Acknowledgment

First and foremost I would like to thank God Almighty for showering us with all the his blessings for completion of this project successfully.

I also express my humble gratitude to my computer science teacher Mrs. Shiny Harris for her constant support and cooperation in clearing all doubts and for the inspiration received during the course of this project.

I sincerely acknowledge the goodwill and enthusiasm shown by our lab assistant Mr. Jobin George by providing all necessary helps without any delay.

I also thank our principal Mrs. Sonia Susan Varkey and Vice Principal Mrs. Anju Priya B for giving us the opportunity to work on **"E SHOP MANAGEMENT SYSTEM"** which helped us to learn so many new and useful things while doing research on this project. Last but not least I want to express my thanks to my family and friends for their motivation and help.

Thank You.

Introduction to The Project

The E Shop Management System called "Nothing" is use for effective management of a retail shop.

It has separate interfaces for administration and customer use. The administrator interface can handle listing / unlisting of products and modification of existing product details. Meanwhile customers can use the customer interface to browse the products, add the different products to cart, remove items from the cart and finally export the details of the products in cart as a CSV file.

The interface is made with ease of use in mind and can be understood at a glance. The administrator and customer interfaces are separate which ensures security of the system.

The details of the products stored in the system is persistent due to the use of MySQL as the database. The front end of the system is made using Python and the backend as mentioned before in MySQL.

Objective

The objective of the E Shop Management System is to streamline the retail process. This system makes the management of products simpler for the seller while at the same time making the customer experience smooth and appealing. Having a proper product management system can greatly increase efficiency, simplify management and also reduce cost and waste of time.

Source Code

Prerequisites

The program is divided into smaller modules for better readability and easier management. Here is the file structure:

```
admin.py
boxprint.py
cart.py
customer.py
error_corrector.py
main.py
sql_handler.py
db_create.py
```

`admin.py` and `customer.py` are the different interfaces. `cart.py` is a user defined module used by customer.py for certain functions. `error_corrector.py` is used to make sure all the required modules are installed and that they are working properly before executing the program. `boxprint.py` is a user defined module which is used to display the output neatly in boxes. `sql_handler.py` is used by the python program to communicate with the MySQL database.

`main.py` is where the program starts, it the file that should be executed to start the program.

`db_create.py` is a script to create a sample database for testing

Code

main.py

```
from boxprint import box
import error_corrector as ec

# Print the logo
print("""
888b    888          888    888      d8b
8888b   888          888    888      Y8P
88888b  888          888    888
888Y88b 888  .d88b.  888888 88888b.  888 88888b.   .d88b.
888 Y88b888 d88""88b 888    888 "88b 888 888 "88b d88P"88b
888  Y88888 888  888 888    888  888 888 888  888 888  888
888   Y8888 Y88..88P Y88b.  888  888 888 888  888 Y88b 888
888    Y888  "Y88P"   "Y888 888  888 888 888  888  "Y88888
                                                      888
We sell everything...                            Y8b d88P
                                                  "Y88P" """)

print("Starting...")
# Run some checks to make sure that all the prerequisites are present
ec.run_checks()

# Log in a as either and admin or customer
box(["Please Log In"], width=20)
user = input("Select User [Customer/Admin]: ")

if user.lower() == "customer" or user.lower() == "c":
    import customer
elif user.lower() == "admin" or user.lower() == "a":
    import admin
else:
    print("User not defined")
```

admin.py

```
# The Admin interface

# This module is used to list / unlist products, modify product details
# This interface is supposed to be used by the Administrator of the shop
```

```python
from boxprint import box
import sql_handler as sqh

box(["Welcome Admin"], width=20)

# Use a while loop to accept and process input
while True:
    print("[ADMIN] q:QUIT l:LIST-ITEM u:UNLIST-ITEM m:MODIFY-ITEM s:SHOW-SHOP
    ch = input(": ")

    if ch=="l":
        # Get product details
        name = input("Enter product name: ")
        price = float(input("Enter product price: "))
        stock = int(input("Enter product stock: "))

        # Add the product to database using sql_handler module
        product = sqh.listProduct(name, price, stock)
        if product:
            box([f"{product[1]} added to shop"], width=5)

    if ch=="u":
        pid = int(input("Enter product id: "))

        # Remove product from database using sql_handler module
        response = sqh.unlistProduct(pid)

        if response:
            box([f"Product {pid} was removed from shop"], width=5)

    if ch=="m":
        pid = int(input("Enter product id: "))

        # Get new product details
        print("Enter the new details")
        name = input("  Name :")
        price = float(input("  Price:"))
        stock = int(input("  Stock:"))

        product = sqh.modifyProduct(pid, name, price, stock)

        if product:
            box([f"{product[1]} was modified"], width=5)

    if ch=="s":
        # Show all items in the database
        sqh.showShop()

    if ch=="q":
        print("[ Exiting ]")
        exit()
```

