COMPUTER SCIENCE

INVESTIGATORY PROJECT

ON

Super Market Management System

Submitted By :

Bhavesh L

Class XII –A

KENDRIYA VIDYALAYA CISF RTC(A), THAKKOLAM

2021-22

CERTIFICATE

This is to certify that Bhavesh L of class XII – A has successfully completed the Investigatory Project on the topic **"Supermarket Management System"** under the guidance of Mr. Vibin V, PGT(Computer Science) during the academic year 2021-22.

Signature (Subject Teacher) Signature (Student)

Signature (Principal)

Signature (Examiner)

ACKNOWLEDGEMENT

I wish to express my deep gratitude and sincere thanks to my Computer Science teacher Mr. Vibin V and C A Malarvizhi, principal Kendriya Vidyalaya CISF RTC(A), Thakkolam, who gave me the golden opportunity to do this wonderful project on the topic “Supermarket Management System”.

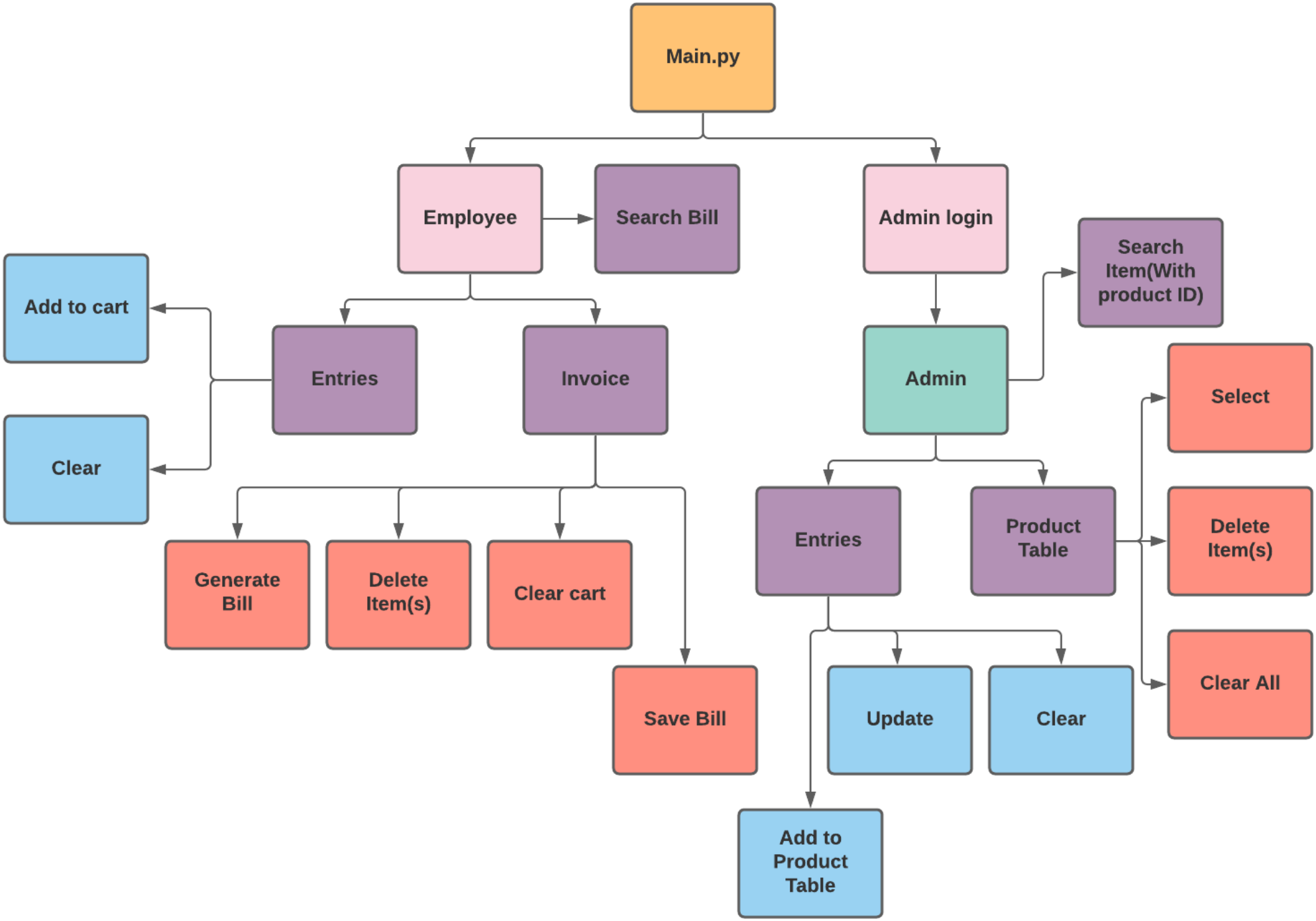
I take this opportunity to express my gratitude to my Computer Teacher for his invaluable guidance, constant motivation and encouragement. I am really thankful to you.

INDEX

INTRODUCTION

This is supermarket management system developed on the basis to manage items, invoice etc. The aim of this project is to develop a Supermarket Management system that is available for the grocery shops. In India, there are many Supermarkets which are accessible to this system. They just need a person to operate this system. This systems reduces work of human by managing all the datas by itself with less number of errors, whereas a human could confuse and could not handle all the datas. Our aim is to offer user-friendly and easy to use system.

SYSTEM DESIGN


Main.py
Employee
Search Bill
Admin login
Add to cart
Clear
Entries
Invoice
Search Item(With product ID)
Admin
Select
Generate Bill
Delete Item(s)
Clear cart
Save Bill
Entries
Product Table
Delete Item(s)
Update
Clear
Clear All
Add to Product Table

SYSTEM REQUIREMENTS

HARDWARE

- PROCESSOR: Intel Core Duo 2.4 Ghz or above
- RAM: 2 GB or higher

SOFTWARE:

- OPERATING SYSTEM: Windows 7 or above.
- INTERPRETER: Python 3.6 (or later version)
- DBMS: Mysql 5.5

BUILT-IN MODULES:

- **OS**

  This module is used to perform os operations like creating and deleting files.

- **MYSQLITE3**

  This module is extensively used to provide mysql-python connectivity and to perform insertion and deletion in database.

- **DATETIME**

  This module is used to get date and time when the respected functions are called.

PROJECT MODULES:

- **TKINTER**

  This module is used to give GUI to our program.

- **TKINTER.TTK**

  This module is used to create table called treeview table inside the GUI. Even though this a part of Tkinter module, it should be imported separately.

- **TKINTER.MESSAGEBOX**

  This module is used to create pop up messages, errors etc. Even though this a part of Tkinter module, it should be imported separately.

- **TKINTER.FONT**

  This module is used to get and use various fonts inside the GUI. Even though this is a part of Tkinter module, it should be imported separately.

➢ Main.py

This is the main module which begins execution of the project.
This module consists of two buttons with commands given in different modules that are explained below.

1. Employee.py
2. Admin_login.py

➢ Employee.py

This module is used to do activities related to invoice. It is subdivided into two parts those are Entries and Invoice. As the name suggests the entries are the normal user entries that are used to do some activities like adding them to cart. Invoice is a table to which the items are being added through entries. The operations which can be done on invoice are listed below.

1. Generate bill – It generates the bill with total, customer name and customer contact which are taken through another set of entries.
2. Delete Item(s) – It is used to delete the selected item(s).
3. Clear cart – It is used to clear the cart.
4. Save Bill – It saves the bill to a text file which can be accessed anytime only with customer number. The concept of file handling is used here.

➢ Admin_login.py

This module is a simple login page which are displayed to the user when clicked on Admin button in Main.py. This page takes two entries these includes Username and Password. If the user enters correct username and password then Admin page will be displayed and when user enters incorrect username and password, appropriate message will be shown to the user.

➢ Admin.py

This module consists an Admin page which is also subdivided into two parts. Those are, Entries and Product table. These entries also have same definition of those entries in Employee.py, the extra here is that we can also update the items in the product table. Product table is a table consist of all the products available in the market with category and rates corresponding to them. The operations that can be performed on product table are listed below.

