may have different levels of access to system functionalities.

2. Data Logging and Tracking: Changes made to system data are logged and tracked for auditing purposes.

3. Customizable Settings: Employees can customize settings and preferences according to their requirements.

4. Reporting: The system provides reports and these reports can be exported to excel for monitoring business performance and making informed decisions.

1. **USER INTERFACE:**

***Employee Login Page:***

The Employee Login page serves as the entry point for authorized personnel to access the inventory management system. It features fields for entering the registered email address and password associated with each employee's account.

A screenshot of a computer

Description automatically generated

After the employee enters the right details, the prompt appears to make sure the **database connection is established.**

A screenshot of a computer

Description automatically generated

***Main Menu Page:***

Upon successful authentication, Employees are directed to the Main Menu page. This centralized hub provides navigation options to key functionalities of the system, including Inventory Management, Vendor Management, Customer Sales, and Password Update.

A screenshot of a computer screen

Description automatically generated

***Vendor Page:***

The Vendor Page offers essential functionalities related to vendor management. Employees can access detailed vendor information and place orders for inventory replenishment.

A screenshot of a computer

Description automatically generated

**Vendor Details:** Displays comprehensive information about registered vendors, including unique Vendor ID, Vendor Name, and the contact person associated with each vendor. A "Back to Main Menu" button facilitates easy navigation.

A screenshot of a computer

Description automatically generated

**Place Order**: Facilitates seamless order placement with a user-friendly interface. Employees can select the desired product from a dropdown menu, specify the quantity required, choose the relevant vendor from another dropdown, and initiate the ordering process with a single click.

A screenshot of a computer screen

Description automatically generated

***Inventory Page:***

The Inventory Page serves as a central hub for managing product information and stock levels efficiently.

**Product Management Form:** Features fields for entering and updating essential product details, including *Product ID, Name, Price, Quantity, Category, and Description. Employees* can perform actions such as adding new products, updating existing information, deleting products, and conducting product searches.

A screenshot of a computer

Description automatically generated

**Product List View:** Displays a comprehensive list of all available products along with their corresponding details, facilitating quick reference and overview of inventory items.

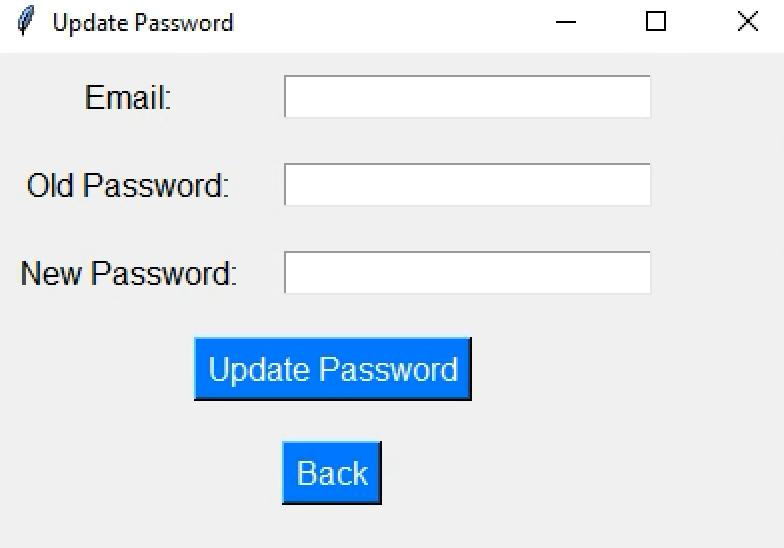
A screenshot of a computer

Description automatically generated

***Update Password Page:***

The Update Password Page enables Employees to securely update their login credentials.

**Password Update Form:** Provides fields for entering the existing password and specifying a new password. The system accommodates password changes without any restrictive requirements, ensuring user convenience and security.



***Customer Sales Page:***

The Customer Sales Page offers a streamlined interface for processing customer orders and managing sales transactions.

**Product Details Window**:

Features fields for searching and selecting products based on Product ID, Name, or Description. Additionally, Employees can specify the desired quantity from a dropdown menu, with the Total Price dynamically adjusting based on the selected quantity. An "Add to Cart" button facilitates the inclusion of selected items in the customer's order.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A place order button facilitates the customer sales page. After the place order button is clicked it automatically updates the quantity in the inventory.

**Design Elements:**

User Interface Design:

The user interface design follows principles of simplicity, consistency, and intuitiveness to enhance user experience and usability.

Clean layout, intuitive navigation menus, and visually appealing elements contribute to a user-friendly interface.

Use of minimalistic colors, icons, and typography is consistent across different modules and pages, ensuring coherence and ease of navigation.

Interactive elements such as dropdown menus, buttons, and input fields are designed to be responsive and intuitive, facilitating seamless interaction with the system.

Scalability and Performance:

The system architecture is designed to be scalable and adaptable to accommodate future growth and expansion.

1. **Database Design:**

ENTITY RELATIONSHIP DIAGRAM:

A diagram of a company

Description automatically generated

1. **System Process:**

**Authentication Process:**

* Upon accessing the system, Employees are prompted to authenticate themselves by providing their registered email address and password.
* The system verifies the credentials against the stored user database to grant access to authorized personnel.
* Failed authentication attempts trigger appropriate error messages, ensuring secure access to the system.

**Main Menu Navigation:**

* After successful authentication, Employees are directed to the Main Menu page, which serves as the central hub for accessing various system functionalities.
* Navigation between different modules, such as Inventory Management, Vendor Management, Customer Sales, and Password Update, is facilitated through intuitive menu options.
* Each menu option corresponds to specific functionalities and processes within the system, providing a structured and user-friendly interface for efficient operation.

