A PROJECT REPORT ON ONLINE STORE DATABASE

SUBMITTED BY

Swetha S - 220701298

Vanisree S.J - 220701308

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING IN
COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE
(AUTONOMOUS) THANDALAM
CHENNAI-602105

2023 – 24

BONAFIDE CERTIFICATE

Certified that this project report “Online Store Database” is the bonafide work of **“SWETHA.S - 220701298, VANISREE S.J - 220701308”** who carried out the project work under my supervision

Submitted for the Practical Examination held on

SIGNATURE	**SIGNATURE**
Dr.R.SABITHA	**V.JANANI**
Professor	Assistant Professor
Computer Science And Engineering	Computer Science And Engineering
Rajalakshmi Engineering College,	Rajalakshmi Engineering College
Chennai - 602 105	Chennai - 602 105

ABSTRACT

The Online Store Database Project in Database Management Systems (DBMS) aims to design and implement a robust, scalable, and efficient database to manage the operations of an e-commerce platform. This project involves the creation of a relational database model to handle various aspects of an online store, including user management, product inventory, order processing, payment transactions, and customer feedback.The database schema is designed to ensure data integrity, minimize redundancy, and optimize query performance.

Key entities such as customers, products, orders, categories, and reviews are meticulously modeled with appropriate relationships and constraints. Advanced SQL techniques are employed to facilitate complex queries and transactions, ensuring seamless data retrieval and manipulation.The project also focuses on implementing security measures to protect sensitive customer information and ensure compliance with data privacy regulations.

Backup and recovery strategies are incorporated to safeguard against data loss and ensure business continuity.Furthermore, the database supports scalability to accommodate the growing data needs of the online store, allowing for future expansions and integrations with other systems. Through this project, students gain practical experience in database design, implementation, and management, preparing them for real-world applications in the e-commerce industry.

The project leverages SQL for relational data management and may incorporate NoSQL databases for specific requirements like handling large volumes of unstructured data. The goal is to provide a seamless shopping experience with reliable backend support.

TABLE OF CONTENTS

Chapter 1

Introduction

1.1 Introduction

The Online Store Database Project aims to develop a comprehensive and efficient database system to support the various functionalities required by an e-commerce platform. In the rapidly evolving world of online retail, having a robust database is crucial for managing product catalogs, user accounts, orders, payments, and inventory seamlessly. This project will focus on creating a structured database that ensures data integrity, security, and performance, thereby facilitating a smooth shopping experience for customers and streamlined operations for administrators.

The scope of this project includes several key components. Product catalog management involves storing detailed product information, handling inventory levels, and updating stock statuses. User account management encompasses maintaining user profiles, securing authentication, and managing order histories. Order processing will cover the recording and tracking of customer orders, payment transactions, and shipping details. Additionally, the project will integrate secure payment gateways for billing and generate invoices. Inventory and supply chain management will be automated to monitor stock levels and manage supplier information efficiently. Each of these components is designed to work cohesively to provide a comprehensive solution for an online store.

The methodology for executing the project includes phases such as requirements analysis, database design, implementation, testing, optimization, and deployment. Initially, the project will gather and analyze the needs of the online store to define a suitable data model. An Entity-Relationship (ER) diagram will be created to visually represent the database structure. The implementation phase will involve developing the database schema using a relational database management system (RDBMS) like MySQL or PostgreSQL, and writing SQL queries, stored procedures, and triggers. Extensive testing will ensure the database meets performance, reliability, and security standards. Finally, the database will be deployed in a production environment with backup, recovery, and maintenance procedures in place. This project will result in a scalable and efficient database system, providing a solid foundation for the growth and success of e-commerce platforms.

1.2 Objectives

The primary objectives of the Online Store Database are:

Comprehensive Product Management:

- Design and implement a database system to store detailed information about products, including names, descriptions, prices, categories, and images.
- Develop mechanisms for managing inventory levels, updating stock statuses, and automating restock alerts.

Secure Payment and Billing Integration:

- Integrate secure payment gateways to facilitate seamless processing of transactions.
- Develop functionalities for generating invoices, managing billing information, and ensuring the security of payment data.

Efficient User Account Management:

- Create a secure system for storing and managing user profiles, including personal information, login credentials, and order history.
- Implement robust authentication and authorization processes to ensure data security and user privacy.

1.3 Modules

1. Admin Module:

User Account Administration:

- Allow administrators to view, edit, and manage user profiles, including resetting passwords, updating personal information, and managing user roles and permissions.
- Secure access controls to ensure only authorized personnel can make changes, activity logs to track administrative actions, and tools for handling user queries and support tickets.

Order and Payment Management:

- Enable administrators to monitor and manage customer orders, update order statuses, process returns and refunds, and oversee payment transactions.

Reports and Analytics:

- Provide comprehensive reporting and analytics tools for administrators to gain insights into sales performance, customer behavior, inventory levels, and other key metrics.

2. Database Module:

- **Data Storage**: Securely stores all data, user information, and system logs.
- **Data Retrieval:** Efficiently retrieves data for real-time processing and reporting.

3. Front-End Module:

- **User Interface**: Built with Python for a responsive and visually appealing interface.
- **Dynamic Interaction**: Uses Python Libraries like tkinter to enable real-time updates and interactivity without full page reloads.

4. Security Module:

- **Authentication and Authorization**: Ensures that only authorized users can access certain features and data.

- **Data Encryption:** Protects sensitive financial data via encryption techniques. This achieves an efficient solution for managing finances.

Chapter 2

Survey of Technologies

The development of the Online Store Database leverages a variety of modern web technologies to achieve a robust, efficient, and user-friendly solution. This survey provides an overview of the key technologies used in the system and their respective roles.

FRONT END LANGUAGE:

PYTHON:

Role:

The Online Store Database Project acts as the central hub for managing all data related to an e-commerce platform. It organizes and secures information on products, users, orders, payments, and inventory, ensuring smooth and reliable business operations.

Usage:

- **Product Management:** Store and update product details and inventory levels.
- **User Account Management:** Maintain user profiles and manage authentication.
- **Order Processing:** Track customer orders, manage statuses, and handle payments.
- **Payment and Billing:** Process transactions securely and manage billing information.
- **Inventory Management:** Monitor stock levels and automate restocking processes.

Advantage:

- **Improved Efficiency:** Automates tasks, reducing manual work and increasing operational efficiency.
- **Enhanced Data Integrity and Security:** Ensures accurate, consistent, and protected data.
- **Scalability:** Supports business growth without compromising performance.

- **Better Decision-Making:** Provides insights through reporting and analytics, aiding informed decision-making.

BACK END LANGUAGE:

MySQL:

Role: Relational database management system.

Usage: MySQL is utilized for storing, retrieving, and managing all the data related to transactions, users, and system logs. It ensures data integrity and supports complex queries necessary for reporting and analytics.

Advantages:

- High performance and reliability.
- Robust security features.
- Scalability to handle growing amounts of data.

Chapter 3

REQUIREMENT AND ANALYSIS

3.1 REQUIREMENTS SPECIFICATION

User Requirements for the Online Store Database Project:

1. **User-Friendly Interface:**
    - Easy-to-navigate interfaces for both customers and administrators.
    - Simple and intuitive forms for product browsing, user registration, order placement, and account management.
2. **Secure User Authentication:**
    - Secure login and registration processes.
    - Password recovery and multi-factor authentication (MFA) options.
3. **Efficient Product Search and Filtering:**

    - Advanced search functionality with filters for categories, price ranges, ratings, etc.
    - Quick access to detailed product information and images.

4.Real-Time Inventory Updates:

- Accurate and real-time display of product availability.
- Notifications for low stock or out-of-stock items.

Software Requirements for the Online Store Database Project

1. **Database Management System (DBMS):**
    - Relational DBMS such as MySQL, PostgreSQL, or SQL Server.
    - Support for SQL queries, stored procedures, and triggers.
2. **Server-Side Technology:**
    - Server-side scripting language like PHP, Python, Ruby, or Node.js.
    - Web server software such as Apache or Nginx.
3. **Client-Side Technology:**
    - HTML, CSS, and JavaScript for front-end development.
    - Frameworks/libraries such as React, Angular, or Vue.js for a dynamic user interface.
4. **Payment Gateway Integration:**
    - APIs for integrating popular payment gateways (Stripe, PayPal, etc.).