1. Select – This grabs the data of the selected item in the table and auto enters it in the entries so as to update the item.
2. Delete Item(s) - It is used to delete the selected item(s).
3. Clear All – It clears all the items available in the market. Think before using it.

DATABASE TABLES:

Database: RSgroceries

Tables:

1. category : This consist of only one column that is category which as all the categories available in the market with category column itself as a primary key.

2. products : This consist of 4 columns those are product_id, product_name, category and product_rate. Product_id is the primary key and category is the foreign key of this table

SOURCE CODE

(Scan the above QR code to access the code)

Main.py

```
# Supermarket management system
# Author - BHAVESH L and JOHNEY B

from tkinter import *
from tkinter.font import Font
import os
from tkinter import messagebox
Main_Interface = Tk()
import sqlite3


# Creating Mysql connection
dbconn = sqlite3.connect("./Database/RSgroceries.db")

# Create a cursor to give commands
cursor = dbconn.cursor()

# Create Tables
# category Table
cursor.execute("""CREATE TABLE if not exists category(
category varchar(100) NOT NULL primary key
    )
    """)
dbconn.commit()
cursor.execute("""CREATE TABLE if not exists products(
    product_id int  not null primary key,
    product_name varchar(100) not null,
    product_rate int not null,
    category varchar(100) not null references category(category)
    )
    """)
dbconn.commit()

products = [
    ['101', 'Maaza 1 litre', '65', 'Beverages'],
    ['102', 'Coco Cola 1 litre', '70', 'Beverages'],
    ['103', 'Fanta 1 litre', '66', 'Beverages'],
    ['104', 'Miranda 1 litre', '72', 'Beverages'],
    ['105', '7 UP 1 litre', '60', 'Beverages'],
    ['106', 'Bovanto 1/2 litre', '35', 'Beverages'],
    ['107', 'Frooti 1/2 litre', '40', 'Beverages'],
    ['108', 'Pepsi 1/2 litre', '30', 'Beverages'],
    ['109', 'Apple Juice 1/2 litre', '25', 'Beverages'],
```

['110', 'Sprite 1/2 litre', '35', 'Beverages'],
['111', 'Aavin Milk 1 litre', '50', 'Dairy'],
['112', 'Aavin Milk 1/2 litre', '26', 'Dairy'],
['113', 'Aavin Milk 250 ml', '12', 'Dairy'],
['114', 'Amul Butter 100 g', '46', 'Dairy'],
['115', 'Arokya Curd 1 litre', '55', 'Dairy'],
['116', 'Aavin Curd 1 litre', '54', 'Dairy'],
['117', 'Amul Ghee 500 g', '245', 'Dairy'],
['118', 'MM Paneer', '230', 'Dairy'],
['119', 'Bhav Cheese 500g', '75', 'Dairy'],
['120', 'Cond. Milk 250ml', '90', 'Dairy'],
['121', 'Chilli Sauce 500g', '118', 'Sauce'],
['122', 'Sweent&Chilli Sauce 500g', '108', 'Sauce'],
['123', 'Tomato Sauce 500g', '100', 'Sauce'],
['124', 'Soya Sauce 500g', '110', 'Sauce'],
['125', 'Hot Tomato Sauce 500g', '115', 'Sauce'],
['126', 'Salt Bread', '21', 'Bread'],
['127', 'Milk Bread', '22', 'Bread'],
['128', 'Wheat Bread', '20', 'Bread'],
['129', 'Chicken Wings 400g', '270', 'Meat'],
['130', 'Chicken Breast 250g', '240', 'Meat'],
['131', 'Pork 500g', '200', 'Meat'],
['132', 'Beaf 1Kg', '290', 'Meat'],
['133', 'Chicken Boneless 500g', '250', 'Meat'],
['134', 'Chicken Leg Pie 1Kg', '190', 'Meat'],
['135', 'Full Chicken ', '470', 'Meat'],
['136', '1Kg Basmati Rice', '200', 'Rice'],
['137', '1Kg Idli Rice ', '275', 'Rice'],
['138', '1Kg Tiffin Rice', '230', 'Rice'],
['139', '1Kg Basmati Rice', '200', 'Rice'],
['140', 'Ashir Atta 1Kg ', '45', 'Cereals'],
['141', 'RS Oats 500g ', '30', 'Cereals'],
['142', 'RS Frosted Flakes 500g ', '50', 'Cereals'],
['143', 'RS Oats 500g ', '30', 'Cereals'],
['144', 'RS Flakes 200g ', '17', 'Cereals'],
['145', 'RS Oats 500g ', '30', 'Cereals'],
['146', 'RS Baking Soda 550g ', '235', 'Bakery'],
['147', 'RS Baking Powder 1Kg ', '60', 'Bakery'],
['148', 'Cake 1Kg', '50', 'Bakery'],
['149', 'Choclate Cake 1piece', '15', 'Bakery'],
['150', 'Strawberry Pastries 1pie', '15', 'Bakery'],
['151', 'Cream Bun', '10', 'Bakery'],
['152', 'Butter Biscuits', '12', 'Bakery'],
['153', 'Natraj 10 Pencils ', '50', 'Stationary'],
['154', 'Natraj Ge. Box', '60', 'Stationary'],