**Inventory Management Process:**

* The Inventory Management process encompasses activities related to managing product information, stock levels, and inventory transactions.
* Employees can perform actions such as adding new products, updating existing product details, deleting obsolete items, and conducting product searches.
* Changes made to inventory records are logged and tracked to ensure data integrity and accountability.
* Automated alerts and notifications notify Employees of low stock levels, enabling timely replenishment and inventory optimization.

**Vendor Management Process:**

* The Vendor Management process involves managing relationships with external suppliers and facilitating the procurement of inventory items.
* Employees can access detailed vendor information, including contact details and order history, to streamline communication and collaboration.
* The process of placing orders involves selecting the desired products, specifying quantities, and assigning vendors for procurement.
* Order status tracking and communication with vendors are facilitated to ensure timely delivery and inventory replenishment.

**Customer Sales Process:**

* The Customer Sales process encompasses activities related to processing customer orders, managing sales transactions, and providing a seamless shopping experience.
* Employees can search for products, select items for purchase, specify quantities, and add selected items to the shopping cart.
* Order processing involves calculating total prices, applying discounts or promotions, and generating invoices for customers.
* Payment processing and order fulfillment are integrated into the system to streamline the entire sales process and enhance customer satisfaction.

1. **Code:**

**Main Python Code:**

from tkinter import \*

from tkinter import messagebox, font

import pymysql

import subprocess # Import subprocess module for opening a new Python script

# Create the main window

window = Tk()

window.title("Login")

window.geometry("400x250")

window.configure(bg="#F0F0F0")

# Set custom font

custom\_font = font.Font(family="Helvetica", size=12)

# Database connection

def connection():

conn = pymysql.connect(

host="141.209.241.91",

user='sp2024bis698g3',

password='warm',

db='sp2024bis698g3s'

)

return conn

# Function to check login credentials

def login():

email = email\_entry.get()

password = password\_entry.get()

# Check if email and password are provided

if email.strip() == "" or password.strip() == "":

messagebox.showwarning("Warning", "Please enter both email and password")

return

# Check if employee exists and password is correct

conn = connection()

cursor = conn.cursor()

cursor.execute("SELECT \* FROM Employee WHERE Email = %s AND PasswordHash = %s", (email, password))

result = cursor.fetchone()

conn.close()

if result:

messagebox.showinfo("Success", "Login successful!")

window.destroy() # Close login window

open\_main\_menu() # Open the main menu window

else:

messagebox.showerror("Error", "Invalid email or password")

# Function to open the main menu window

def open\_main\_menu():

main\_menu\_window = Tk()

main\_menu\_window.title("Main Menu")

main\_menu\_window.geometry("400x300")

main\_menu\_window.configure(bg="#F0F0F0")

# Add inventory button

inventory\_button = Button(main\_menu\_window, text="Inventory", command=open\_inventory, bg="#007bff", fg="white")

inventory\_button['font'] = custom\_font

inventory\_button.pack(pady=10)

# Add three more buttons

customer\_order\_button = Button(main\_menu\_window, text="Customer Order", command=customer\_order\_clicked, bg="#007bff", fg="white")

customer\_order\_button['font'] = custom\_font

customer\_order\_button.pack(pady=5)

vendor\_button = Button(main\_menu\_window, text="Vendor", command=vendor\_clicked, bg="#007bff", fg="white")

vendor\_button['font'] = custom\_font

vendor\_button.pack(pady=5)

update\_password\_button = Button(main\_menu\_window, text="Update Password", command=update\_password\_clicked, bg="#007bff", fg="white")

update\_password\_button['font'] = custom\_font

update\_password\_button.pack(pady=5)

main\_menu\_window.mainloop()

# Function to open the inventory window

def open\_inventory():

subprocess.Popen(["python", "InventoryPage.py"]) # Open InventoryPage.py using subprocess

# Placeholder functions for additional buttons

def customer\_order\_clicked():

subprocess.Popen(["python", "customer.py"]) # Open InventoryPage.py using subprocess

def vendor\_clicked():

subprocess.Popen(["python", "vendor.py"]) # Open InventoryPage.py using subprocess

def update\_password\_clicked():

subprocess.Popen(["python", "update\_password.py"]) # Open InventoryPage.py using subprocess

# Entry fields for email and password

email\_label = Label(window, text="Email:", bg="#F0F0F0")

email\_label['font'] = custom\_font

email\_label.grid(row=0, column=0, padx=10, pady=10)

email\_entry = Entry(window, bg="white")

email\_entry['font'] = custom\_font

email\_entry.grid(row=0, column=1, padx=10, pady=10)

password\_label = Label(window, text="Password:", bg="#F0F0F0")

password\_label['font'] = custom\_font

password\_label.grid(row=1, column=0, padx=10, pady=10)

password\_entry = Entry(window, show="\*", bg="white")

password\_entry['font'] = custom\_font

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

# Button to login

login\_button = Button(window, text="Login", command=login, bg="#007bff", fg="white")

login\_button['font'] = custom\_font

login\_button.grid(row=2, columnspan=2, padx=10, pady=10)

window.mainloop()

**Login Page Code:**

from tkinter import \*

from tkinter import messagebox, font

import pymysql

# Create the main window

window = Tk()

window.title("Login")

window.geometry("400x200")

window.configure(bg="#F0F0F0")

# Set custom font

custom\_font = font.Font(family="Helvetica", size=12)

# Database connection

def connection():

conn = pymysql.connect(

host='141.209.241.91',

user='sp2024bis698g3',

password='warm',

db='sp2024bis698g3s'