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

Software Requirements

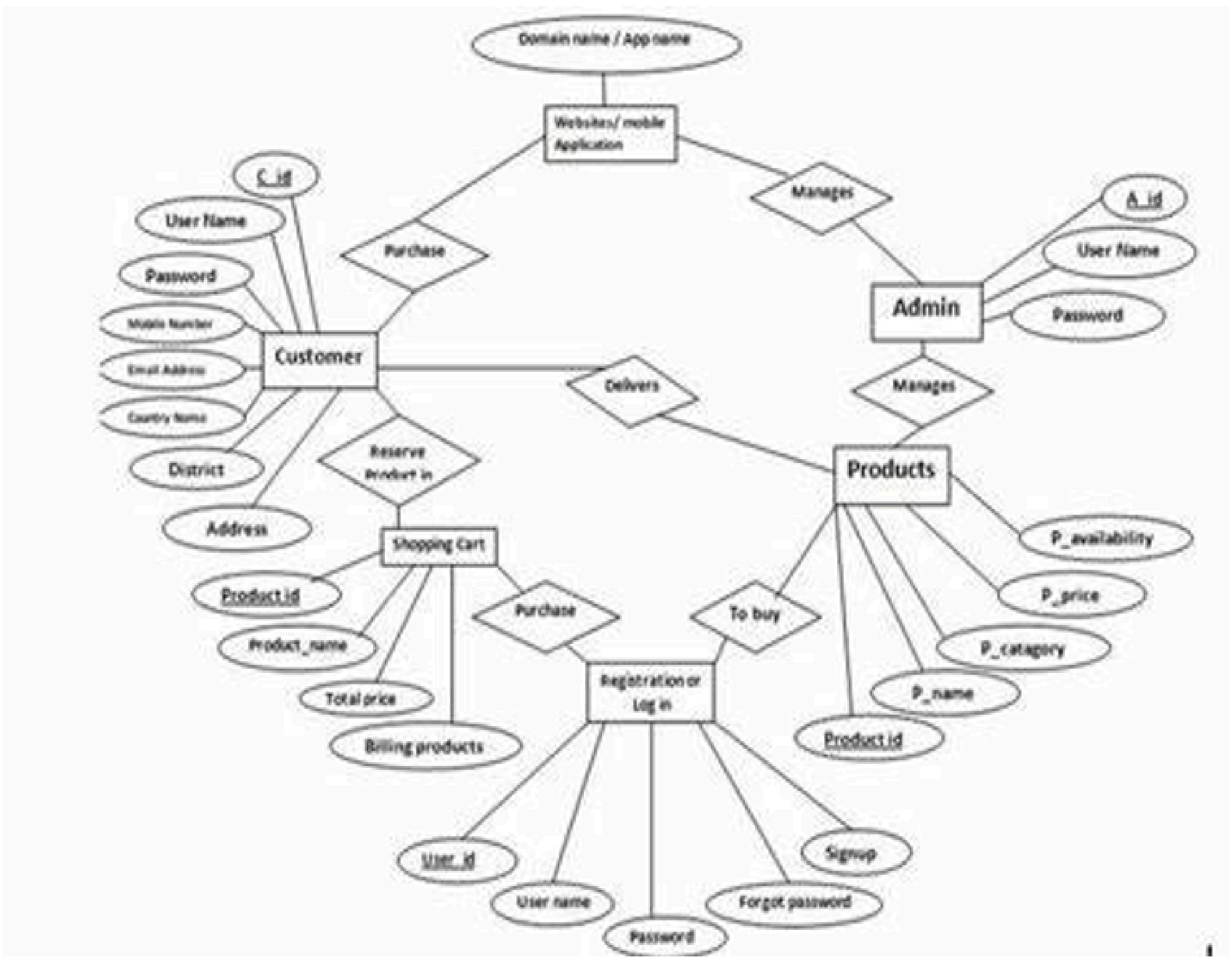
- Operating System Windows 10
- Front End python
- Back End MySQL
- search

Hardware Requirements

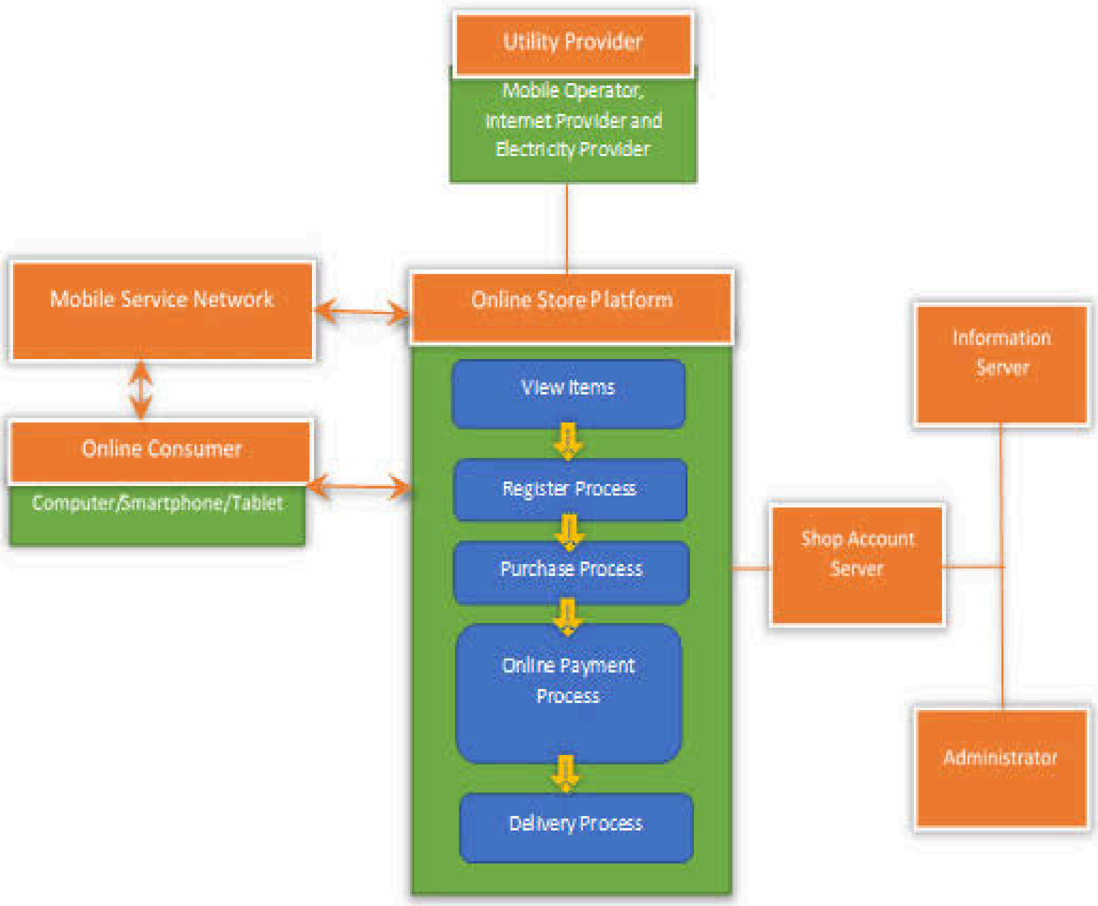
- Desktop PC or a Laptop

● Printer
● Operating System – Windows 10
● Intel® Core TM i3-6006U CPU @ 2.00GHz
● 4.00 GB RAM
● 64-bit operating system, x64 based processor
● 1024 x 768 monitor resolution
● Keyboard and Mouse

3.3 ER DIAGRAM



3.4. ARCHITECTURE DIAGRAM:



3.5. NORMALIZATION:

Normalization for the Online Store Database

Normalization is a process to organize the data in the database to minimize redundancy and improve data integrity. The normalization process involves dividing large tables into smaller, related tables and defining relationships between them. The most common normal forms are:

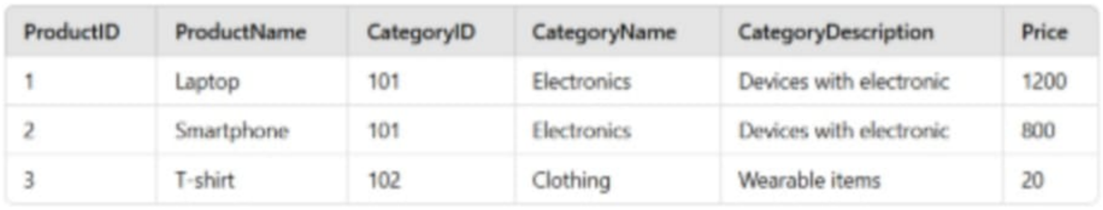
1. **First Normal Form (1NF)**: Ensures that each table column contains atomic (indivisible) values, and each record is unique.
2. **Second Normal Form (2NF):** Achieves 1NF and ensures that all non-key attributes are fully functionally dependent on the primary key.