['155', 'Natraj LS Scale', '10', 'Stationary'],
['156', 'Natraj SS Scalw', '5', 'Stationary'],
['157', 'DOMS ColourPencils 10', '20', 'Stationary'],
['158', 'DOMS Oil Pastels', '30', 'Stationary'],
['159', 'Natraj Sharpner', '3', 'Stationary'],
['160', 'FaberCastle M.pencil 0.7', '15', 'Stationary'],
['161', 'Apsara 0.7 led box ', '10', 'Stationary'],
['162', 'Lizol 500ml', '65', 'Hygiene'],
['163', 'Lizol l Litre', '120', 'Hygiene'],
['164', 'Harpic 500ml', '70', 'Hygiene'],
['165', 'Colgate Toothpaste BS', '25', 'Hygiene'],
['166', 'Pantanjli Toothpaste', '30', 'Hygiene'],
['167', 'Oral B Toothbrush', '15', 'Hygiene'],
['168', 'Close Up Toothpaste S', '20', 'Hygiene'],
['169', 'Colgate Toothbrush', '17', 'Hygiene'],
['170', 'MouthWasher 500ml', '50', 'Hygiene'],
['171', 'Sanitiser 500ml', '60', 'Hygiene'],
['172', 'Horlicks 350g', '49', 'Health'],
['173', 'Boost 500g', '100', 'Health'],
['174', 'Complan 500g', '45', 'Health'],
['175', 'Lays Blue S', '5', 'Snacks'],
['176', 'Lays Red S', '5', 'Snacks'],
['177', 'Lays Yellow S', '5', 'Snacks'],
['178', 'Lays Green S', '5', 'Snacks'],
['179', 'Lays Orange S', '5', 'Snacks'],
['180', 'Bingo Mad Angles S', '5', 'Snacks'],
['181', 'Bingo Mad Angles B', '10', 'Snacks'],
['182', 'Taka Tak B', '10', 'Snacks'],
['183', 'Lays Blue B', '10', 'Snacks'],
['184', 'Lays Blue B', '10', 'Snacks'],
['185', 'Lays Green B', '10', 'Snacks'],
['186', 'Lays Yellow B', '10', 'Snacks'],
['187', 'Lays Red B', '10', 'Snacks'],
['188', 'Lays Orange B', '10', 'Snacks'],
['189', 'Jim Jam S', '5', 'Snacks'],
['190', 'Jim Jam B', '10', 'Snacks'],
['191', 'Bourbon Bis ', '10', 'Snacks'],
['192', 'Cinnamon 50g ', '10', 'Seasonings'],
['193', 'Pepper 50g ', '10', 'Seasonings'],
['194', 'Fennugreek 50g ', '5', 'Seasonings'],
['195', 'Chinese Sea. 50g ', '10', 'Seasonings'],
['196', 'FCB Chicken M.', '10', 'Masalas'],
['197', 'FCB Fish F M.', '10', 'Masalas'],
['198', 'FCB Mutton M.', '10', 'Masalas'],
['199', 'FCB Sambar M.', '10', 'Masalas'],

```
    ['200', 'Arun cupI.', '15', 'IceCreams'],
    ['201', 'Arun ConeI. S', '17', 'IceCreams'],
    ['202', 'Arun ConeI. M', '25', 'IceCreams'],
    ['203', 'Arun ConeI. B', '35', 'IceCreams'],
    ['204', 'Jamai Kulfi.', '10', 'IceCreams'],
    ['205', 'Aman Family Pack I.', '80', 'IceCreams']]


# Add datas
for data in products:
    try:
        cursor.execute("INSERT INTO products VALUES(:product_id, :product_name,
:product_rate, :category)",
                {
                    "product_id": data[0],
                    "product_name": data[1],
                    "product_rate": data[2],
                    "category": data[3]
                }
                )
        dbconn.commit()
    except sqlite3.IntegrityError:
        pass

# Category
category_values = [
    ['Bakery'],
    ['Beverages'],
    ['Bread'],
    ['Cereals'],
    ['Dairy'],
    ['Hygiene'],
    ['IceCreams'],
    ['Masalas'],
    ['Meat'],
    ['Rice'],
    ['Sauce'],
    ['Seasonings'],
    ['Snacks'],
    ['Stationary']]

for data_1 in category_values:
    try:
        cursor.execute("INSERT INTO category VALUES(:category)",
                {"category": data_1[0]}
                )
```

```
            dbconn.commit()
        except sqlite3.IntegrityError:
            pass

def Exit():
    sure = messagebox.askyesno("Exit","Are you sure you want to exit?",
parent=Main_Interface)
    if sure == True:
        Main_Interface.destroy()

Main_Interface.protocol("WM_DELETE_WINDOW", Exit)


def admpg():
    Main_Interface.withdraw()
    os.system("python Admin_login.py")
    Main_Interface.deiconify()

def emp():
    Main_Interface.withdraw()
    os.system("python Employee.py")
    Main_Interface.deiconify()


# Fixing GUI Dimensions
Main_Interface.geometry("1150x650")
Main_Interface.resizable(0, 0)

# Fixing Title
Main_Interface.title("RS Groceries")

# Fixing GUI Background
Background = PhotoImage(file="./images/Bg_main.png")
Bg_label = Label(Main_Interface, image=Background)
Bg_label.place(x=0, y=0, relwidth=1, relheight=1)


#Fixing GUI Icon
Main_Interface.iconbitmap("./images/Logo.ico")


# Creating Button
font_1 = Font(family="Franklin Gothic Medium",size=15,weight="bold")
```

```
# Button 1
button1 =
Button(Main_Interface,text="EMPLOYEE",bg="#38b7fe",fg="black",padx=30,pady=1
0,width=20,font=font_1,activebackground="#38b7fe",activeforeground="black",comm
and=emp)
button1.configure(relief="flat")
button1.configure(overrelief="flat")
button1.configure(borderwidth="0")
button1.place(relx=0.32, rely=0.42, width=180, height=90,anchor=E)

# Button 2
button2 = Button(Main_Interface, text="ADMIN",bg="#38b7fe",
fg="black",padx=30,pady=10,width=20,font=font_1,activebackground="#38b7fe",acti
veforeground="black",command=admpg)
button2.configure(relief="flat")
button2.configure(overrelief="flat")
button2.configure(borderwidth="0")
button2.place(relx=0.70, rely=0.42, width=240, height=90, anchor=W)

dbconn.close()
Main_Interface.mainloop()
```