)

return conn

# Function to check login credentials

def login():

email = email\_entry.get()

password = password\_entry.get()

# Check if email and password are provided

if email.strip() == "" or password.strip() == "":

messagebox.showwarning("Warning", "Please enter both email and password")

return

# Check if employee exists and password is correct

conn = connection()

cursor = conn.cursor()

cursor.execute("SELECT \* FROM Employee WHERE Email = %s AND PasswordHash = %s", (email, password))

result = cursor.fetchone()

conn.close()

if result:

messagebox.showinfo("Success", "Login successful!")

window.destroy() # Close login window

# Add your code here to open the main menu window or another window

else:

messagebox.showerror("Error", "Invalid email or password")

# Entry fields for email and password

email\_label = Label(window, text="Email:", bg="#F0F0F0")

email\_label['font'] = custom\_font

email\_label.grid(row=0, column=0, padx=10, pady=10)

email\_entry = Entry(window, bg="white")

email\_entry['font'] = custom\_font

email\_entry.grid(row=0, column=1, padx=10, pady=10)

password\_label = Label(window, text="Password:", bg="#F0F0F0")

password\_label['font'] = custom\_font

password\_label.grid(row=1, column=0, padx=10, pady=10)

password\_entry = Entry(window, show="\*", bg="white")

password\_entry['font'] = custom\_font

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

# Button to login

login\_button = Button(window, text="Login", command=login, bg="#007bff", fg="white")

login\_button['font'] = custom\_font

login\_button.grid(row=2, columnspan=2, padx=10, pady=10)

window.mainloop()

**Update Password Python Code:**

from tkinter import \*

from tkinter import messagebox, font

import pymysql

# Create the main window

window = Tk()

window.title("Update Password")

window.geometry("400x250")

window.configure(bg="#F0F0F0")

# Set custom font

custom\_font = font.Font(family="Helvetica", size=12)

# Function to update password

def update\_password():

email = email\_entry.get()

old\_password = old\_password\_entry.get()

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

# Check if all fields are filled

if email.strip() == "" or old\_password.strip() == "" or new\_password.strip() == "":

messagebox.showwarning("Warning", "Please fill in all the fields")

return

# Database connection

conn = pymysql.connect(

host='141.209.241.91',

user='sp2024bis698g3',

password='warm',

db='sp2024bis698g3s'

)

cursor = conn.cursor()

# Check if old password is correct

cursor.execute("SELECT \* FROM Employee WHERE Email = %s AND PasswordHash = %s", (email, old\_password))

result = cursor.fetchone()

if result:

# Update password

cursor.execute("UPDATE Employee SET PasswordHash = %s WHERE Email = %s", (new\_password, email))

conn.commit()

conn.close()

messagebox.showinfo("Success", "Password updated successfully!")

window.destroy()

else:

messagebox.showerror("Error", "Invalid email or old password")

# Labels and entry fields for email, old password, and new password

email\_label = Label(window, text="Email:", bg="#F0F0F0")

email\_label['font'] = custom\_font

email\_label.grid(row=0, column=0, padx=10, pady=10)

email\_entry = Entry(window, bg="white")

email\_entry['font'] = custom\_font

email\_entry.grid(row=0, column=1, padx=10, pady=10)

old\_password\_label = Label(window, text="Old Password:", bg="#F0F0F0")

old\_password\_label['font'] = custom\_font

old\_password\_label.grid(row=1, column=0, padx=10, pady=10)

old\_password\_entry = Entry(window, show="\*", bg="white")

old\_password\_entry['font'] = custom\_font

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

new\_password\_label = Label(window, text="New Password:", bg="#F0F0F0")

new\_password\_label['font'] = custom\_font

new\_password\_label.grid(row=2, column=0, padx=10, pady=10)

new\_password\_entry = Entry(window, show="\*", bg="white")

new\_password\_entry['font'] = custom\_font

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

# Button to update password

update\_button = Button(window, text="Update Password", command=update\_password, bg="#007bff", fg="white")

update\_button['font'] = custom\_font

update\_button.grid(row=3, columnspan=2, padx=10, pady=10)

window.mainloop()

Inventory Page Python Code:

from tkinter import \*

from tkinter import ttk

from tkinter import messagebox

import tkinter

import random

import pymysql

import csv

from datetime import datetime

window = tkinter.Tk()

window.title("Inventory Management System")

window.geometry("720x640")

my\_tree = ttk.Treeview(window, show='headings', height=20)

style = ttk.Style()

placeholderArray = ['', '', '', '', '']

numeric = '1234567890'

alpha = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def connection():

conn = pymysql.connect(

host='141.209.241.91',

user='sp2024bis698g3',

password='warm',

db='sp2024bis698g3s'

)

return conn

conn = connection()

cursor = conn.cursor()

for i in range(0, 5):

placeholderArray[i] = tkinter.StringVar()

def read():

cursor.connection.ping()

sql = f"SELECT `ProductID`, `Name`, `Price`, `Quantity`, `Category`, `Description` FROM Product ORDER BY `ProductID` DESC"

cursor.execute(sql)

results = cursor.fetchall()

conn.commit()

conn.close()

return results

def refreshTable():

for data in my\_tree.get\_children():

my\_tree.delete(data)

for array in read():

my\_tree.insert(parent='', index='end', iid=array, text="", values=(array), tag="orow")

my\_tree.tag\_configure('orow', background="#EEEEEE")

my\_tree.pack()

def setph(word, num):

for ph in range(0, 5):

if ph == num:

placeholderArray[ph].set(word)

def save():