3. **Third Normal Form (3NF):** Achieves 2NF and ensures that all the attributes are not only fully functionally dependent on the primary key but are also non-transitively dependent.

Example: Product and Category Tables

Initial Table (Unnormalized):

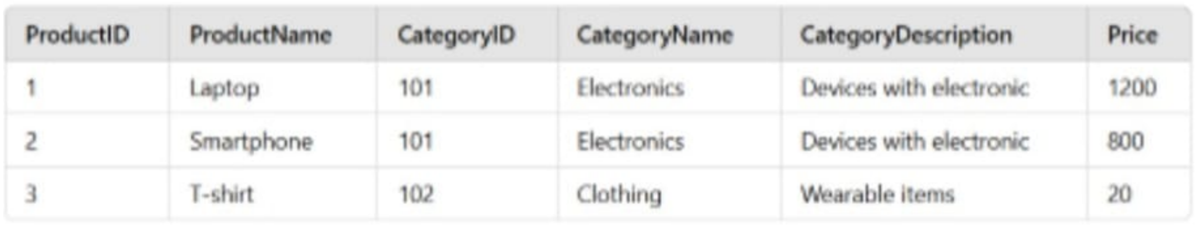
Let's consider a simple initial table that contains product information, including redundant category data:



ProductID	ProductName	CategoryID	CategoryName	CategoryDescription	Price
1	Laptop	101	Electronics	Devices with electronic	1200
2	Smartphone	101	Electronics	Devices with electronic	800
3	T-shirt	102	Clothing	Wearable items	20

First Normal Form (1NF):

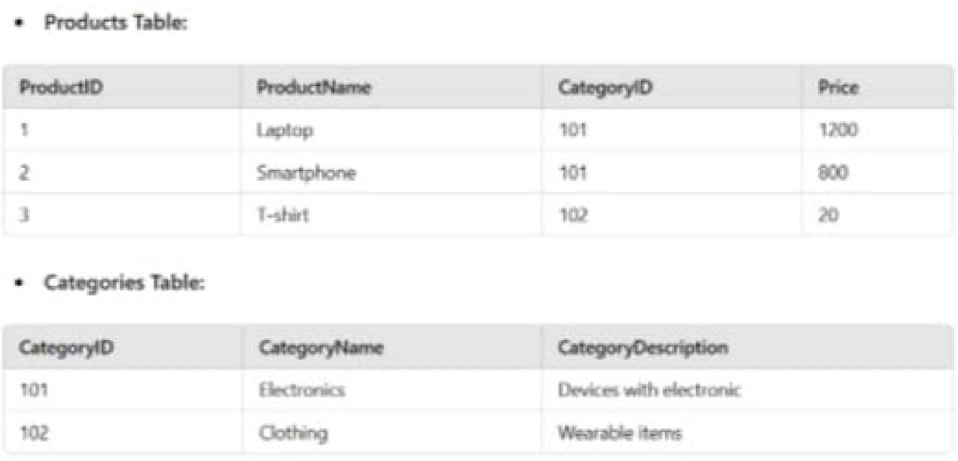
Ensure that each column contains atomic values and each record is unique.



ProductID	ProductName	CategoryID	CategoryName	CategoryDescription	Price
1	Laptop	101	Electronics	Devices with electronic	1200
2	Smartphone	101	Electronics	Devices with electronic	800
3	T-shirt	102	Clothing	Wearable items	20

Second Normal Form (2NF):

Remove partial dependencies; ensure that all non-key attributes are fully dependent on the primary key. Here, CategoryName and CategoryDescription depend on CategoryID, not ProductID.



- **Products Table:**

ProductID	ProductName	CategoryID	Price
1	Laptop	101	1200
2	Smartphone	101	800
3	T-shirt	102	20

- **Categories Table:**

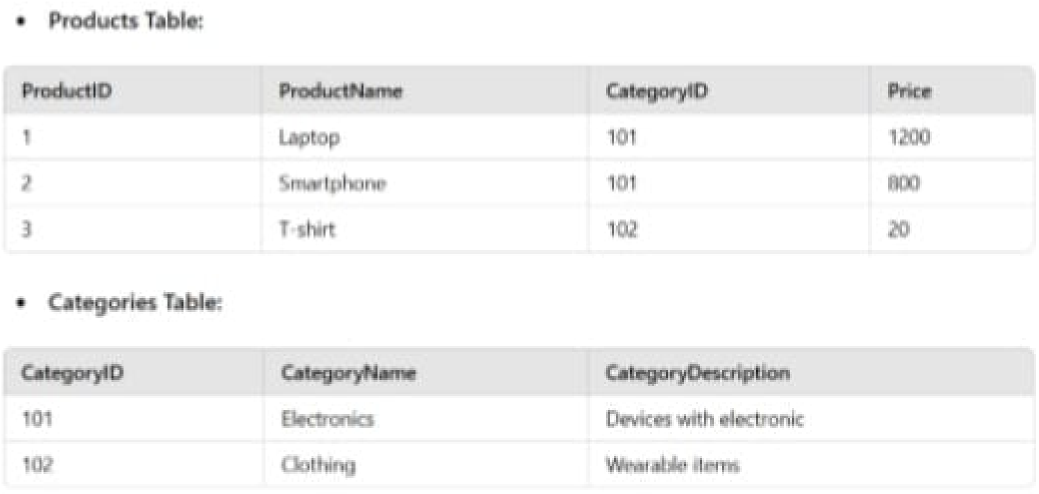
CategoryID	CategoryName	CategoryDescription
101	Electronics	Devices with electronic
102	Clothing	Wearable items

Third Normal Form (3NF):

Remove transitive dependencies; ensure that non-key attributes do not depend on other non-key attributes.

In this example, the tables already comply with 3NF because CategoryDescription depends only on CategoryID, and all attributes in the Products table depend only on ProductID.

Final Tables in 3NF:



- **Products Table:**

ProductID	ProductName	CategoryID	Price
1	Laptop	101	1200
2	Smartphone	101	800
3	T-shirt	102	20

- **Categories Table:**

CategoryID	CategoryName	CategoryDescription
101	Electronics	Devices with electronic
102	Clothing	Wearable items

Benefits of Normalization

- Reduced Redundancy: Eliminates duplicate data, saving storage space.
- Improved Data Integrity: Ensures data consistency and accuracy.
- Efficient Data Management: Simplifies data updates, insertions, and deletions.

Chapter 4

4. Program code

4.1.Login Page:

```
import tkinter as tk

import subprocess

from tkinter import messagebox

users = {"admin": "password123"}

def login():

    username = entry_username.get()

    password = entry_password.get()

    if username in users and users[username] == password:

        messagebox.showinfo("Login", "Login successful! Welcome to the Online
Store Database")

    else:

        messagebox.showerror("Login", "Invalid credentials, please try again.")

root = tk.Tk()

root.title("Login Page")

tk.Label(root, text="Username:").grid(row=0, column=0, padx=25, pady=25)

entry_username = tk.Entry(root)

entry_username.grid(row=0, column=1, padx=50, pady=50)

tk.Label(root, text="Password:").grid(row=1, column=0, padx=25, pady=25)

entry_password = tk.Entry(root, show="*")

entry_password.grid(row=1, column=1, padx=100, pady=100)

tk.Button(root, text="Login", command=login).grid(row=2, column=0,
columnspan=2, pady=50)

root.mainloop()

subprocess.run(["python","h2.py"])
```