Employee.py

```
from tkinter import *
from tkinter import messagebox
from tkinter.font import Font
from tkinter import ttk
import datetime
# import mysql.connector as mysql
import sqlite3

# Creating Mysql connection
dbconn = sqlite3.connect("./Database/RSgroceries.db")


# Create a cursor to give commands
cursor = dbconn.cursor()

# Create Tables
# category Table
cursor.execute("""CREATE TABLE if not exists category(
category varchar(100) NOT NULL primary key
    )
    """)
dbconn.commit()
cursor.execute("""CREATE TABLE if not exists products(
    product_id int  not null primary key,
    product_name varchar(100) not null,
    product_rate int not null,
    category varchar(100) not null references category(category)
    )
    """)
dbconn.commit()

cursor.execute("""
Select * From category
""")
Category_1 = cursor.fetchall()

# Creating TKinter Window
billing = Tk()
billing.geometry("1330x750")
billing.resizable(0, 0)
billing.iconbitmap("./images/Logo.ico")
billing.title("Employee")
font_1 = Font(family="Calibri",size=15,weight="bold")
```

```
# Fixing GUI Background
Background = PhotoImage(file="./images/Employee_bg.png")
Bg_label = Label(billing, image=Background)
Bg_label.place(x=0, y=0, relwidth=1, relheight=1)

# Logout command
def Exit():
    sure = messagebox.askyesno("Exit","Are you sure you want to exit?", parent=billing)
    if sure == True:
        billing.destroy()

# Creating logout button
logout_img = PhotoImage(file="./images/logout.png")
logout_button = Button(billing, image=logout_img,
borderwidth=0,relief="flat",overrelief="flat", command=Exit)
logout_button.place(relx=0.0155, rely=0.038,width=39,height=31)


# Creating invoice
invoice = ttk.Treeview(billing)
invoice["columns"] = ("Product Name", "Qty", "Rate", "Cost")

invoice.column("#0", width=0,stretch=NO)
invoice.column("#1", width=301,anchor="center")
invoice.column("#2", width=80,anchor="center")
invoice.column("#3",width=120,anchor="center")
invoice.column("#4",width=120,anchor="center")

invoice.heading("#0",text="")
invoice.heading("#1",text="Product Name")
invoice.heading("#2",text="Qty")
invoice.heading("#3",text="Rate")
invoice.heading("#4",text="Cost")
invoice.place(relx=0.5032,rely=0.4517,height=245)

Scroll_invoice = Scrollbar(orient="vertical",command=invoice.yview)
invoice.configure(yscroll=Scroll_invoice.set)
Scroll_invoice.place(relx=0.9593,rely=0.4537, height=275)


# Creating all the entry fields
# Creating Entry for name and contact
# Name
Name_entry = Entry(billing,font=font_1,relief="flat",bg="#0089fe")
Name_entry.bind("")
```

```
Name_entry.place(relx=0.619,rely=0.124,width=140,height=30)
# Contact
contact_entry = Entry(billing,font=font_1,relief="flat",bg="#0089fe")
contact_entry.place(relx=0.869,rely=0.124,width=140,height=30)

# Creating entry for product and quantity
# List of categories
category = ["Choose the Category"]
for cat_n in Category_1:
    category.append(cat_n[0])

# defining required functions
global Rate
global Final_prod
Rate = []
Final_prod = ["Choose product"]
def sel_cat(n):
    global Rate
    global Final_prod
    if Items.get() == "" or Items.get() == "Choose the Category":
        Items_1.configure(values=Final_prod)
        Items_1.current(0)

    cursor.execute("SELECT product_name, product_rate FROM products WHERE
category='{}'".format(Items.get()))
    prod_and_rate = cursor.fetchall()
    prods = ["Choose product"]
    rates = []
    for i in prod_and_rate:
        prods.append(i[0])
        rates.append(i[1])
    Final_prod=prods
    Rate=rates
    Items_1.configure(value=Final_prod)
    Items_1.current(0)

# Items category Drop Down
Items = ttk.Combobox(billing,values=category,font=font_1)
Items.current(0)
Items.place(relx=0.049,rely=0.355,width=428,height=53)

# Bind Items
Items.bind("<<ComboboxSelected>>", sel_cat)
```

```
# Product drop down
Items_1 = ttk.Combobox(billing, values=["Choose product"],font=font_1)
Items_1.current(0)
Items_1.place(relx=0.049,rely=0.536,width=430,height=53)

# Creating entry box for quantity
quantity_entry = Entry(billing,font=font_1,relief="flat")
quantity_entry.place(relx=0.050,rely=0.730,width=423,height=48)

# Defining Funtions
# Non billing commands
# Add to Cart
def add_to_cart():
    global Final_prod
    global Rate
    if (quantity_entry.get().isdigit()) or (quantity_entry.get()== ""):
        if (Items_1.get()!="" and quantity_entry.get()!="" and
Items.get().lower()!="choose the category" and Items_1.get()!="Choose product"):
            n = Final_prod.index(Items_1.get())
            rate_n = Rate[n - 1]
            if Items.get() in category:

invoice.insert("",index="end",values=(Items_1.get(),quantity_entry.get(),rate_n,int(qu
antity_entry.get())*rate_n))
                Items.current(0)
                quantity_entry.delete(0,END)
                Items_1.current(0)
                # Rate = []
                # Final_prod = ["Choose product"]
            else:
                messagebox.showerror("Error","Item not in the cart!")
        else:
            messagebox.showerror("Error", "Please fill the details")

    else:
        messagebox.showerror("Error", "Please Enter Correct Quantity!")

# Clear
def clear():
    Items.current(0)
    quantity_entry.delete(0, END)
    Items_1.current(0)

# Billing commands
font_3 = Font(family="Calibri",size=11,weight="bold")
```

```
global cust_name
global cust_contact
global date_time
global cust_no
global Total_n
global dummy
cust_name = ""
cust_contact = ""
date_time = ""
cust_no = ""
Total_n = ""
dummy = 0
def generate_bill():
    all_rec = invoice.get_children()
    Rows = []
    for rec in all_rec:
        values = invoice.item(rec).get("values")
        Rows.append(values)
    confirm_1 = messagebox.askyesno("Generate Bill", "Do you want to generate
bill?")
    if confirm_1 == 1:
        if Name_entry.get() != "" and contact_entry.get() != "":
            if Rows!=[]:
                costs_n = []
                if len(contact_entry.get()) == 10:
                    Delete_btn.configure(state="disabled")
                    global cust_name
                    global cust_contact
                    global date_time
                    global cust_no
                    global Total_n
                    global dummy
                    dummy = 1
                    all_rec = invoice.get_children()
                    for rec in all_rec:
                        values = invoice.item(rec).get("values")
                        costs_n.append(values[3])
                    Total_n = sum(costs_n)
                    # Customer number reading and writing from/to(respectively) a file
                    cust_no_read = open("Customer_number_counter.txt", "r")
                    count = cust_no_read.read()
                    cust_no_read.close()
                    cust_no = count
                    cust_no_write = open("Customer_number_counter.txt", "w")
                    count_inc = str(int(count) + 1)
                    cust_no_write.write(count_inc)
```

```
cust_no_write.close()

                # Other labels
                cust_name=Name_entry.get()
                cust_contact=contact_entry.get()
                date_time = datetime.datetime.now()
                # Adding customer name
                label_1 = Label(billing, text=cust_name, font=font_3, bg="#dae2f2",
anchor="w")
                label_1.place(relx=0.602, rely=0.368, width=250, height=40)

                # Adding customer number
                label_2 = Label(billing, text=cust_no, font=font_3, bg="#dae2f2",
anchor="w")
                label_2.place(relx=0.593, rely=0.423, width=70, height=15)

                # Adding customer contact
                label_3 = Label(billing, text=cust_contact, font=font_3,bg="#dae2f2",
anchor="w")
                label_3.place(relx=0.899,rely=0.368, width=80, height=40)

                # Adding date and time
                label_4 = Label(billing, text=date_time, font=font_3, bg="#dae2f2",
anchor="w")
                label_4.place(relx=0.886, rely=0.423, width=104, height=15)

                # Total
                font_4 = Font(family="Calibri", size=18, weight="bold")
                label_5 = Label(billing, text="Total = {}".format(Total_n), font=font_4,
bg="#ffffff", anchor="e")
                label_5.place(relx=0.800, rely=0.780, width=200, height=31)
                Name_entry.delete(0,END)
                contact_entry.delete(0,END)

            else:
                messagebox.showerror("Error", "Please enter correct contact number")
        else:
            messagebox.showerror("Error", "Cart is empty")
    else:
        messagebox.showerror("Error", "Fill the details of the customer")
else:
    pass

# Clear function definition
def clear_all():
    Delete_btn.configure(state="active")
```

```
for rec in all_rec:
        values = invoice.item(rec).get("values")
        Rows.append(values)
    if Rows == []:
        messagebox.showerror("Error","Cart is already empty")
    else:
        # Overwriting customer name
        label_1 = Label(billing, text="", font=font_3, bg="#dae2f2", anchor="w")
        label_1.place(relx=0.602, rely=0.368, width=250, height=40)

        # Overwriting customer number
        label_2 = Label(billing, text="", font=font_3, bg="#dae2f2", anchor="w")
        label_2.place(relx=0.593, rely=0.423, width=70, height=15)

        # Overwriting customer contact
        label_3 = Label(billing, text="", font=font_3, bg="#dae2f2", anchor="w")
        label_3.place(relx=0.899, rely=0.368, width=80, height=40)

        # Overwriting date and time
        label_4 = Label(billing, text="", font=font_3, bg="#dae2f2", anchor="w")
        label_4.place(relx=0.886, rely=0.423, width=104, height=15)

        # Overwriting
        font_4 = Font(family="Calibri", size=18, weight="bold")
        label_5 = Label(billing, text="", font=font_4, bg="#ffffff", anchor="e")
        label_5.place(relx=0.800, rely=0.780, width=200, height=31)
        for rows in invoice.get_children():
            invoice.delete(rows)
        Save_btn.configure(state="active")
        Generate_btn.configure(state="active")
        Delete_btn.configure(state="active")

def delete_many():
    items_n = invoice.selection()
    if items_n == ():
        messagebox.showerror("Error","No Item(s) selected")
    else:
        for rows_n in items_n:
            invoice.delete(rows_n)

def save_bill():
    global cust_name
    global cust_contact
    global date_time
    global cust_no
    global Total_n
```

```
    global dummy
    all_rec = invoice.get_children()
    if dummy == 0:
        messagebox.showerror("Error", "Please Generate the bill first")
    else:
        yes_no = messagebox.askyesno("Save Bill", "Are you sure you want to Save
Bill?")
        if yes_no == 1:
            Delete_btn.configure(state="active")
            bill_n = open("./All_bills/zBill_{}.txt".format(cust_no), "w")
            cust_det = [cust_name, cust_contact, cust_no, date_time,Total_n]
            for i in cust_det:
                bill_n.write(str(i) + "`")
            bill_n.write("\n")
            all_rec = invoice.get_children()
            for rec in all_rec:
                values = invoice.item(rec).get("values")
                for j in values:
                    bill_n.write(str(j) + "`")
                bill_n.write("\n")
            cust_name = ""
            cust_contact = ""
            date_time = ""
            cust_no = ""
            Total_n = ""
            clear_all()
            dummy = 0
        else:
            pass

# Creating main button widgets
# ***** Non billing widgets *****
# Add to invoice
Add_btn_1 = Button(billing,text="Add to
cart",bg="#ff1616",fg="black",font=font_1,command=add_to_cart)
Add_btn_1.configure(activebackground="#ff1616")
Add_btn_1.configure(activeforeground="black")
Add_btn_1.configure(relief="flat")
Add_btn_1.configure(borderwidth="0")
Add_btn_1.place(relx=0.064,rely=0.882,width=135,height=43)

# Clear
Clear_btn_1 =
Button(billing,text="Clear",bg="#ff1616",fg="black",font=font_1,command=clear)
Clear_btn_1.configure(activebackground="#ff1616")
Clear_btn_1.configure(activeforeground="black")
```

```
Clear_btn_1.configure(relief="flat")
Clear_btn_1.configure(borderwidth="0")
Clear_btn_1.place(relx=0.256,rely=0.882,width=135,height=43)



# ***** Billing widgets *****
font_2 = Font(family="Calibri",size=13,weight="bold")
# Save bill
Save_btn = Button(billing, text="Save Bill", bg="#ff1616",fg="black",font=font_2,
command=save_bill)
Save_btn.configure(activebackground="#ff1616")
Save_btn.configure(activeforeground="black")
Save_btn.configure(relief="flat")
Save_btn.configure(borderwidth="0")
Save_btn.place(relx=0.861,rely=0.887,width=135,height=43)

# Generate Bill
Generate_btn=Button(billing,text="Generate
Invoice",bg="#ff1616",fg="black",font=font_2,command=generate_bill)
Generate_btn.configure(activebackground="#ff1616")
Generate_btn.configure(activeforeground="black")
Generate_btn.configure(relief="flat")
Generate_btn.configure(borderwidth="0")
Generate_btn.place(relx=0.5165,rely=0.887,width=135,height=43)

# Delete Item
Delete_btn=Button(billing,text="Delete Item(s)",bg="#ff1616",fg="black",font=font_2,
command=delete_many)
Delete_btn.configure(activebackground="#ff1616")
Delete_btn.configure(activeforeground="black")
Delete_btn.configure(relief="flat")
Delete_btn.configure(borderwidth="0")
Delete_btn.place(relx=0.631,rely=0.887,width=135,height=43)

# Clear Items
Clear_btn_2=Button(billing,text="Clear",bg="#ff1616",fg="black",font=font_2,
command=clear_all)
Clear_btn_2.configure(activebackground="#ff1616")
Clear_btn_2.configure(activeforeground="black")
Clear_btn_2.configure(relief="flat")
Clear_btn_2.configure(borderwidth="0")
Clear_btn_2.place(relx=0.745,rely=0.887,width=135,height=43)
```

```
# Search Bills
# Defining funciton for searching bill
def search_bill():
    for rows in invoice.get_children():
        invoice.delete(rows)
    bill_no_2 = Cust_no_entry.get()
    try:
        # Getting and adding other details
        bill = open("./All_bills/zBill_{}.txt".format(bill_no_2),"r")
        other_details = bill.readline().split("`")
        customer_name = other_details[0]
        customer_contact = other_details[1]
        customer_id = bill_no_2
        date_time_n = other_details[3]
        Total_bill = other_details[4]
        # writing customer name
        label_1 = Label(billing, text=customer_name, font=font_3, bg="#dae2f2",
anchor="w")
        label_1.place(relx=0.602, rely=0.368, width=250, height=40)

        # writing customer number
        label_2 = Label(billing, text=customer_id, font=font_3, bg="#dae2f2",
anchor="w")
        label_2.place(relx=0.593, rely=0.423, width=70, height=15)

        # writing customer contact
        label_3 = Label(billing, text=customer_contact, font=font_3, bg="#dae2f2",
anchor="w")
        label_3.place(relx=0.899, rely=0.368, width=80, height=40)

        # writing date and time
        label_4 = Label(billing, text=date_time_n, font=font_3, bg="#dae2f2",
anchor="w")
        label_4.place(relx=0.886, rely=0.423, width=104, height=15)

        # writing Total
        font_4 = Font(family="Calibri", size=18, weight="bold")
        label_5 = Label(billing, text="Total = {}".format(Total_bill), font=font_4,
bg="#ffffff", anchor="e")
        label_5.place(relx=0.800, rely=0.780, width=200, height=31)

# Reading records
        records = bill.readlines()
        for i in records:
            splitted = i.split("`")
            invoice.insert("", index="end",
```