ProductID = str(ProductIDEntry.get())

Name = str(NameEntry.get())

Price = str(PriceEntry.get())

Quantity = str(QuantityEntry.get())

Category = str(CategoryCombo.get())

Description = str(DescriptionEntry.get("1.0", END))

if not ProductID.isdigit() or len(ProductID) != 2:

messagebox.showwarning("", "Invalid Product ID. Please enter a 5-digit numeric ID.")

return

if not (ProductID and ProductID.strip()) or not (Name and Name.strip()) or not (Price and Price.strip()) or not (

Quantity and Quantity.strip()) or not (Category and Category.strip()) or not (

Description and Description.strip()):

messagebox.showwarning("", "Please fill up all entries")

return

try:

cursor.connection.ping()

sql = f"SELECT \* FROM Product WHERE `ProductID` = '{ProductID}' "

cursor.execute(sql)

checkProductNo = cursor.fetchall()

if len(checkProductNo) > 0:

messagebox.showwarning("", "Product ID already used")

return

else:

cursor.connection.ping()

sql = f"INSERT INTO Product (`ProductID`, `Name`, `Price`, `Quantity`, `Category`, `Description`) VALUES ('{ProductID}','{Name}','{Price}','{Quantity}','{Category}','{Description}')"

cursor.execute(sql)

conn.commit()

conn.close()

for num in range(0, 5):

setph('', (num))

except Exception as e:

print(e)

messagebox.showwarning("", "Error while saving ref: " + str(e))

return

refreshTable()

def update():

selectedProductID = ''

try:

selectedItem = my\_tree.selection()[0]

selectedProductID = str(my\_tree.item(selectedItem)['values'][0])

except:

messagebox.showwarning("", "Please select a data row")

print(selectedProductID)

ProductID = str(ProductIDEntry.get())

Name = str(NameEntry.get())

Price = str(PriceEntry.get())

Quantity = str(QuantityEntry.get())

Category = str(CategoryCombo.get())

Description = str(DescriptionEntry.get("1.0", END))

if not (ProductID and ProductID.strip()) or not (Name and Name.strip()) or not (Price and Price.strip()) or not (

Quantity and Quantity.strip()) or not (Category and Category.strip()) or not (

Description and Description.strip()):

messagebox.showwarning("", "Please fill up all entries")

return

if selectedProductID != ProductID:

messagebox.showwarning("", "You can't change Product ID")

return

try:

cursor.connection.ping()

sql = f"UPDATE Product SET `Name` = '{Name}', `Price` = '{Price}', `Quantity` = '{Quantity}', `Category` = '{Category}', `Description` = '{Description}' WHERE `ProductID` = '{ProductID}' "

cursor.execute(sql)

conn.commit()

conn.close()

for num in range(0, 5):

setph('', (num))

except Exception as err:

messagebox.showwarning("", "Error occured ref: " + str(err))

return

refreshTable()

def delete():

try:

if (my\_tree.selection()[0]):

decision = messagebox.askquestion("", "Delete the selected data?")

if (decision != 'yes'):

return

else:

selectedItem = my\_tree.selection()[0]

ProductID = str(my\_tree.item(selectedItem)['values'][0])

try:

cursor.connection.ping()

sql = f"DELETE FROM Product WHERE `ProductID` = '{ProductID}' "

cursor.execute(sql)

conn.commit()

conn.close()

messagebox.showinfo("", "Data has been successfully deleted")

except:

messagebox.showinfo("", "Sorry, an error occured")

refreshTable()

except:

messagebox.showwarning("", "Please select a data row")

def select():

try:

selectedItem = my\_tree.selection()[0]

ProductID = str(my\_tree.item(selectedItem)['values'][0])

Name = str(my\_tree.item(selectedItem)['values'][1])

Price = str(my\_tree.item(selectedItem)['values'][2])

Quantity = str(my\_tree.item(selectedItem)['values'][3])

Category = str(my\_tree.item(selectedItem)['values'][4])

Description = str(my\_tree.item(selectedItem)['values'][5])

setph(ProductID, 0)

setph(Name, 1)

setph(Price, 2)

setph(Quantity, 3)

setph(Category, 4)

DescriptionEntry.delete("1.0", END)

DescriptionEntry.insert("1.0", Description)

except:

messagebox.showwarning("", "Please select a data row")

def find():

ProductID = str(ProductIDEntry.get())

Name = str(NameEntry.get())

Price = str(PriceEntry.get())

Quantity = str(QuantityEntry.get())

Category = str(CategoryCombo.get())

cursor.connection.ping()

if (ProductID and ProductID.strip()):

sql = f"SELECT `ProductID`, `Name`, `Price`, `Quantity`, `Category`, `Description` FROM Product WHERE `ProductID` LIKE '%{ProductID}%' "

elif (Name and Name.strip()):

sql = f"SELECT `ProductID`, `Name`, `Price`, `Quantity`, `Category`, `Description` FROM Product WHERE `Name` LIKE '%{Name}%' "

elif (Price and Price.strip()):

sql = f"SELECT `ProductID`, `Name`, `Price`, `Quantity`, `Category`, `Description` FROM Product WHERE `Price` LIKE '%{Price}%' "

elif (Quantity and Quantity.strip()):

sql = f"SELECT `ProductID`, `Name`, `Price`, `Quantity`, `Category`, `Description` FROM Product WHERE `Quantity` LIKE '%{Quantity}%' "

elif (Category and Category.strip()):

sql = f"SELECT `ProductID`, `Name`, `Price`, `Quantity`, `Category`, `Description` FROM Product WHERE `Category` LIKE '%{Category}%' "

else:

messagebox.showwarning("", "Please fill up one of the entries")

return

cursor.execute(sql)

try:

result = cursor.fetchall();

for num in range(0, 5):

setph(result[0][num], (num))