customer.py

```
# The Customer interface

# This interface is used to add products to cart and also to start the cart m
# to manage the cart (i.e add / remove items, view cart and export items in c
# This interface is supposed to be used by the customer

from boxprint import box
import sql_handler as sqh
import cart

box(["Welcome Customer"], width=20)

box(["Shop"], width=10)

# Use a while loop to accept and process input
while True:
    print("[SHOP] q:QUIT a:ADD c:GO-TO-CART s:SHOW-SHOP")
    ch = input(": ")

    if ch=="a":
        # Add a product to the Cart
        cart.addProduct()

    if ch=="c":
        # Start the cart loop
        cart.cart_init()

    if ch=="s":
        sqh.showShop()

    if ch=="q":
        print("[ Exiting ]")
        exit()
```

cart.py

```
# Cart

# The cart module used by the customer interface.

from boxprint import box, pad
import sql_handler as sqh
import csv

# The cart is temporary and is cleared on exit, unlike the database
```

```
user_cart = [] # Product ids are stored in the cart

# A function to start a loop
def cart_init():

    # Use a while loop to accept and process input
    while True:

        box(["Cart"], width=10)

        showCart()

        print("[CART] q:QUIT r:REMOVE x:EXPORT")
        ch = input(": ")

        if ch=="r":
            removeProduct()

        elif ch=="x":
            exportAsCSV()

        elif ch=="q":
            print("[ Returning to SHOP ]")
            break # Break from this loop to enter previous loop in customer.p

        else:
            print("[ INVALID INPUT ]")


# Some functions are defined here

# Add a product to the cart
def addProduct():
    pid = int(input("Enter the product id: "))
    pids = sqh.getPIDs()

    if pid in user_cart:
        box(["Product already in cart"], width=5)

    elif pid in pids:
        user_cart.append(pid)
        product = sqh.getProduct(pid)
        box([f"{product[1]} added to cart"], width=5)

    else:
        print("[ INVALID PRODUCT ID ]")

# Remove a product from the cart
def removeProduct():
    pid = int(input("Enter the product id: "))

    if pid in user_cart:
```

```
        user_cart.remove(pid)
        product = sqh.getProduct(pid)
        box([f"{product[1]} removed from cart"], width=5)

    else:
        print("[ PRODUCT NOT IN CART ]")

# Show all items in the cart
def showCart():
    if user_cart == []:
        print("[ CART IS EMPTY ]")

    else:

        lines = [] # All the lines to be printed

        # Add the header
        lines.append("ID" + "  "+ "Name" + " "*28 + "Price"+ " "*5)
        lines.append("-"*46)

        price_total = 0

        # Add the products
        for pid in user_cart:
            product = sqh.getProduct(pid)
            lines.append(f"{pad(product[0], 2)}  {pad(product[1], 30)}  {pad(
            price_total+=product[2]

        box(lines)
        box([f"Total: {price_total}"], width=28) # Print the total price

# Export all items in the cart as a CSV file
def exportAsCSV():
    name = input("Enter name of file (without .csv): ")

    with open(name+".csv", "w", newline="") as f:

        wr = csv.writer(f)
        wr.writerow(["Product ID", "Name", "Price"])

        price_total = 0 # Price of each product is added to the total
        for pid in user_cart:
            product = sqh.getProduct(pid)
            wr.writerow([product[0], product[1], product[2]])
            price_total += product[2] # Keep tally of the prices

        # Write the total price
        wr.writerow(["Total", "", price_total])

    box([f"Cart was exported to {name}.csv"], width=5)
```

boxprint.py

```
# Box Print

# A module to print output in a neat little box.

charset1 = {"tr":"┐", "tl":"┌", "br":"┘", "bl":"└", "vr":"│", "hr":"─"}
charset2 = {"tr":"╗", "tl":"╔", "br":"╝", "bl":"╚", "vr":"║", "hr":"="}
charset3 = {"tr":"ヽ", "tl":"ㄏ", "br":"ノ", "bl":"L", "vr":"|", "hr":"−"}

chars = charset3

# Neatly print the output in a box
# Can be used to replace the print() function
def box(lines, width=40):
    """Print the output neatly in a box

    Accepts list of strings as argument and prints
    each string as a line in a box"""

    # if line is longer that width, then set it as width
    for line in lines:
        if len(line)>width:
            width = len(line)

    # pad the right of all lines with spaces
    newlines = []
    for line in lines:
        newlines.append(line+ " "*(width-len(line)))

    # print lines in a box
    print(chars["tl"] + chars["hr"]*(width+2) +chars["tr"]) # print top of bo
    for line in newlines:
        print(chars["vr"]+" " + line + " "+chars["vr"])
    print(chars["bl"] + chars["hr"]*(width+2) +chars["br"]) # print bottom of

# A function to add spaces to the end of a string to make it a given length
def pad(string, length):
    """Pad right side of a string with spaces"""

    string = str(string)
    if len(string) > length:
        return string
    else:
        new_string = string + (length-len(string))*" " # Adds spaces to end o
        return new_string
```