Admin_login.py

```
from tkinter import *
from tkinter import messagebox
import os
from tkinter.font import Font

adm = Tk()
adm.geometry("500x715")
adm.resizable(0, 0)
adm.iconbitmap("./images/Logo.ico")
adm.title("Login Page")

user = StringVar()
password = StringVar()

# Admin page
def admpage():
    adm.withdraw()
    os.system("python Admin.py")
    adm.deiconify()


# Fixing GUI Background
Background = PhotoImage(file="./images/Admin_login.png")
Bg_label = Label(adm, image=Background)
Bg_label.place(x=0, y=0, relwidth=1, relheight=1)


# Username Entry
font_1 = Font(family="Comic Sans MS",size=15,weight="bold")

entry1 = Entry(adm)
entry1.place(relx=0.225, rely=0.272, width=315, height=26)
entry1.configure(font=font_1)
entry1.configure(relief="flat")
entry1.configure(textvariable=user)


# Password Entry
entry2 = Entry(adm)
entry2.place(relx=0.225, rely=0.405, width=315, height=26)
entry2.configure(font=font_1)
entry2.configure(relief="flat")
entry2.configure(show="•")
entry2.configure(textvariable=password)
```

```
def admlog_op():
    Username = user.get()
    Password = password.get()
    if Username == "ADMIN":
        if Password == "1234":
            messagebox.showinfo("Login Page", "The login is successful.")
            entry1.delete(0, END)
            entry2.delete(0, END)
            adm.withdraw()
            admpage()
        else:
            messagebox.showerror("Oops!!", "You are not an admin.")
    else:
        messagebox.showerror("Error", "Incorrect username or password.")

# Confirm Button
font_2 = Font(family="Franklin Gothic Medium",size=15,weight="bold")


button1 = Button(adm)
button1.place(relx=0.230, rely=0.755, width=280, height=43)
button1.configure(relief="flat")
button1.configure(overrelief="flat")
button1.configure(activebackground="#D2463E")
button1.configure(foreground="#ffffff")
button1.configure(background="#D2463E")
button1.configure(font=font_2)
button1.configure(borderwidth="0")
button1.configure(text="""LOGIN""")
button1.configure(command=admlog_op)

# Exit
def Exit():
    adm.destroy()

adm.protocol("WM_DELETE_WINDOW", Exit)

adm.mainloop()
```