DescriptionEntry.delete("1.0", END)

DescriptionEntry.insert("1.0", result[0][5])

conn.commit()

conn.close()

except:

messagebox.showwarning("", "No data found")

def clear():

for num in range(0, 5):

setph('', (num))

DescriptionEntry.delete("1.0", END)

def exportExcel():

cursor.connection.ping()

sql = f"SELECT `ProductID`, `Name`, `Price`, `Quantity`, `Category`, `Description` FROM Product ORDER BY `ProductID` DESC"

cursor.execute(sql)

dataraw = cursor.fetchall()

date = str(datetime.now())

date = date.replace(' ', '\_')

date = date.replace(':', '-')

dateFinal = date[0:16]

with open("products\_" + dateFinal + ".csv", 'a', newline='') as f:

w = csv.writer(f, dialect='excel')

for record in dataraw:

w.writerow(record)

print("saved: products\_" + dateFinal + ".csv")

conn.commit()

conn.close()

messagebox.showinfo("", "Excel file downloaded")

frame = tkinter.Frame(window, bg="#02577A")

frame.pack()

btnColor = "#196E78"

manageFrame = tkinter.LabelFrame(frame, text="Manage", borderwidth=5)

manageFrame.grid(row=0, column=0, sticky="w", padx=[10, 200], pady=20, ipadx=[6])

saveBtn = Button(manageFrame, text="SAVE", width=10, borderwidth=3, bg=btnColor, fg='white', command=save)

updateBtn = Button(manageFrame, text="UPDATE", width=10, borderwidth=3, bg=btnColor, fg='white', command=update)

deleteBtn = Button(manageFrame, text="DELETE", width=10, borderwidth=3, bg=btnColor, fg='white', command=delete)

selectBtn = Button(manageFrame, text="SELECT", width=10, borderwidth=3, bg=btnColor, fg='white', command=select)

findBtn = Button(manageFrame, text="FIND", width=10, borderwidth=3, bg=btnColor, fg='white', command=find)

clearBtn = Button(manageFrame, text="CLEAR", width=10, borderwidth=3, bg=btnColor, fg='white', command=clear)

exportBtn = Button(manageFrame, text="EXPORT EXCEL", width=15, borderwidth=3, bg=btnColor, fg='white',

command=exportExcel)

saveBtn.grid(row=0, column=0, padx=5, pady=5)

updateBtn.grid(row=0, column=1, padx=5, pady=5)

deleteBtn.grid(row=0, column=2, padx=5, pady=5)

selectBtn.grid(row=0, column=3, padx=5, pady=5)

findBtn.grid(row=0, column=4, padx=5, pady=5)

clearBtn.grid(row=0, column=5, padx=5, pady=5)

exportBtn.grid(row=0, column=6, padx=5, pady=5)

entriesFrame = tkinter.LabelFrame(frame, text="Form", borderwidth=5)

entriesFrame.grid(row=1, column=0, sticky="w", padx=[10, 200], pady=[0, 20], ipadx=[6])

ProductIDLabel = Label(entriesFrame, text="PRODUCT ID", anchor="e", width=10)

NameLabel = Label(entriesFrame, text="NAME", anchor="e", width=10)

PriceLabel = Label(entriesFrame, text="PRICE", anchor="e", width=10)

QuantityLabel = Label(entriesFrame, text="QUANTITY", anchor="e", width=10)

CategoryLabel = Label(entriesFrame, text="CATEGORY", anchor="e", width=10)

DescriptionLabel = Label(entriesFrame, text="DESCRIPTION", anchor="e", width=10)

ProductIDLabel.grid(row=0, column=0, padx=10)

NameLabel.grid(row=1, column=0, padx=10)

PriceLabel.grid(row=2, column=0, padx=10)

QuantityLabel.grid(row=3, column=0, padx=10)

CategoryLabel.grid(row=4, column=0, padx=10)

DescriptionLabel.grid(row=5, column=0, padx=10)

categoryArray = ['Networking Tools', 'Computer Parts', 'Repair Tools', 'Gadgets']

ProductIDEntry = Entry(entriesFrame, width=50, textvariable=placeholderArray[0])

NameEntry = Entry(entriesFrame, width=50, textvariable=placeholderArray[1])

PriceEntry = Entry(entriesFrame, width=50, textvariable=placeholderArray[2])

QuantityEntry = Entry(entriesFrame, width=50, textvariable=placeholderArray[3])

CategoryCombo = ttk.Combobox(entriesFrame, width=47, textvariable=placeholderArray[4], values=categoryArray)

DescriptionEntry = Text(entriesFrame, width=47, height=5)

ProductIDEntry.grid(row=0, column=1, padx=5, pady=5)

NameEntry.grid(row=1, column=1, padx=5, pady=5)

PriceEntry.grid(row=2, column=1, padx=5, pady=5)

QuantityEntry.grid(row=3, column=1, padx=5, pady=5)

CategoryCombo.grid(row=4, column=1, padx=5, pady=5)