error_corrector.py

```
# Error Corrector

# A simple script to check whether all the required modules are installed
# and that they are working properly.

def run_checks():
    """Check whether MySQL is installed, it is accessible
    and whether it has the proper databases and tables"""

    # Check for MySQL-Python connector
    try:
        import mysql.connector
        print("[ ok ] mysql.connector working")
    except:
        print("[Error] Unable to import mysql.connector")
        exit()

    # Check for MySQL installation
    try:
        conn = mysql.connector.connect(host="localhost", user="root", passwd=
        print("[ ok ] MySQL found")
    except:
        print("[Error] Unable to connect to MySQL")
        exit()

    # Check for database
    try:
        cursor = conn.cursor()
        cursor.execute("use nothing_shop")
        print("[ ok ] Database found")
    except:
        print("[Error] Unable to access database")
        exit()

    # Check for tables
    try:
        cursor.execute("select * from products")
        print("[ ok ] Tables found")
    except:
        print("[Error] Unable to access table")
        exit()

    # Check for CSV module
    try:
        import csv
        print("[ ok ] CSV module found")
    except:
        print("[Error] Unable to import csv")
        exit()
```

sql_handler.py

```
# SQL Handler

# This module is used to communicate with the MySQL Database Server.
# This is the module that accesses data from the database and parses it for
# easier use throughout the program.

import mysql.connector
from boxprint import box

conn = mysql.connector.connect(host="localhost",user="root",passwd="password"
cur = conn.cursor()

# Get a list of all the product IDs
def getPIDs():
    cur.execute("select pid from products")
    data = cur.fetchall()
    pids = []

    for item in data:
        pids.append(item[0])

    return pids

# Get all the products in the shop
def getShop():
    cur.execute("select * from products")
    products = cur.fetchall()
    return products

# Print all the products in the shop
def showShop():
    products = getShop()

    if products == []:
        box(["Shop is Empty"], width=5)

    else:
        for item in products:
            box([
            f"ID: {item[0]}",
            f"Name  :  {item[1]}",
            f"Price :  {item[2]}",
            f"Stock :  {item[3]}",
            ], width=30)

# Get the details of a product using its ID
def getProduct(pid):
    cur.execute(f"select * from products where pid={pid}")
```

```
    product = cur.fetchall()

    if product:
        return product[0]
    else:
        return False

# List a product in the shop
def listProduct(name, price, stock):
    pids = getPIDs()

    for n in range(1, len(pids)+2):
        if n not in pids:
            newpid = n
            break

    try:
        cur.execute(f"insert into products values({newpid}, '{name}', {price}
        conn.commit()

    except:
        print("[ INVALID PARAMETERS ]")
        return False

    return getProduct(newpid)

# Unlist a product from the shop
def unlistProduct(pid):
    pids = getPIDs()

    if pid in pids:
        cur.execute(f"delete from products where pid={pid}")
        conn.commit()
        return True

    else:
        print("[ INVALID PRODUCT ID ]")
        return False

# Modify the details of a product in the shop
def modifyProduct(pid, name, price, stock):

    if pid in getPIDs():
        cur.execute(f"update products set name='{name}', price={price}, stock
        conn.commit()

        return getProduct(pid)

    else:
        print("[ INVALID PRODUCT ID ]")
        return False
```

db_create.py

```
# DB_Create

# A simple script to create a sample shop's database.
# Note that this script overwrites the currently present shop's database.

import mysql.connector

conn = mysql.connector.connect(host="localhost",user="root",passwd="password"
cur = conn.cursor()

cur.execute("drop database if exists nothing_shop")
cur.execute("create database nothing_shop")

conn.close()

conn = mysql.connector.connect(host="localhost",user="root",passwd="password"
cur = conn.cursor()

products = [
    ["Item1", "10.00", "12343"],
    ["Item2", "432.00", "1233"],
    ["Item3", "3420.00", "134323"],
    ["Item4", "4310.00", "123"],
    ["Item5", "1340.00", "1323"],
    ["Item6", "140.00", "1243"],
    ["Item7", "130.00", "12323"],
    ["Item8", "12340.00", "1423"],
    ["Item9", "140.00", "1243"],
    ["Item10", "1230.00", "1243"]
]

cur.execute("create table products(pid int primary key, name varchar(30), pri

i = 1
for item in products:
    cur.execute(f"insert into products values({i}, '{item[0]}', {item[1]}, {i
    i+=1

conn.commit()
```