Admin.py

```
from tkinter import *
from tkinter import messagebox
from tkinter.font import Font
from tkinter import ttk
# import mysql.connector as mysql
import sqlite3

# Creating Tkinter Window
Admin = Tk()
Admin.geometry("1330x750")
Admin.resizable(0, 0)
Admin.iconbitmap("./images/Logo.ico")
Admin.title("Admin")

# Creating Mysql connection
dbconn = sqlite3.connect("./Database/RSgroceries.db")
# Create a cursor to give commands
cursor = dbconn.cursor()

# Create Tables
# category Table
cursor.execute("""CREATE TABLE if not exists category(
category varchar(100) NOT NULL primary key
    )
    """)
dbconn.commit()
cursor.execute("""CREATE TABLE if not exists products(
    product_id int  not null primary key,
    product_name varchar(100) not null,
    product_rate int not null,
    category varchar(100) not null references category(category)
    )
    """)
dbconn.commit()
cursor.execute("SELECT * FROM products")
prod_1 = cursor.fetchall()
# print(prod_1)
dbconn.commit()

# Fixing GUI Background
Background = PhotoImage(file="./images/Admin_bg.png")
Bg_label = Label(Admin, image=Background)
Bg_label.place(x=0, y=0, relwidth=1, relheight=1)
```

```
# Creating invoice
table = ttk.Treeview(Admin)
table["columns"] = ("ID","Product Name","Category", "Rate")

table.column("#0", width=0,stretch=NO)
table.column("#1", width=50,anchor="center")
table.column("#2", width=230,anchor="center")
table.column("#3",width=230,anchor="center")
table.column("#4",width=120,anchor="center")

table.heading("#0",text="")
table.heading("#1",text="ID")
table.heading("#2",text="Product Name")
table.heading("#3",text="Category")
table.heading("#4",text="Rate")
table.place(relx=0.50,rely=0.1139,height=528.8, width=630)

Scroll_invoice = Scrollbar(orient="vertical",command=table.yview)
table.configure(yscroll=Scroll_invoice.set)
Scroll_invoice.place(relx=0.961,rely=0.1140, height=527.3)

for row in prod_1:
    table.insert("",index="end",values=(row[0],row[1],row[3],row[2]))
# Defining Exit function
def Exit():
    sure = messagebox.askyesno("Exit","Are you sure you want to exit?",
parent=Admin)
    if sure == True:
        Admin.destroy()
        # adm.destroy()


# Creating logout button
logout_img = PhotoImage(file="./images/logout.png")
logout_button = Button(Admin, image=logout_img,
borderwidth=0,relief="flat",overrelief="flat", command=Exit)
logout_button.place(relx=0.0155, rely=0.038,width=39,height=31)


# Creating all the required widgets
# Creating text variables
cat = StringVar()
pro_name = StringVar()
pro_rate = StringVar()

font_1 = Font(family="Calibri",size=15,weight="bold")
```

```
# All Entry widgets
# Product Category Widget
Entry_1 = Entry(Admin,font=font_1,relief="flat",bg="#fefffe")
Entry_1.place(relx=0.043,rely=0.622,width=423,height=50)

# Product Rate Widget
Entry_2 = Entry(Admin, font=font_1,relief="flat",bg="#fefffe")
Entry_2.place(relx=0.043,rely=0.780,width=423,height=50)

# Product Name Widget
Entry_3 = Entry(Admin,font=font_1,relief="flat",bg="#fefffe")
Entry_3.place(relx=0.043,rely=0.463,width=423,height=50)

# Product Id Widget
Entry_4 = Entry(Admin,font=font_1,relief="flat",bg="#fefffe")
Entry_4.place(relx=0.043,rely=0.3205,width=423,height=50)

# Search code Entry Widget
Entry_5 = Entry(Admin, font=font_1,relief="flat",bg="#fefafa")
Entry_5.place(relx=0.161,rely=0.115,width=255,height=40)



# Defining all the required functions
# CREATING FUNCTION TO REMOVE UNWANTED CATEGORY
def unwanted_cat():
    category_delete_1 = table.get_children()
    categories_avail = []
    for rec in category_delete_1:
        values = table.item(rec).get("values")[2]
        categories_avail.append(values)
    cursor.execute("SELECT category FROM category")
    cat_t = cursor.fetchall()
    all_cat = []
    for i in cat_t:
        all_cat.append(i[0])
    available_category = []
    for fin in all_cat:
        if fin in categories_avail:
            available_category.append(fin)
        else:
            pass
    cursor.execute("DROP TABLE category")
    dbconn.commit()
    # Creating table product if not exist
    cursor.execute("""CREATE TABLE if not exists category(
      category varchar(100) NOT NULL primary key
```

```
)
        """)
    dbconn.commit()
    for last in available_category:
        try:
            cursor.execute("INSERT INTO category VALUES('{}')".format(last))
            dbconn.commit()
        except sqlite3.IntegrityError:
            pass

# Add to cart
def add_to_cart():
    # Creating table product if not exist
    cursor.execute("""CREATE TABLE if not exists category(
    category varchar(100) NOT NULL primary key
        )
        """)
    cursor.execute("""CREATE TABLE if not exists products(
        product_id int  not null primary key,
        product_name varchar(100) not null,
        product_rate int not null,
        category varchar(100) not null references category(category)
        )
        """)
    dbconn.commit()
    all_rec = table.get_children()
    ids = []
    for rec in all_rec:
        values = table.item(rec).get("values")[0]
        ids.append(values)
    if (Entry_2.get().isdigit() or Entry_2.get()==""):
        try:
            if Entry_1.get() != "" and Entry_2.get() != "" and Entry_3.get() != "" and
Entry_4.get() != "":
                n = messagebox.askyesno("Add to Market", "Are you sure you want to add
it to the Market?")
                if n == 1:
                    cursor.execute("SELECT product_id FROM products")
                    id_check = cursor.fetchall()
                    id_check_fin = []
                    dbconn.commit()
                    if (int(Entry_4.get()),) in id_check:
                        messagebox.showerror("Error", "Product id already in the market")
                    else:
                        table.insert("", index="end", values=(Entry_4.get(), Entry_3.get(),
Entry_1.get(), Entry_2.get()))
```

```
                cursor.execute("INSERT INTO products VALUES(:product_id,
:product_name,  product_rate, :category)",
                               {
                                   "product_id": Entry_4.get(),
                                   "product_name": Entry_3.get(),
                                   "product_rate": Entry_2.get(),
                                   "category": Entry_1.get()
                               }
                               )
                    cursor.execute("SELECT category FROM category")
                    categories_db = cursor.fetchall()
                    categories = []
                    for i in categories_db:
                        categories.append(i[0])
                    if Entry_1.get() not in categories:
                        cursor.execute("INSERT INTO category VALUES(:category)",
                                       {"category": Entry_1.get()})
                        dbconn.commit()
                    else:
                        pass
                    dbconn.commit()
                    Entry_1.delete(0, END)
                    Entry_2.delete(0, END)
                    Entry_3.delete(0, END)
                    Entry_4.delete(0, END)
                    unwanted_cat()

                else:
                    pass
            else:
                messagebox.showerror("Error", "Please fill the details")
        except ValueError:
            messagebox.showerror("Error", "Please enter correct product ID!")
    else:
        Entry_2.delete(0, END)
        messagebox.showerror("Error", "Please enter correct quantity!”)

# Update
def update():
    # Creating table product if not exist
    cursor.execute("""CREATE TABLE if not exists category(
        category varchar(100) NOT NULL primary key
            )
            """)
    cursor.execute("""CREATE TABLE if not exists products(
```

```
        product_id int  not null primary key,
        product_name varchar(100) not null,
        product_rate int not null,
        category varchar(100) not null references category(category)
        )
        """)
    dbconn.commit()

    Button_1.configure(state="active")
    if Entry_1.get() != "" and Entry_2.get() != "" and Entry_3.get() != "" and
Entry_4.get() != "":
        cursor.execute("SELECT product_id FROM products")
        id_check = cursor.fetchall()
        dbconn.commit()
        if (int(Entry_4.get()),) in id_check:
            all_rows = table.get_children()
            k = []
            for i in all_rows:
                if table.item(i).get("values")[0] == int(Entry_4.get()):
                    k.append(i)
                else:
                    pass
            table.item(k[0], text="", values=(int(Entry_4.get()) ,Entry_3.get(),
Entry_1.get(), Entry_2.get()))
            cursor.execute("""
            UPDATE products SET product_name = '{}', category = '{}', product_rate = {}
WHERE product_id = {}"""
                           .format(Entry_3.get(), Entry_1.get(), Entry_2.get(),
int(Entry_4.get())))
            dbconn.commit()
            cursor.execute("SELECT category FROM category")
            categories_db = cursor.fetchall()
            categories = []
            for i in categories_db:
                categories.append(i[0])
            if Entry_1.get() not in categories:
                cursor.execute("INSERT INTO category VALUES(:category)",
                               {"category": Entry_1.get()})
                dbconn.commit()
            Entry_1.delete(0, END)
            Entry_2.delete(0, END)
            Entry_3.delete(0, END)
            Entry_4.delete(0, END)
            unwanted_cat()

        else:
```

```
 else:
        messagebox.showerror("Error", "Fill all the details")

# Clear
def clear():
    # Creating table product if not exist
    cursor.execute("""CREATE TABLE if not exists category(
        category varchar(100) NOT NULL primary key
            )
            """)
    cursor.execute("""CREATE TABLE if not exists products(
        product_id int  not null primary key,
        product_name varchar(100) not null,
        product_rate int not null,
        category varchar(100) not null references category(category)
        )
        """)
    dbconn.commit()

    Entry_1.delete(0, END)
    Entry_2.delete(0, END)
    Entry_3.delete(0, END)
    Entry_4.delete(0, END)
    Button_1.configure(state="active")
    unwanted_cat()
# Select Item
def select_item():
    # Creating table product if not exist
    cursor.execute("""CREATE TABLE if not exists category(
        category varchar(100) NOT NULL primary key
            )
            """)
    cursor.execute("""CREATE TABLE if not exists products(
        product_id int  not null primary key,
        product_name varchar(100) not null,
        product_rate int not null,
        category varchar(100) not null references category(category)
        )
        """)
    dbconn.commit()

    items_n = table.selection()
    if len(items_n)>1:
        messagebox.showerror("Error", "Two or more items are selected")
    else:
```

```
 if items_n == ():
            messagebox.showerror("Error", "No Item(s) selected")
        else:
            sel_item = []
            for i in items_n:
                k = table.item(i, "values")
                for j in k:
                    sel_item.append(j)
            Entry_4.insert(0, sel_item[0])
            Entry_3.insert(0, sel_item[1])
            Entry_2.insert(0, sel_item[3])
            Entry_1.insert(0, sel_item[2])
            unwanted_cat()
            Button_1.configure(state="disabled")

# Delete item(s)
def delete_many():
    # Creating table product if not exist
    cursor.execute("""CREATE TABLE if not exists category(
        category varchar(100) NOT NULL primary key
            )
            """)
    cursor.execute("""CREATE TABLE if not exists products(
        product_id int  not null primary key,
        product_name varchar(100) not null,
        product_rate int not null,
        category varchar(100) not null references category(category)
        )
        """)
    dbconn.commit()

    items_n = table.selection()
    if items_n == ():
        messagebox.showerror("Error", "No Item(s) selected")

    else:
        n = messagebox.askyesno("Delete item(s)","Are you sure you want to delete
the selected item(s)?")
        if n == 1:
            pro_id = []
            for i in items_n:
                k = table.item(i, "values")
                pro_id.append(k[0])
            for rows_n in items_n:
                table.delete(rows_n)
```

```
 for row in pro_id:
                cursor.execute("DELETE FROM products WHERE
product_id={}".format(row))
                dbconn.commit()
            unwanted_cat()
        else:
            pass

# Clear All
def clear_all():
    # Creating table product if not exist
    cursor.execute("""CREATE TABLE if not exists category(
       category varchar(100) NOT NULL primary key
          )
          """)
    cursor.execute("""CREATE TABLE if not exists products(
        product_id int  not null primary key,
        product_name varchar(100) not null,
        product_rate int not null,
        category varchar(100) not null references category(category)
        )
        """)
    dbconn.commit()

    if table.get_children() == ():
        messagebox.showerror("Error","No Items in the Market")
    else:
        n = messagebox.askyesno("Clear All", "Are you sure you want to clear all the
items?")
        if n == 1:
            for rows in table.get_children():
                table.delete(rows)
            cursor.execute("DROP TABLE products")
            dbconn.commit()
            unwanted_cat()
        else:
            pass

def search_id():
    if Entry_5.get() == "":
        messagebox.showerror("Error", "Enter ID to search")
    else:
        id = int(Entry_5.get())
        cursor.execute("SELECT product_id FROM products")
        id_check = cursor.fetchall()
        dbconn.commit()
```