4.2.Home Page:

```
import tkinter as tk
from tkinter import Toplevel, messagebox
from PIL import ImageTk, Image
def login():
    login_window = Toplevel(root)
    login_window.title("Login")
    tk.Label(login_window, text="Username:").pack(pady=50)
    tk.Entry(login_window).pack(pady=5)
    tk.Label(login_window, text="Password:").pack(pady=50)
    tk.Entry(login_window, show="*").pack(pady=50)
    tk.Button(login_window, text="Submit", command=lambda:
messagebox.showinfo("Login", "Login Successful")).pack(pady=10)
def register():
    register_window = Toplevel(root)
    register_window.title("Register")
    tk.Label(register_window, text="Username:").pack(pady=50)
    tk.Entry(register_window).pack(pady=5)
    tk.Label(register_window, text="Password:").pack(pady=50)
    tk.Entry(register_window, show="*").pack(pady=50)
    tk.Label(register_window, text="Confirm Password:").pack(pady=50)
    tk.Entry(register_window, show="*").pack(pady=50)
    tk.Button(register_window, text="Submit", command=lambda:
messagebox.showinfo("Register", "Registration Successful")).pack(pady=10)
def browse_products(*args):
    category = category_var.get()
      # Destroy previous product window if exists
```

```
 if hasattr(browse_products, "browse_window") and
browse_products.browse_window:
        browse_products.browse_window.destroy()
    browse_products.browse_window = Toplevel(root)
    browse_products.browse_window.title(f"Browse {category}")
products = []
 # Example: Define products with name, description, and image path
    if category == "Laptop":
        products = [
            {"name": "LENOVO", "description": "Empowering innovation through
reliable technology solutions.\nPROCESSOR:intel core i5\nStorage: 256GB
SSD.\nDisplay: 13.3-inch, 1080p resolution.\nPRICE:40,000","image_path":
"lenovo.png"},
            {"name": "HP", "description": "Unleash productivity and style with HP
laptops, seamlessly blending performance, reliability, and elegance for every
task.\nPROCESSOR:intel core i3\nStorage: 298GB SSD.\nDisplay: 11.3-inch,
1080p resolution\nPRICE:50,000", "image_path": "hp.png"},
            {"name": "DELL", "description": "Experience unrivaled performance
and reliability with Dell laptops, engineered to empower your digital lifestyle
with precision and innovation.\nPROCESSOR:intel core i5\nStorage: 295GB
SSD.\nDisplay: 13.3-inch, 1080p resolution\nPRICE:50,000", "image_path":
"dell.png"}
        ]
    elif category == "Mobile Phone":
        products = [
            {"name": "REDMI", "description": "Experience innovation in the palm
of your hand with the Redmi mobile, delivering cutting-edge features at an
unbeatable value.\nPRICE:12,000\nSTORAGE:128GB\nRAM:6GB",
"image_path": "redmi.png"},
            {"name": "OPPO", "description": "Elevate your mobile experience with
Oppo: sleek design, cutting-edge technology, and stunning performance in every
touch.\nPRICE:15,000\nSTORAGE:128GB\nRAM:8GB", "image_path":
"oppo.png"},
            {"name": "VIVO", "description": "Explore the perfect blend of style and performance with Vivo mobiles, designed to elevate your smartphone experience to new heights.\nPRICE:16,000\nSTORAGE:117GB\nRAM:12GB", "image_path": "vivo.png"}
        ]
    elif category == "Watch":
        products = [
            {"name": "SMARTWATCH", "description": "A smartwatch is a wearable device that offers fitness tracking, notifications, and app functionality, usually paired with a smartphone.\nPRICE:1500\nWARRENTY:2YR\nCOLOUR:PINK", "image_path": "smart_watch.png"},
            {"name": "WRIST WATCH", "description": "Discover elegance and functionality with our timeless wristwatch, blending classic design with modern precision.\nPRICE:350\nWARRENTY:1YR\nCOLOUR:PURPLE", "image_path": "wrist_watch.png"},
            {"name": "STOPWATCH", "description": "Precision at your fingertips, perfect for timing any activity with accuracy and ease.\nPRICE:500\nWARRENTY:1.5YR\nCOLOUR:BLACK", "image_path": "stop_watch.png"}
        ]
 if not products:
        tk.Label(browse_products.browse_window, text="No products available in this category").pack(pady=10)
    else:
        for product in products:
            product_frame = tk.Frame(browse_products.browse_window)
            product_frame.pack(pady=10)
     try:
                img = Image.open(product["image_path"])
                img = img.resize((100, 100))
                photo_img = ImageTk.PhotoImage(img)
                image_label = tk.Label(product_frame, image=photo_img)
                image_label.image = photo_img
                image_label.pack(side=tk.LEFT, padx=10)
            except Exception as e:
                print(f"Error loading image {product['image_path']}: {e}")
                tk.Label(product_frame, text="Image not 
available").pack(side=tk.LEFT, padx=10)
    info_label = tk.Label(product_frame, 
text=f"{product['name']}\n{product['description']}")
            info_label.pack(side=tk.LEFT)
def search_products():
    search_window = Toplevel(root)
    search_window.title("Search Products")
    tk.Label(search_window, text="Search for a product:").pack(pady=50)
    search_entry = tk.Entry(search_window)
    search_entry.pack(pady=50)
    def show_search_results():
        query = search_entry.get().lower()
        search_results = []
        # Example: Define products with name, description, and image path
        products = [
            {"name": "LENOVO", "description": "Empowering innovation through 
reliable technology solutions.\nPROCESSOR:intel core i5\nStorage: 256GB 
SSD.\nDisplay: 13.3-inch, 1080p resolution.\nPRICE:40000", "image_path": 
"lenovo.png"},
            {"name": "HP", "description": "Unleash productivity and style with HP 
laptops, seamlessly blending performance, reliability, and elegance for every 
task.\nPROCESSOR:intel core i3\nStorage: 298GB SSD.\nDisplay: 11.3-inch, 
1080p resolution.\nPRICE:50000", "image_path": "hp.png"},
            {"name": "DELL", "description": "Experience unrivaled performance and reliability with Dell laptops, engineered to empower your digital lifestyle with precision and innovation.\nPROCESSOR:intel core i5\nStorage: 295GB SSD.\nDisplay: 13.3-inch, 1080p resolution.\nPRICE:50000", "image_path": "dell.png"}
        ]
        # Filter products based on search query
        search_results = [product for product in products if query in product["name"].lower()]
        if not search_results:
            tk.Label(search_window, text="No products found for the search query.").pack(pady=10)
        else:
            for product in search_results:
                product_frame = tk.Frame(search_window)
                product_frame.pack(pady=10)


                try:
                    img = Image.open(product["image_path"])
                    img = img.resize((100, 100))
                    photo_img = ImageTk.PhotoImage(img)


                    image_label = tk.Label(product_frame, image=photo_img)
                    image_label.image = photo_img
                    image_label.pack(side=tk.LEFT, padx=10)
                except Exception as e:
                    print(f"Error loading image {product['image_path']}: {e}")
                    tk.Label(product_frame, text="Image not available").pack(side=tk.LEFT, padx=10)
                info_label = tk.Label(product_frame,
text=f"{product['name']}\n{product['description']}")
                info_label.pack(side=tk.LEFT)
    tk.Button(search_window, text="Search",
command=show_search_results).pack(pady=10)
def add_to_cart(product):
    cart.append(product)
    checkout_items.append(product)
    messagebox.showinfo("Cart", f"{product['name']} added to cart!")
def view_cart():
    cart_window = Toplevel(root)
    cart_window.title("View Cart")
    if not cart:
        tk.Label(cart_window, text="Your cart is empty").pack(pady=10)
    else:
        for item in cart:
            tk.Label(cart_window, text=item['name']).pack(pady=5)
            tk.Label(cart_window, text=item['description']).pack(pady=5)
            tk.Label(cart_window, text="------------------------").pack(pady=5)
def checkout():
    checkout_window = Toplevel(root)
    checkout_window.title("Checkout")
    if not checkout_items:
        tk.Label(checkout_window, text="Your cart is empty").pack(pady=10)
    else:
        tk.Label(checkout_window, text="Cart Items:").pack(pady=10)
        for item in checkout_items:
            item_frame = tk.Frame(checkout_window)
            item_frame.pack(pady=5)
            item_label = tk.Label(item_frame, text=f"{item['name']} -
{item['description']}")
            item_label.pack(side=tk.LEFT)
            price_label = tk.Label(item_frame, text=f"Price: {item['Price']}")
            price_label.pack(side=tk.RIGHT)
        total_label = tk.Label(checkout_window, text=f"Total: {sum(item['Price']
for item in checkout_items)}")
        total_label.pack(pady=10)
        checkout_button = tk.Button(checkout_window, text="Finalize Purchase",
command=finalize_purchase)
        checkout_button.pack(pady=10)
def finalize_purchase():
    # Placeholder function for finalizing the purchase
    messagebox.showinfo("Purchase Complete", "Thank you for your
purchase!")
def update_image():
    new_image_path = "new_image.png"
    img = Image.open(new_image_path)
    img = img.resize((1000, 500))
    photo_img = ImageTk.PhotoImage(img)
    image_label.config(image=photo_img)
    image_label.image = photo_img
root = tk.Tk()
root.title("Online Store")
checkout_items = []
cart = []
product1 = {"name": "LENOVO", "description": "Redefining innovation with
style and performance at your fingertips!","Price":40000}
```

```
add_to_cart(product1)

product2 = {"name": "REDMI", "description": "Experience innovation in the
palm of your hand with the Redmi mobile.","Price":12000}

add_to_cart(product2)

login_img = Image.open("login.png")

login_img = login_img.resize((50, 50))

login_img = ImageTk.PhotoImage(login_img)

register_img = Image.open("register.png")

register_img = register_img.resize((50, 50))

register_img = ImageTk.PhotoImage(register_img)

title_label = tk.Label(root, text="Welcome to Our Online Store",
font=("Helvetica", 20))

title_label.pack(pady=10)

default_img = Image.open("default_image.png")

default_img = default_img.resize((1000, 500))

default_photo_img = ImageTk.PhotoImage(default_img)

image_label = tk.Label(root, image=default_photo_img)

image_label.pack(pady=10)

login_button = tk.Button(root, text="Login", image=login_img,
compound=tk.RIGHT, command=login)

login_button.pack(pady=10)

register_button = tk.Button(root, text="Register", image=register_img,
compound=tk.LEFT, command=register)

register_button.pack(pady=10)

categories_frame = tk.Frame(root)

categories_frame.pack(pady=10)

categories_label = tk.Label(categories_frame, text="Select a Category:")

categories_label.grid(row=0, column=0)
```

```
category_var = tk.StringVar()

category_var.set("---")

categories_menu = tk.OptionMenu(categories_frame, category_var, "Laptop",
"Mobile Phone", "Watch", command=browse_products)

categories_menu.grid(row=0, column=1)

search_button = tk.Button(root, text="Search Products", width=20,
command=search_products)

search_button.pack(pady=10)

view_cart_button = tk.Button(root, text="View Cart", width=20,
command=view_cart)

view_cart_button.pack(pady=10)

checkout_button = tk.Button(root, text="Checkout", width=20,
command=checkout)

checkout_button.pack(pady=10)

update_image_button = tk.Button(root, text="Update Image",
command=update_image)

update_image_button.pack(pady=10)

root.mainloop()
```