DescriptionEntry.grid(row=5, column=1, padx=5, pady=5)

style.configure(window)

my\_tree['columns'] = ("ProductID", "Name", "Price", "Quantity", "Category", "Description")

my\_tree.column("#0", width=0, stretch=NO)

my\_tree.column("ProductID", anchor=W, width=70)

my\_tree.column("Name", anchor=W, width=125)

my\_tree.column("Price", anchor=W, width=125)

my\_tree.column("Quantity", anchor=W, width=100)

my\_tree.column("Category", anchor=W, width=150)

my\_tree.column("Description", anchor=W, width=150)

my\_tree.heading("ProductID", text="ProductID", anchor=W)

my\_tree.heading("Name", text="Name", anchor=W)

my\_tree.heading("Price", text="Price", anchor=W)

my\_tree.heading("Quantity", text="Quantity", anchor=W)

my\_tree.heading("Category", text="Category", anchor=W)

my\_tree.heading("Description", text="Description", anchor=W)

my\_tree.tag\_configure('orow', background="#EEEEEE")

my\_tree.pack()

refreshTable()

window.resizable(False, False)

window.mainloop()

**Customer Page Python Code:**

import tkinter as tk

from tkinter import ttk, messagebox

import pymysql

class CustomerOrderPage(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Customer Order Page")

self.geometry("800x600")

self.customerID = tk.StringVar()

self.customerName = tk.StringVar()

self.productID = tk.StringVar()

self.productName = tk.StringVar()

self.description = tk.StringVar()

self.price = tk.StringVar()

self.quantity = tk.IntVar(value=1)

self.totalPrice = tk.StringVar()

self.order\_list = []

self.create\_widgets()

def create\_widgets(self):

customer\_frame = ttk.LabelFrame(self, text="Customer Details")

customer\_frame.pack(padx=10, pady=10, fill="both", expand=True)

tk.Label(customer\_frame, text="Customer ID:").grid(row=0, column=0, padx=5, pady=5, sticky="e")

tk.Entry(customer\_frame, textvariable=self.customerID).grid(row=0, column=1, padx=5, pady=5)

tk.Label(customer\_frame, text="Customer Name:").grid(row=1, column=0, padx=5, pady=5, sticky="e")

tk.Entry(customer\_frame, textvariable=self.customerName).grid(row=1, column=1, padx=5, pady=5)

product\_frame = ttk.LabelFrame(self, text="Product Details")

product\_frame.pack(padx=10, pady=10, fill="both", expand=True)

tk.Label(product\_frame, text="Product ID:").grid(row=0, column=0, padx=5, pady=5, sticky="e")

tk.Entry(product\_frame, textvariable=self.productID).grid(row=0, column=1, padx=5, pady=5)

tk.Button(product\_frame, text="Search", command=self.search\_product).grid(row=0, column=2, padx=5, pady=5)

tk.Label(product\_frame, text="Product Name:").grid(row=1, column=0, padx=5, pady=5, sticky="e")

tk.Entry(product\_frame, textvariable=self.productName, state="readonly").grid(row=1, column=1, padx=5, pady=5)

tk.Label(product\_frame, text="Description:").grid(row=2, column=0, padx=5, pady=5, sticky="e")

tk.Entry(product\_frame, textvariable=self.description, state="readonly").grid(row=2, column=1, padx=5, pady=5)

tk.Label(product\_frame, text="Price:").grid(row=3, column=0, padx=5, pady=5, sticky="e")

tk.Entry(product\_frame, textvariable=self.price, state="readonly").grid(row=3, column=1, padx=5, pady=5)

tk.Label(product\_frame, text="Quantity:").grid(row=4, column=0, padx=5, pady=5, sticky="e")

tk.Spinbox(product\_frame, from\_=1, to=100, textvariable=self.quantity).grid(row=4, column=1, padx=5, pady=5)

tk.Label(product\_frame, text="Total Price:").grid(row=5, column=0, padx=5, pady=5, sticky="e")

tk.Entry(product\_frame, textvariable=self.totalPrice, state="readonly").grid(row=5, column=1, padx=5, pady=5)

tk.Button(product\_frame, text="Add to Cart", command=self.add\_to\_cart).grid(row=6, column=1, padx=5, pady=5)

order\_frame = ttk.LabelFrame(self, text="Order Details")

order\_frame.pack(padx=10, pady=10, fill="both", expand=True)

self.order\_table = ttk.Treeview(order\_frame, columns=("Product ID", "Product Name", "Description", "Price", "Quantity", "Total Price"), show="headings")

self.order\_table.heading("Product ID", text="Product ID")

self.order\_table.heading("Product Name", text="Product Name")

self.order\_table.heading("Description", text="Description")

self.order\_table.heading("Price", text="Price")

self.order\_table.heading("Quantity", text="Quantity")

self.order\_table.heading("Total Price", text="Total Price")

self.order\_table.column("Product ID", width=100)

self.order\_table.column("Product Name", width=150)

self.order\_table.column("Description", width=200)

self.order\_table.column("Price", width=100)

self.order\_table.column("Quantity", width=100)

self.order\_table.column("Total Price", width=100)

self.order\_table.pack(padx=10, pady=10, fill="both", expand=True)

place\_order\_button = tk.Button(self, text="Place Order", command=self.place\_order)

place\_order\_button.pack(padx=10, pady=10, anchor="ne")

def search\_product(self):

product\_id = self.productID.get()

if not product\_id:

messagebox.showerror("Error", "Please enter a product ID!")

return

conn = self.connect\_to\_database()

if conn:

cursor = conn.cursor()

cursor.execute(f"SELECT Name, Description, Price FROM Product WHERE ProductID = '{product\_id}'")

result = cursor.fetchone()

conn.close()

if result:

self.productName.set(result[0])

self.description.set(result[1])

self.price.set(result[2])

self.calculate\_total\_price()

else:

messagebox.showerror("Error", "Product not found!")