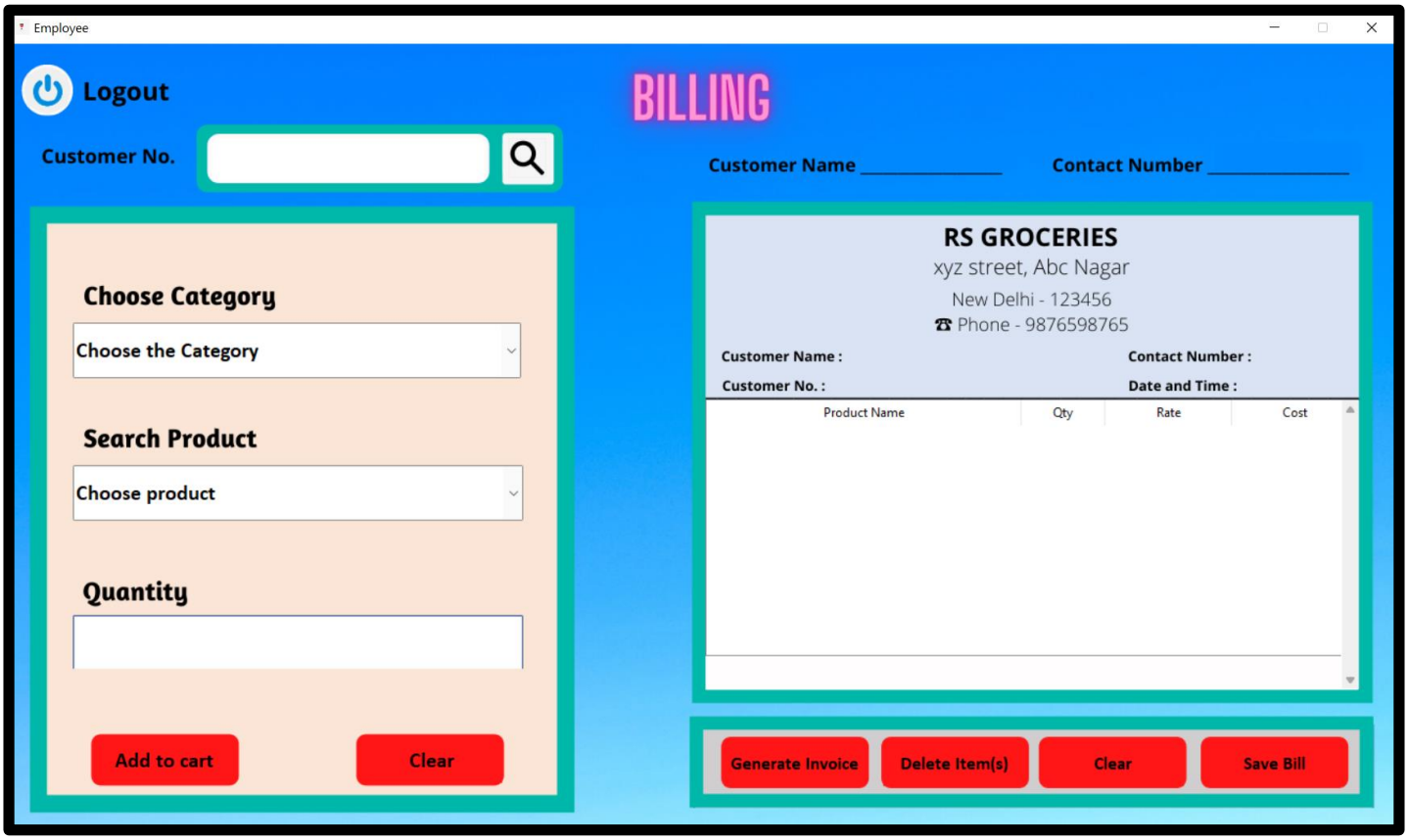
```
 for i in all_rows:
            if table.item(i).get("values")[0] == id:
                row.append(i)
        if row == []:
            messagebox.showerror("Error", "No product with ID {}".format(id))
        else:
            Button_1.configure(state="disabled")
            for j in row:
                values = table.item(j).get("values")
                Entry_4.insert(0, values[0])
                Entry_3.insert(0, values[1])
                Entry_2.insert(0, values[3])
                Entry_1.insert(0, values[2])

        Entry_5.delete(0, END)
        unwanted_cat()




# All Button Widgets
# Non-Table widgets
# Add to Market
Button_1 = Button(Admin, text="Add to market", relief="flat",
bg="#fe1716",fg="black",borderwidth=0,font=font_1,command=add_to_cart)
Button_1.configure(activebackground="#fe1716")
Button_1.place(relx=0.04325,rely=0.878,width=135,height=43)

# Modify
Button_2 = Button(Admin, text="Update", relief="flat", bg="#fe1716", fg="black",
borderwidth=0, font=font_1, command=update)
Button_2.configure(activebackground="#fe1716")
Button_2.place(relx=0.161, rely=0.878, width=135, height=43)

# Clear
Button_3 = Button(Admin, text="Clear", relief="flat", bg="#fe1716", fg="black",
borderwidth=0, font=font_1,command=clear)
Button_3.configure(activebackground="#fe1716")
Button_3.place(relx=0.278, rely=0.878, width=135, height=43)

# Search
search_img = PhotoImage(file="./images/search.png")
search_button = Button(Admin, image=search_img,
borderwidth=0,relief="flat",overrelief="flat", command=search_id)
search_button.place(relx=0.3713, rely=0.1175)
```

```
# Table widgets
# Select
Button_4 = Button(Admin, text="Select", relief="flat", bg="#fe1716", fg="black",
borderwidth=0, font=font_1,command=select_item)
Button_4.configure(activebackground="#fe1716")
Button_4.place(relx=0.512, rely=0.8855, width=135, height=43)

# Delete item(s)
Button_5 = Button(Admin, text="Delete item(s)", relief="flat",
bg="#fe1716",fg="black",borderwidth=0,font=font_1, command=delete_many)
Button_5.configure(activebackground="#fe1716")
Button_5.place(relx=0.686,rely=0.8855,width=135,height=43)

# Clear All
Button_6 = Button(Admin, text="Clear All", relief="flat", bg="#fe1716", fg="black",
borderwidth=0, font=font_1, command=clear_all)
Button_6.configure(activebackground="#fe1716")
Button_6.place(relx=0.862, rely=0.8855, width=135, height=43)


Admin.protocol("WM_DELETE_WINDOW", Exit)

Admin.mainloop()
```