4.3.Connectivity code:

```
import tkinter as tk

from tkinter import ttk, messagebox

import mysql.connector

from mysql.connector import errorcode

# Database connection details

try:

   connection=
mysql.connector.connect(host="localhost",user="root",password="Swetha@298
",database="dbms" )

   cursor = connection.cursor()
```

```
except mysql.connector.Error as err:
    if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
        print("Something is wrong with your user_name or password")
    elif err.errno == errorcode.ER_BAD_DB_ERROR:
        print("Database does not exist")
    else:
        print(err)
def create_user():
    username = username_entry.get()
    email = email_entry.get()
    password = password_entry.get()
    try:
        cursor.callproc("create_user", [username, email, password])
        connection.commit()
        messagebox.showinfo("Success", "User created successfully")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to create user: {err}")
    finally:
        clear_entries(user_entries)
def read_user():
    user_id = user_id_entry.get()
    if not user_id:
        messagebox.showerror("Error", "User ID is required")
        return
    try:
        cursor.execute("SELECT * FROM get_user(%s)", (user_id,))
        user_data = cursor.fetchone()
        if user_data:
            messagebox.showinfo("User Data", f"User ID:
{user_data[0]}\nUsername: {user_data[1]}\nEmail: {user_data[2]}")
        else:
            messagebox.showerror("Error", "User not found")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to retrieve user: {err}")



def update_user():
    user_id = user_id_entry.get()
    if not user_id:
        messagebox.showerror("Error", "User ID is required")
        return

    new_username = new_username_entry.get()
    new_email = new_email_entry.get()
    new_password = new_password_entry.get()

    try:
        cursor.callproc("update_user", [user_id, new_username, new_email,
new_password])
        connection.commit()
        messagebox.showinfo("Success", "User updated successfully")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to update user: {err}")
    finally:
        clear_entries(user_entries)
```

```
def delete_user():
    user_id = user_id_entry.get()
    if not user_id:
        messagebox.showerror("Error", "User ID is required")
        return

    try:
        cursor.callproc("delete_user", [user_id])
        connection.commit()
        messagebox.showinfo("Success", "User deleted successfully")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to delete user: {err}")
    finally:
        clear_entries(user_entries)


def create_product():
    name = name_entry.get()
    description = description_entry.get()
    price = float(price_entry.get())
    stock_quantity = int(stock_quantity_entry.get())
    category_id = int(category_id_entry.get())

    try:
        cursor.callproc("create_product", [name, description, price, stock_quantity,
category_id])
        connection.commit()
        messagebox.showinfo("Success", "Product created successfully")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to create product: {err}")
    finally:
        clear_entries(product_entries)



def read_product():
    product_id = product_id_entry.get()
    if not product_id:
        messagebox.showerror("Error", "Product ID is required")
        return

    try:
        cursor.execute("SELECT * FROM get_product(%s)", (product_id,))
        product_data = cursor.fetchone()
        if product_data:
            messagebox.showinfo("Product Data", f"Product ID:
{product_data[0]}\nName: {product_data[1]}\nDescription:
{product_data[2]}\nPrice: {product_data[3]}\nStock Quantity:
{product_data[4]}\nCategory ID: {product_data[5]}")
        else:
            messagebox.showerror("Error", "Product not found")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to retrieve product: {err}")



def update_product():
    product_id = product_id_entry.get()
    if not product_id:
        messagebox.showerror("Error", "Product ID is required")
        return

    new_name = new_name_entry.get()
    new_description = new_description_entry.get()
    new_price = float(new_price_entry.get())
    new_stock_quantity = int(new_stock_quantity_entry.get())
    new_category_id = int(new_category_id_entry.get())

    try:
        cursor.callproc("update_product", [product_id, new_name, 
new_description, new_price, new_stock_quantity, new_category_id])
        connection.commit()
        messagebox.showinfo("Success", "Product updated successfully")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to update product: {err}")
    finally:
        clear_entries(product_entries)


def delete_product():
    product_id = product_id_entry.get()
    if not product_id:
        messagebox.showerror("Error", "Product ID is required")
        return
    try:
        cursor.callproc("delete_product", [product_id])
        connection.commit()
        messagebox.showinfo("Success", "Product deleted successfully")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to delete product: {err}")
    finally:
        clear_entries(product_entries)


def create_order():
    user_id = order_user_id_entry.get()
    product_id = order_product_id_entry.get()
    quantity = int(order_quantity_entry.get())
    order_date = order_date_entry.get()

    try:
        cursor.callproc("create_order", [user_id, product_id, quantity, order_date])
        connection.commit()
        messagebox.showinfo("Success", "Order created successfully")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to create order: {err}")
    finally:
        clear_entries(order_entries)


def read_order():
    order_id = order_id_entry.get()
    if not order_id:
        messagebox.showerror("Error", "Order ID is required")
        return

    try:
        cursor.execute("SELECT * FROM get_order(%s)", (order_id,))
        order_data = cursor.fetchone()
        if order_data:
            messagebox.showinfo("Order Data", f"Order ID: {order_data[0]}\nUser
ID: {order_data[1]}\nProduct ID: {order_data[2]}\nQuantity:
{order_data[3]}\nOrder Date: {order_data[4]}")
        else:
            messagebox.showerror("Error", "Order not found")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to retrieve order: {err}")


def update_order():
    order_id = order_id_entry.get()
    if not order_id:
        messagebox.showerror("Error", "Order ID is required")
        return

    new_user_id = new_order_user_id_entry.get()
    new_product_id = new_order_product_id_entry.get()
    new_quantity = int(new_order_quantity_entry.get())
    new_order_date = new_order_date_entry.get()
    try:
        cursor.callproc("update_order", [order_id, new_user_id, new_product_id,
new_quantity, new_order_date])
        connection.commit()
        messagebox.showinfo("Success", "Order updated successfully")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to update order: {err}")
    finally:
        clear_entries(order_entries)


def delete_order():
    order_id = order_id_entry.get()
    if not order_id:
        messagebox.showerror("Error", "Order ID is required")
        return

    try:
        cursor.callproc("delete_order", [order_id])
        connection.commit()
        messagebox.showinfo("Success", "Order deleted successfully")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to delete order: {err}")
    finally:
        clear_entries(order_entries)
```

```
def clear_entries(entries):
    for entry in entries:
        entry.delete(0, tk.END)



# Create main window
root = tk.Tk()