def calculate\_total\_price(self):

try:

price = float(self.price.get())

quantity = int(self.quantity.get())

total\_price = price \* quantity

self.totalPrice.set(total\_price)

except ValueError:

messagebox.showerror("Error", "Invalid price or quantity!")

def add\_to\_cart(self):

product\_id = self.productID.get()

product\_name = self.productName.get()

description = self.description.get()

price\_str = self.price.get()

quantity = int(self.quantity.get())

if not product\_id:

messagebox.showerror("Error", "Please enter a product ID!")

return

if not product\_name:

messagebox.showerror("Error", "Product name is empty!")

return

if not description:

messagebox.showerror("Error", "Product description is empty!")

return

if not price\_str:

messagebox.showerror("Error", "Product price is empty!")

return

try:

price = float(price\_str)

except ValueError:

messagebox.showerror("Error", "Invalid product price!")

return

total\_price = price \* quantity

self.order\_table.insert("", "end", values=(product\_id, product\_name, description, price, quantity, total\_price))

self.order\_list.append((product\_id, quantity))

self.productID.set("")

self.productName.set("")

self.description.set("")

self.price.set("")

self.quantity.set(1)

self.totalPrice.set("")

def place\_order(self):

if not self.order\_list:

messagebox.showerror("Error", "No items in the order!")

return

if not self.customerID.get() or not self.customerName.get():

messagebox.showerror("Error", "Please enter customer ID and name!")

return

conn = self.connect\_to\_database()

if conn:

cursor = conn.cursor()

# Retrieve customer ID based on entered customer ID or name

cursor.execute(f"SELECT CustomerID FROM Customer WHERE CustomerID = '{self.customerID.get()}' OR FullName = '{self.customerName.get()}'")

customer = cursor.fetchone()

if not customer:

messagebox.showerror("Error", "Customer not found!")

conn.close()

return

customer\_id = customer[0]

# Calculate total amount

total\_amount = sum(item[1] for item in self.order\_list)

# Insert order into Order table using parameterized query

cursor.execute("INSERT INTO `Order` (CustomerID, TotalAmount) VALUES (%s, %s)", (customer\_id, total\_amount))

order\_id = cursor.lastrowid

# Insert order items into Order\_Product table

for item in self.order\_list:

product\_id, quantity = item

cursor.execute(f"INSERT INTO Order\_Product (OrderID, ProductID, Quantity) VALUES ('{order\_id}', '{product\_id}', '{quantity}')")

conn.commit()

# Fetch total order value from Order\_Product table and update Order table

cursor.execute(f"SELECT SUM(p.Price \* op.Quantity) AS TotalOrderValue FROM Product p JOIN Order\_Product op ON p.ProductID = op.ProductID WHERE op.OrderID = {order\_id}")

total\_order\_value = cursor.fetchone()[0]

cursor.execute("UPDATE `Order` SET TotalAmount = %s WHERE OrderID = %s", (total\_order\_value, order\_id))

conn.commit()

conn.close()

messagebox.showinfo("Success", "Order placed successfully!")

self.order\_list.clear()

def connect\_to\_database(self):

try:

conn = pymysql.connect(

host='141.209.241.91',

user='sp2024bis698g3',

password='warm',

db='sp2024bis698g3s'

)

return conn

except pymysql.Error as e:

messagebox.showerror("Database Error", f"Error: {e}")

return None

if \_\_name\_\_ == "\_\_main\_\_":

app = CustomerOrderPage()

app.mainloop()

**Vendor Page Python Code:**

import tkinter as tk

from tkinter import ttk, messagebox

import pymysql

import random

class MainPage:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Vendor")

# Main Frame

self.main\_frame = tk.Frame(self.root)

self.main\_frame.pack(padx=20, pady=20)

# Place Order Button

self.place\_order\_button = ttk.Button(self.main\_frame, text="Place Order", command=self.place\_order)

self.place\_order\_button.grid(row=0, column=0, padx=10, pady=10)

# Vendor Details Button

self.vendor\_details\_button = ttk.Button(self.main\_frame, text="Vendor Details", command=self.redirect\_to\_vendor\_details)

self.vendor\_details\_button.grid(row=0, column=1, padx=10, pady=10)

def place\_order(self):

# Instantiate PlaceOrderPage and open it

place\_order\_window = tk.Toplevel(self.root)

place\_order\_page = PlaceOrderPage(place\_order\_window)

def redirect\_to\_vendor\_details(self):

# Instantiate VendorDetailsPage and open it

vendor\_details\_window = tk.Toplevel(self.root)

vendor\_details\_page = VendorDetailsPage(vendor\_details\_window)

class VendorDetailsPage:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Vendor Details")

# Main Frame

self.main\_frame = tk.Frame(self.root)

self.main\_frame.pack(padx=20, pady=20)

# Fetch vendors from database and display details in a grid view

self.display\_vendor\_details()

# Back Button

self.back\_button = ttk.Button(self.root, text="Back", command=self.go\_back)

self.back\_button.pack(side=tk.BOTTOM, pady=10)

def display\_vendor\_details(self):

# Connect to MySQL database

connection = self.connect\_to\_database()

if connection:

try:

with connection.cursor() as cursor:

# Fetch vendor details

cursor.execute("SELECT VendorID, VendorName, ContactPerson FROM Vendor")

vendors = cursor.fetchall()

if vendors:

# Display vendor details in a grid view

tk.Label(self.main\_frame, text="Vendor ID").grid(row=0, column=0, padx=5, pady=5)

tk.Label(self.main\_frame, text="Vendor Name").grid(row=0, column=1, padx=5, pady=5)

tk.Label(self.main\_frame, text="Contact Person").grid(row=0, column=2, padx=5, pady=5)

for i, vendor in enumerate(vendors, start=1):

tk.Label(self.main\_frame, text=vendor[0]).grid(row=i, column=0, padx=5, pady=5)

tk.Label(self.main\_frame, text=vendor[1]).grid(row=i, column=1, padx=5, pady=5)

tk.Label(self.main\_frame, text=vendor[2]).grid(row=i, column=2, padx=5, pady=5)

else:

messagebox.showerror("Error", "No vendors found in the database.")

except pymysql.Error as e:

messagebox.showerror("Database Error", f"Error executing SQL query: {e}")

finally:

connection.close()

else:

messagebox.showerror("Database Error", "Failed to connect to the database.")

def connect\_to\_database(self):

try:

connection = pymysql.connect(

host="141.209.241.91",

user="sp2024bis698g3",

password="warm",

database="sp2024bis698g3s",

)

return connection

except pymysql.Error as e:

messagebox.showerror("Database Error", f"Error connecting to MySQL database: {e}")

return None

def go\_back(self):

self.root.destroy()

class PlaceOrderPage(tk.Frame):

def \_\_init\_\_(self, master):

super().\_\_init\_\_(master)

self.master = master

self.master.title("Place Order")

# Main Frame

self.main\_frame = tk.Frame(self.master)

self.main\_frame.pack(padx=20, pady=20)

# Product Name Label and Entry

tk.Label(self.main\_frame, text="Product Name:").grid(row=0, column=0, padx=5, pady=5)

self.product\_name\_entry = ttk.Combobox(self.main\_frame, state="readonly")

self.product\_name\_entry.grid(row=0, column=1, padx=5, pady=5)

# Quantity Label and Entry

tk.Label(self.main\_frame, text="Quantity:").grid(row=1, column=0, padx=5, pady=5)

self.quantity\_entry = ttk.Entry(self.main\_frame)

self.quantity\_entry.grid(row=1, column=1, padx=5, pady=5)

# Vendor Label and Dropdown

tk.Label(self.main\_frame, text="Vendor:").grid(row=2, column=0, padx=5, pady=5)

self.vendor\_dropdown = ttk.Combobox(self.main\_frame, state="readonly")

self.vendor\_dropdown.grid(row=2, column=1, padx=5, pady=5)

# Place Order Button

self.place\_order\_button = ttk.Button(self.main\_frame, text="Place Order", command=self.place\_order)

self.place\_order\_button.grid(row=3, columnspan=2, padx=5, pady=10)

# Populate product and vendor dropdowns

self.populate\_product\_dropdown()

self.populate\_vendor\_dropdown()

def place\_order(self):

product\_name = self.product\_name\_entry.get()

quantity = self.quantity\_entry.get()

vendor\_name = self.vendor\_dropdown.get()

if not product\_name or not quantity or not vendor\_name:

messagebox.showwarning("Incomplete Information", "Please select a product, enter a quantity, and select a vendor.")

return

# Connect to the database

conn = self.connect\_to\_database()

# Get the product ID

product\_id = self.get\_product\_id(conn, product\_name)

if product\_id is None:

messagebox.showerror("Error", "Selected product not found in the database")

conn.close()

return

# Generate a random order ID

order\_id = self.generate\_order\_id()

# Insert the order into the Order\_Product table

cursor = conn.cursor()

cursor.execute("INSERT INTO Order\_Product (OrderID, ProductID, Quantity) VALUES (%s, %s, %s)", (order\_id, product\_id, quantity))

conn.commit()

# Close the database connection

conn.close()

# Display success message

messagebox.showinfo("Success", "Order placed successfully!")

def populate\_product\_dropdown(self):

# Connect to MySQL database

connection = self.connect\_to\_database()

if connection:

try:

with connection.cursor() as cursor:

# Fetch product names

cursor.execute("SELECT Name FROM Product")

products = cursor.fetchall()

product\_names = [product[0] for product in products]

self.product\_name\_entry["values"] = product\_names

except pymysql.Error as e:

messagebox.showerror("Database Error", f"Error executing SQL query: {e}")

finally:

connection.close()

else:

messagebox.showerror("Database Error", "Failed to connect to the database.")

def populate\_vendor\_dropdown(self):

# Connect to MySQL database

connection = self.connect\_to\_database()

if connection:

try:

with connection.cursor() as cursor:

# Fetch vendor names

cursor.execute("SELECT VendorName FROM Vendor")

vendors = cursor.fetchall()

vendor\_names = [vendor[0] for vendor in vendors]

self.vendor\_dropdown["values"] = vendor\_names

except pymysql.Error as e:

messagebox.showerror("Database Error", f"Error executing SQL query: {e}")

finally:

connection.close()

else:

messagebox.showerror("Database Error", "Failed to connect to the database.")

def connect\_to\_database(self):

try:

connection = pymysql.connect(

host="141.209.241.91",

user="sp2024bis698g3",

password="warm",

database="sp2024bis698g3s",

)

return connection

except pymysql.Error as e:

messagebox.showerror("Database Error", f"Error connecting to MySQL database: {e}")

return None

def get\_product\_id(self, conn, product\_name):

try:

with conn.cursor() as cursor:

cursor.execute("SELECT ProductID FROM Product WHERE Name = %s", (product\_name,))

result = cursor.fetchone()

if result:

return result[0] # Fetch the first column (ProductID)

else:

return None

except pymysql.Error as e:

messagebox.showerror("Database Error", f"Error executing SQL query: {e}")

return None

def generate\_order\_id(self):

# Generate a random order ID

return random.randint(1000, 9999)

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

main\_page = MainPage(root)

root.mainloop()