root.title("Online Store Management")

# Create notebook
notebook = ttk.Notebook(root)
notebook.pack(fill=tk.BOTH, expand=True)

# Create frames
user_frame = ttk.Frame(notebook)
product_frame = ttk.Frame(notebook)
order_frame = ttk.Frame(notebook)

# Add frames to notebook
notebook.add(user_frame, text="User Management")
notebook.add(product_frame, text="Product Management")
notebook.add(order_frame, text="Order Management")

tk.Label(product_frame, text="Name:").grid(row=0, column=0, padx=5,
pady=5)
name_entry = tk.Entry(product_frame)
name_entry.grid(row=0, column=1, padx=5, pady=5)
```

```
tk.Label(product_frame, text="Description:").grid(row=1, column=0, padx=5,
pady=5)
description_entry = tk.Entry(product_frame)
description_entry.grid(row=1, column=1, padx=5, pady=5)

tk.Label(product_frame, text="Price:").grid(row=2, column=0, padx=5,
pady=5)
price_entry = tk.Entry(product_frame)
price_entry.grid(row=2, column=1, padx=5, pady=5)

tk.Label(product_frame, text="Stock Quantity:").grid(row=3, column=0,
padx=5, pady=5)
stock_quantity_entry = tk.Entry(product_frame)
stock_quantity_entry.grid(row=3, column=1, padx=5, pady=5)

tk.Label(product_frame, text="Category ID:").grid(row=4, column=0, padx=5,
pady=5)
category_id_entry = tk.Entry(product_frame)
category_id_entry.grid(row=4, column=1, padx=5, pady=5)

tk.Label(product_frame, text="Product ID:").grid(row=5, column=0, padx=5,
pady=5)
product_id_entry = tk.Entry(product_frame)
product_id_entry.grid(row=5, column=1, padx=5, pady=5)

tk.Label(product_frame, text="New Name:").grid(row=6, column=0, padx=5,
pady=5)
new_name_entry = tk.Entry(product_frame)
new_name_entry.grid(row=6, column=1, padx=5, pady=5)
```

```
tk.Label(product_frame, text="New Description:").grid(row=7, column=0,
padx=5, pady=5)

new_description_entry = tk.Entry(product_frame)

new_description_entry.grid(row=7, column=1, padx=5, pady=5)


tk.Label(product_frame, text="New Price:").grid(row=8, column=0, padx=5,
pady=5)

new_price_entry = tk.Entry(product_frame)

new_price_entry.grid(row=8, column=1, padx=5, pady=5)


tk.Label(product_frame, text="New Stock Quantity:").grid(row=9, column=0,
padx=5, pady=5)

new_stock_quantity_entry = tk.Entry(product_frame)

new_stock_quantity_entry.grid(row=9, column=1, padx=5, pady=5)


tk.Label(product_frame, text="New Category ID:").grid(row=10, column=0,
padx=5, pady=5)

new_category_id_entry = tk.Entry(product_frame)

new_category_id_entry.grid(row=10, column=1, padx=5, pady=5)


product_buttons = ttk.Frame(product_frame)

product_buttons.grid(row=7, columnspan=2, pady=10)


tk.Button(product_buttons, text="Create Product",
command=create_product).grid(row=0, column=0, padx=5, pady=5)

tk.Button(product_buttons, text="Read Product",
command=read_product).grid(row=0, column=1, padx=5, pady=5)

tk.Button(product_buttons, text="Update Product",
command=update_product).grid(row=0, column=2, padx=5, pady=5)
```

```
tk.Button(product_buttons, text="Delete Product",
command=delete_product).grid(row=0, column=3, padx=5, pady=5)

# User Management Frame
tk.Label(user_frame, text="Username:").grid(row=0, column=0, padx=5,
pady=5)
username_entry = tk.Entry(user_frame)
username_entry.grid(row=0, column=1, padx=5, pady=5)

tk.Label(user_frame, text="Email:").grid(row=1, column=0, padx=5, pady=5)
email_entry = tk.Entry(user_frame)
email_entry.grid(row=1, column=1, padx=5, pady=5)

tk.Label(user_frame, text="Password:").grid(row=2, column=0, padx=5,
pady=5)
password_entry = tk.Entry(user_frame, show='*')
password_entry.grid(row=2, column=1, padx=5, pady=5)

tk.Label(user_frame, text="User ID:").grid(row=3, column=0, padx=5, pady=5)
user_id_entry = tk.Entry(user_frame)
user_id_entry.grid(row=3, column=1, padx=5, pady=5)

tk.Label(user_frame, text="New Username:").grid(row=4, column=0, padx=5,
pady=5)
new_username_entry = tk.Entry(user_frame)
new_username_entry.grid(row=4, column=1, padx=5, pady=5)

tk.Label(user_frame, text="New Email:").grid(row=5, column=0, padx=5,
pady=5)
```

```
new_email_entry = tk.Entry(user_frame)
new_email_entry.grid(row=5, column=1, padx=5, pady=5)

tk.Label(user_frame, text="New Password:").grid(row=6, column=0, padx=5,
pady=5)
new_password_entry = tk.Entry(user_frame, show='*')
new_password_entry.grid(row=6, column=1, padx=5, pady=5)

user_buttons = ttk.Frame(user_frame)
user_buttons.grid(row=7, columnspan=2, pady=10)

tk.Button(user_buttons, text="Create User",
command=create_user).grid(row=0, column=0, padx=5, pady=5)
tk.Button(user_buttons, text="Read User", command=read_user).grid(row=0,
column=1, padx=5, pady=5)
tk.Button(user_buttons, text="Update User",
command=update_user).grid(row=0, column=2, padx=5, pady=5)
tk.Button(user_buttons, text="Delete User",
command=delete_user).grid(row=0, column=3, padx=5, pady=5)


tk.Label(order_frame, text="User ID:").grid(row=0, column=0, padx=5,
pady=5)
order_user_id_entry = tk.Entry(order_frame)
order_user_id_entry.grid(row=0, column=1, padx=5, pady=5)

tk.Label(order_frame, text="Product ID:").grid(row=1, column=0, padx=5,
pady=5)
order_product_id_entry = tk.Entry(order_frame)
order_product_id_entry.grid(row=1, column=1, padx=5, pady=5)
```

```
tk.Label(order_frame, text="Quantity:").grid(row=2, column=0, padx=5,
pady=5)
order_quantity_entry = tk.Entry(order_frame)
order_quantity_entry.grid(row=2, column=1, padx=5, pady=5)

tk.Label(order_frame, text="Order Date:").grid(row=3, column=0, padx=5,
pady=5)
order_date_entry = tk.Entry(order_frame)
order_date_entry.grid(row=3, column=1, padx=5, pady=5)

tk.Label(order_frame, text="Order ID:").grid(row=4, column=0, padx=5,
pady=5)
order_id_entry = tk.Entry(order_frame)
order_id_entry.grid(row=4, column=1, padx=5, pady=5)

tk.Label(order_frame, text="New User ID:").grid(row=5, column=0, padx=5,
pady=5)
new_order_user_id_entry = tk.Entry(order_frame)
new_order_user_id_entry.grid(row=5, column=1, padx=5, pady=5)

tk.Label(order_frame, text="New Product ID:").grid(row=6, column=0,
padx=5, pady=5)
new_order_product_id_entry = tk.Entry(order_frame)
new_order_product_id_entry.grid(row=6, column=1, padx=5, pady=5)

tk.Label(order_frame, text="New Quantity:").grid(row=7, column=0, padx=5,
pady=5)
new_order_quantity_entry = tk.Entry(order_frame)
```

```
new_order_quantity_entry.grid(row=7, column=1, padx=5, pady=5)


tk.Label(order_frame, text="New Order Date:").grid(row=8, column=0,
padx=5, pady=5)

new_order_date_entry = tk.Entry(order_frame)

new_order_date_entry.grid(row=8, column=1, padx=5, pady=5)


order_buttons = ttk.Frame(order_frame)

order_buttons.grid(row=7, columnspan=2, pady=10)


tk.Button(order_buttons, text="Create Order",
command=create_order).grid(row=0, column=0, padx=5, pady=5)

tk.Button(order_buttons, text="Read Order",
command=read_order).grid(row=0, column=1, padx=5, pady=5)

tk.Button(order_buttons, text="Update Order",
command=update_order).grid(row=0, column=2, padx=5, pady=5)

tk.Button(order_buttons, text="Delete Order",
command=delete_order).grid(row=0, column=3, padx=5, pady=5)



product_entries = [name_entry, description_entry, price_entry,
stock_quantity_entry, category_id_entry, product_id_entry, new_name_entry,
new_description_entry, new_price_entry, new_stock_quantity_entry,
new_category_id_entry]

order_entries = [order_user_id_entry, order_product_id_entry,
order_quantity_entry, order_date_entry, order_id_entry,
new_order_user_id_entry, new_order_product_id_entry,
new_order_quantity_entry, new_order_date_entry]

user_entries = [username_entry, email_entry, password_entry, user_id_entry,
new_username_entry, new_email_entry, new_password_entry]
```

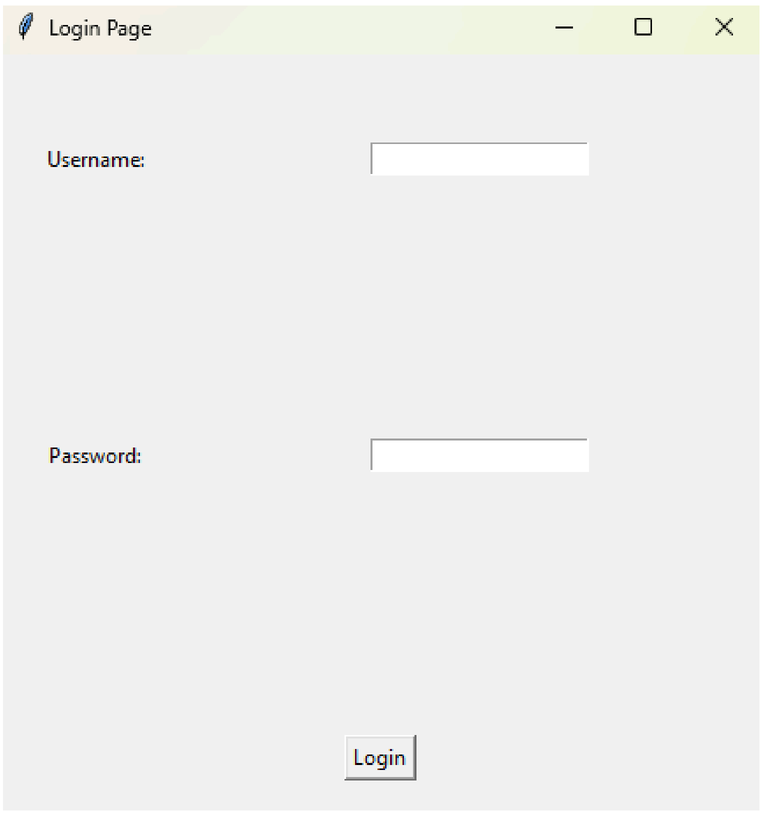
```
# Search Feature
def search():
    search_term = search_entry.get()
    if search_option.get() == "Users":
        cur.execute("SELECT * FROM Users WHERE username LIKE ? OR
email LIKE ?", ('%' + search_term + '%', '%' + search_term + '%'))
    elif search_option.get() == "Products":
        cur.execute("SELECT * FROM Products WHERE name LIKE ? OR
description LIKE ?", ('%' + search_term + '%', '%' + search_term + '%'))
    elif search_option.get() == "Orders":
        cur.execute("SELECT * FROM Orders WHERE user_id LIKE ? OR
product_id LIKE ?", ('%' + search_term + '%', '%' + search_term + '%'))
    else:
        result_text.set("Invalid search option selected.")
        return
    results = cur.fetchall()
    result_text.set(results)

search_frame = ttk.LabelFrame(root, text="Search")
search_frame.pack(fill="x", padx=10, pady=10)

tk.Label(search_frame, text="Search Term:").grid(row=0, column=0, padx=5,
pady=5)
search_entry = tk.Entry(search_frame)
search_entry.grid(row=0, column=1, padx=5, pady=5)

search_option = tk.StringVar(value="Users")
search_menu = ttk.OptionMenu(search_frame, search_option, "Users", "Users",
"Products", "Orders")
```

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


tk.Button(search_frame, text="Search", command=search).grid(row=0,
column=3, padx=5, pady=5)


result_text = tk.StringVar()
tk.Label(search_frame, textvariable=result_text).grid(row=1, columnspan=4,
padx=5, pady=5)


root.mainloop()
```

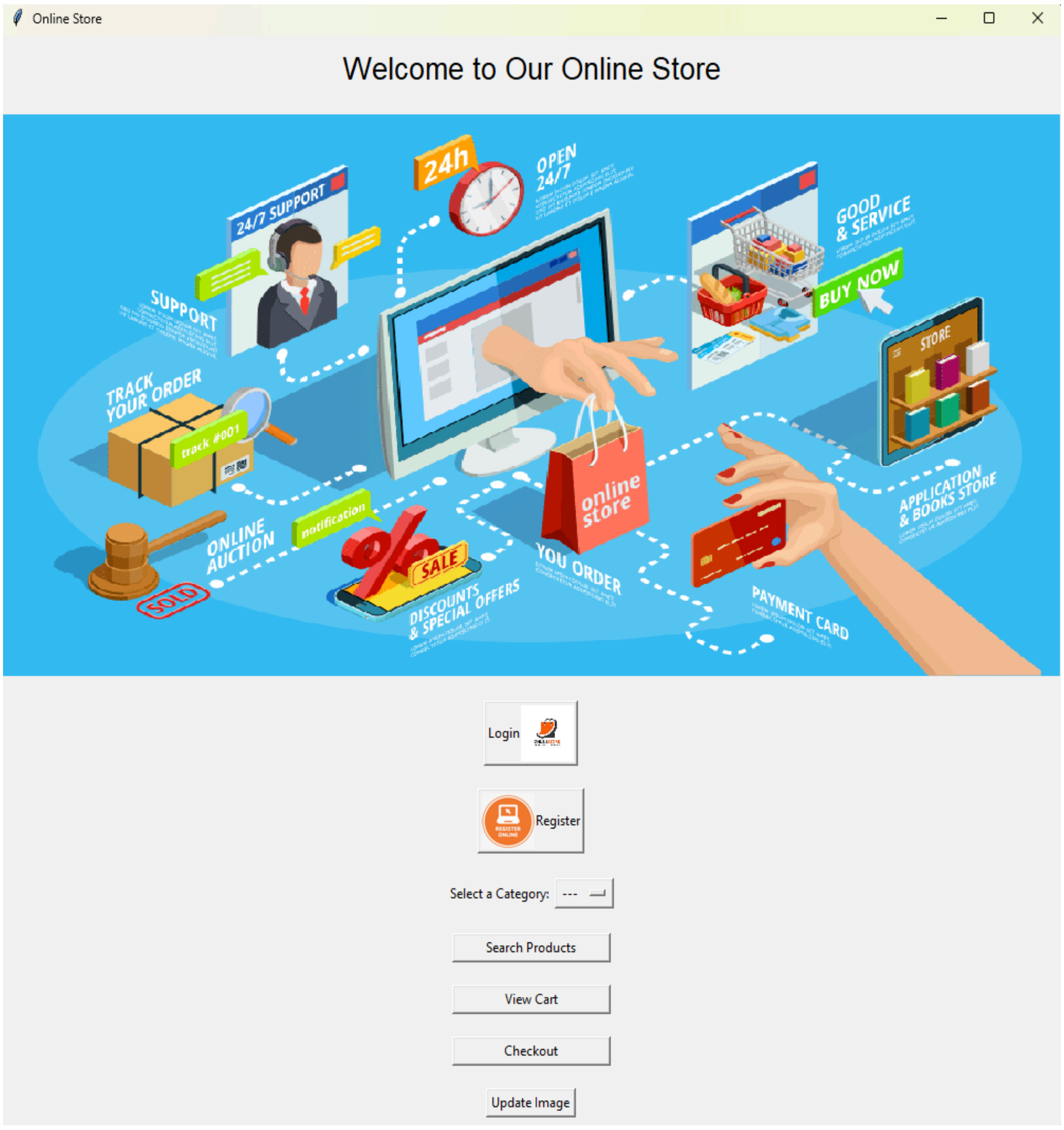
RESULT AND DISCUSSION

5.1 USER DOCUMENTATION

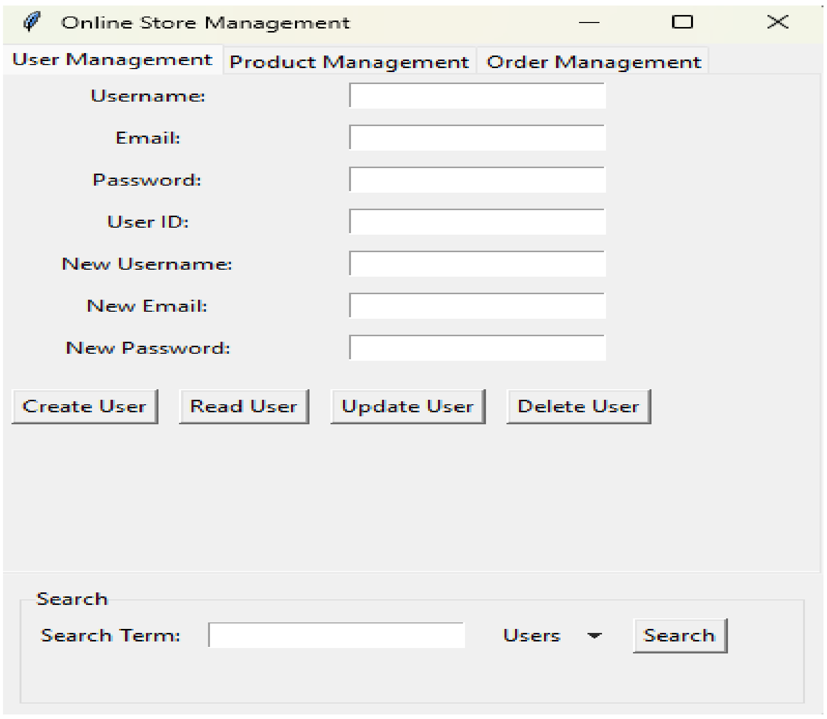
5.1.1.Login Page



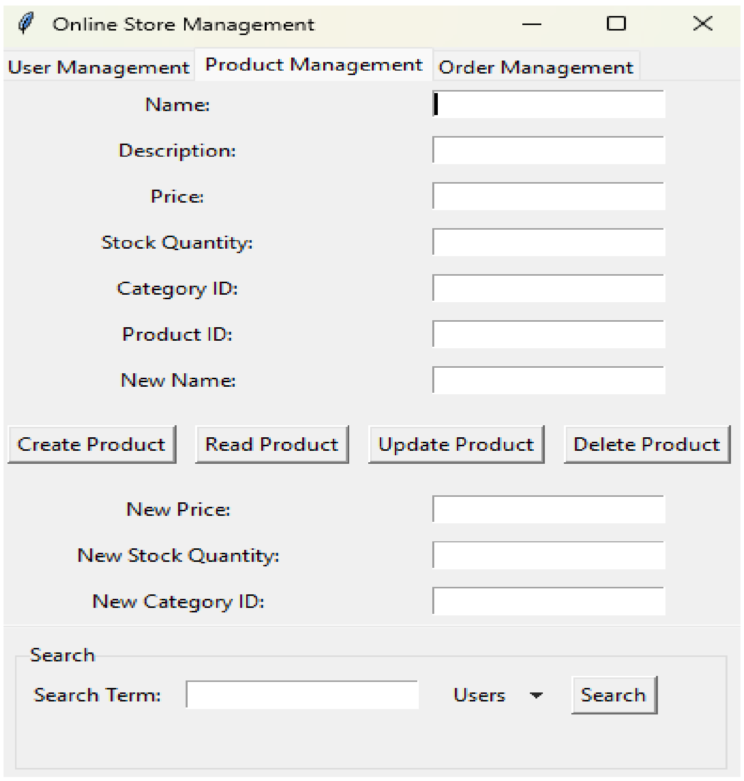
5.1.2.Home Page



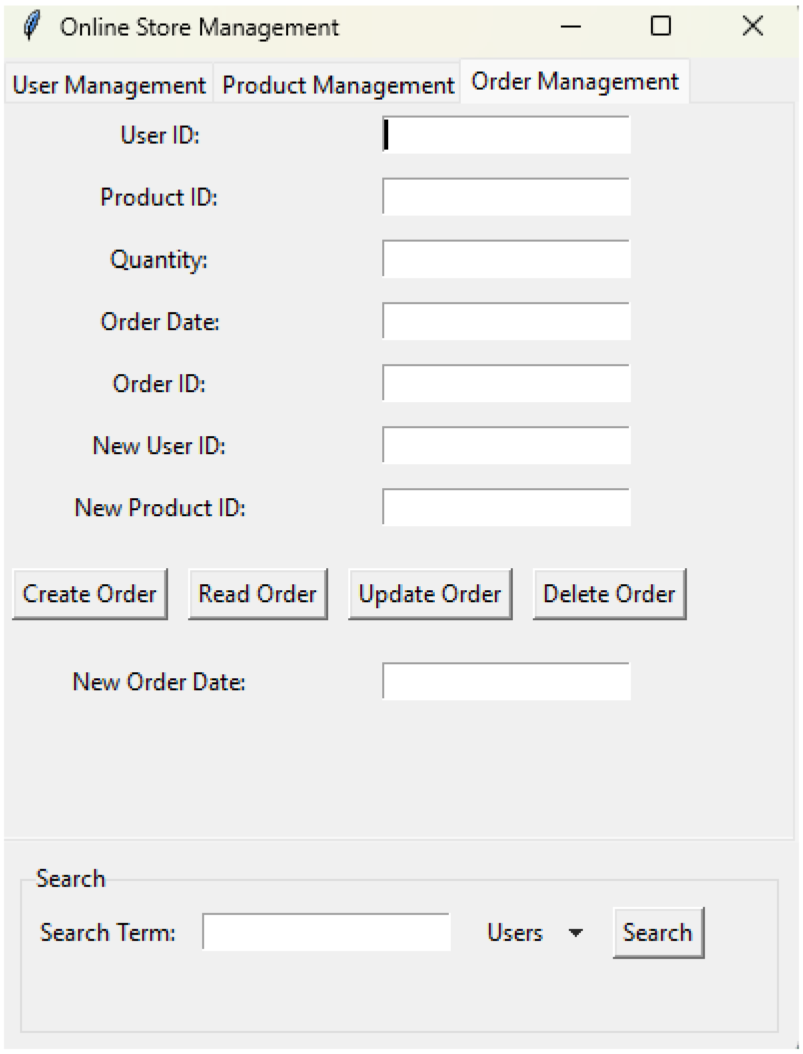
5.1.3.User Management



5.1.4.Product Management



5.1.5.Order Management



5.Conclusion:

The Online Store Database Project has been meticulously designed to provide a robust, scalable, and secure solution for managing an e-commerce platform's extensive data needs. Through a structured and methodical approach, the project encompasses critical aspects such as product management, user account handling, order processing, payment integration, and inventory control. By addressing these fundamental components, the project ensures that both customers and administrators can interact with the platform efficiently and securely, fostering a seamless online shopping experience.

One of the key achievements of this project is the implementation of a comprehensive product management system that not only stores detailed product information but also ensures real-time inventory tracking and automated restock alerts. This capability significantly reduces the manual workload for administrators and minimizes the risk of stockouts, thereby enhancing customer satisfaction. Additionally, the secure user authentication and account management features protect sensitive user data while providing a personalized shopping experience.

The integration of secure payment gateways and efficient order processing mechanisms further underscores the project's commitment to providing a reliable and trustworthy platform. Customers can complete transactions with confidence, knowing their payment information is protected through advanced encryption and security protocols. Meanwhile, administrators benefit from streamlined order management processes that facilitate quick and accurate order fulfillment, contributing to overall operational efficiency.

Finally, the project’s emphasis on performance, scalability, and data integrity ensures that the online store can grow and adapt to increasing demands without compromising on quality or security. In conclusion, the Online Store Database Project lays a solid foundation for a thriving e-commerce platform, capable of supporting long-term business growth and success in the competitive digital marketplace.

References for the Online Store Database Project:

1. **Elmasri, R., & Navathe, S. B. (2016). Fundamentals of Database Systems (7th ed.). Pearson.**
    - This book provides a comprehensive introduction to database concepts, design, and applications. It covers database models, SQL, and database design techniques, making it an essential resource for understanding the fundamentals required for the online store database project.

2. **Connolly, T., & Begg, C. (2015). Database Systems: A Practical Approach to Design, Implementation, and Management (6th ed.). Pearson.**
    - This textbook is a practical guide to database design and management. It includes case studies and practical examples that can be directly applied to designing and implementing the online store database.

3. **Coronel, C., & Morris, S. (2019). Database Systems: Design, Implementation, & Management (13th ed.). Cengage Learning.**
    - This book offers in-depth coverage of database design, SQL, and data management. It is particularly useful for understanding the implementation and maintenance of a robust database system for an online store.

These references provide a solid foundation in database design, implementation, and management, essential for the successful completion of the online store database project.