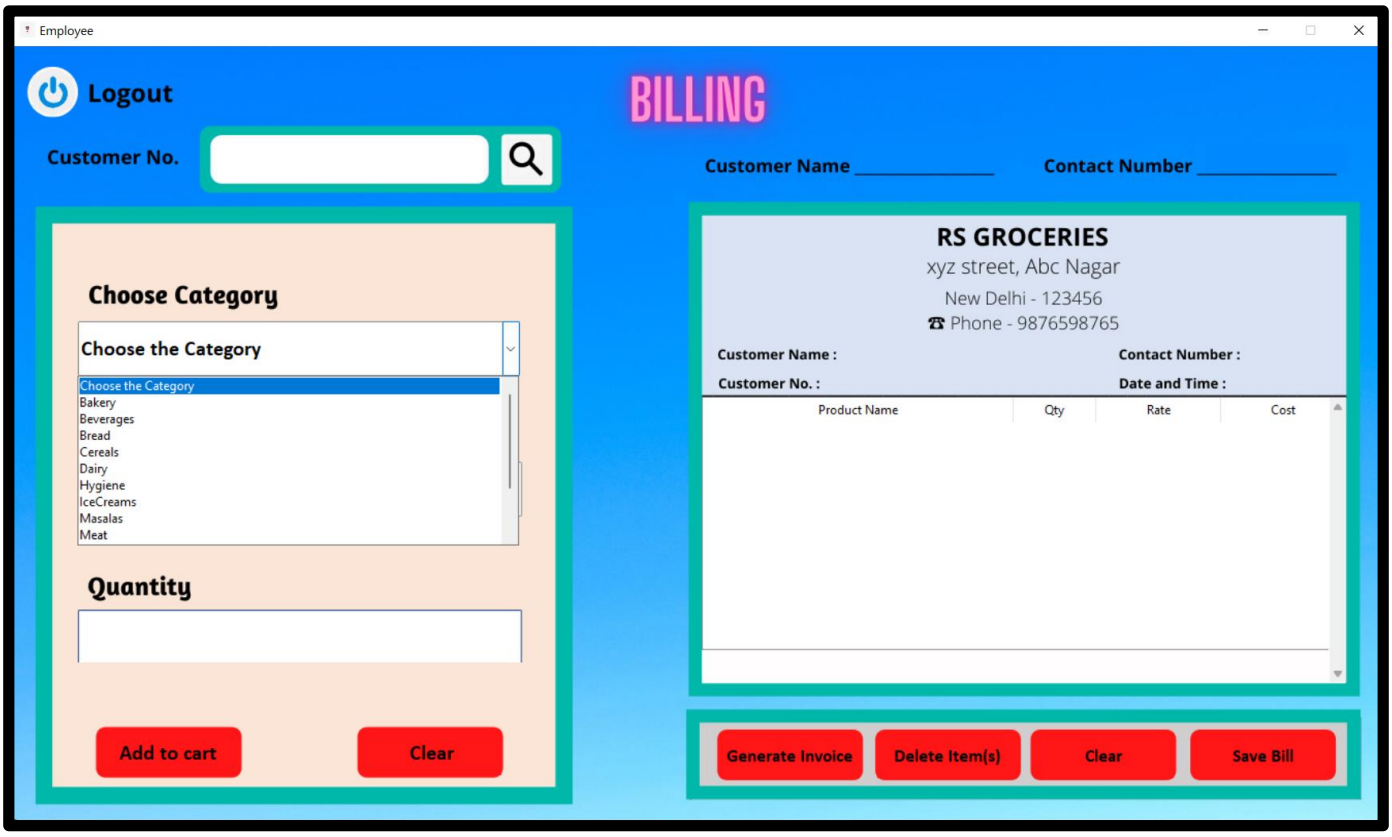
SNAP SHOTS

Running Main.py

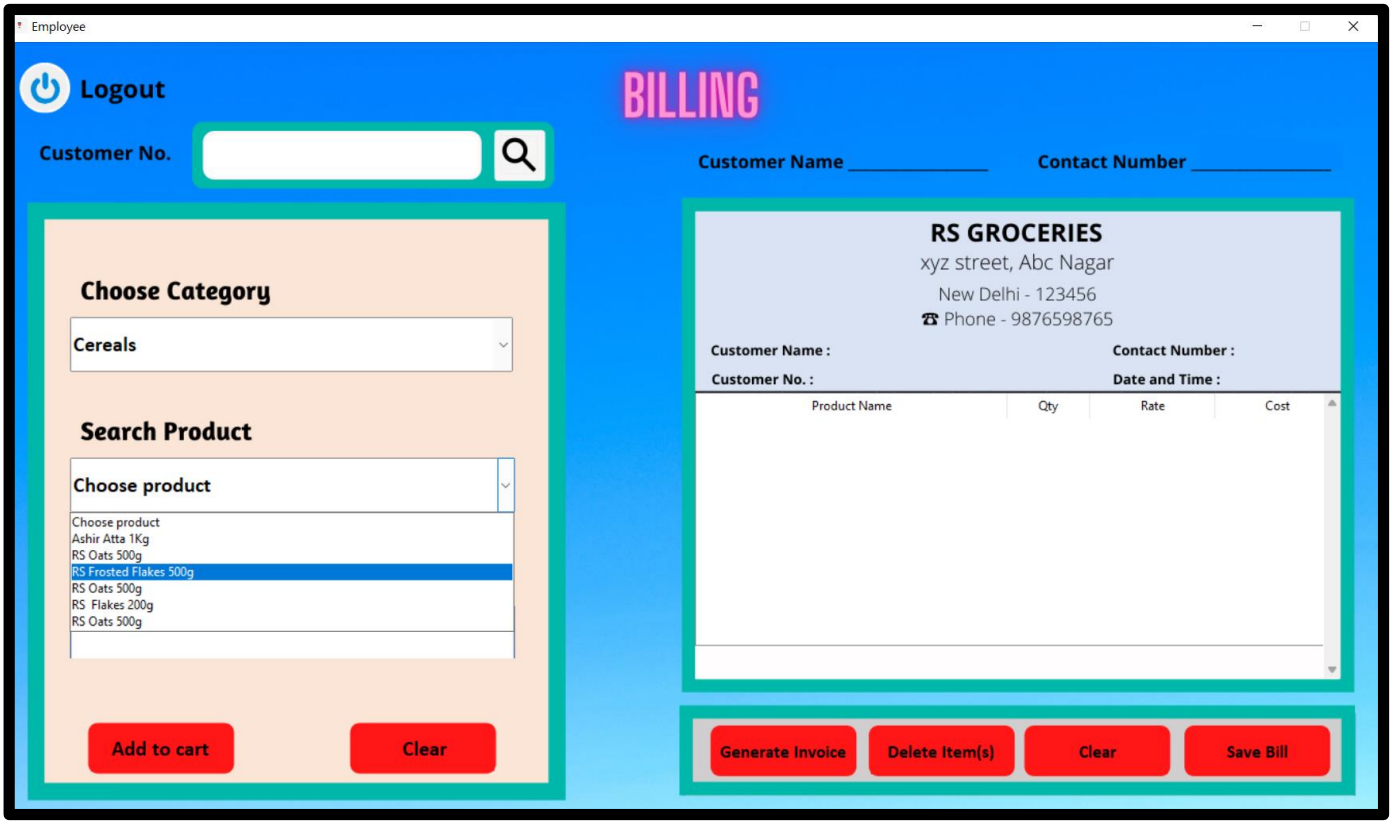
Clicking on Employee



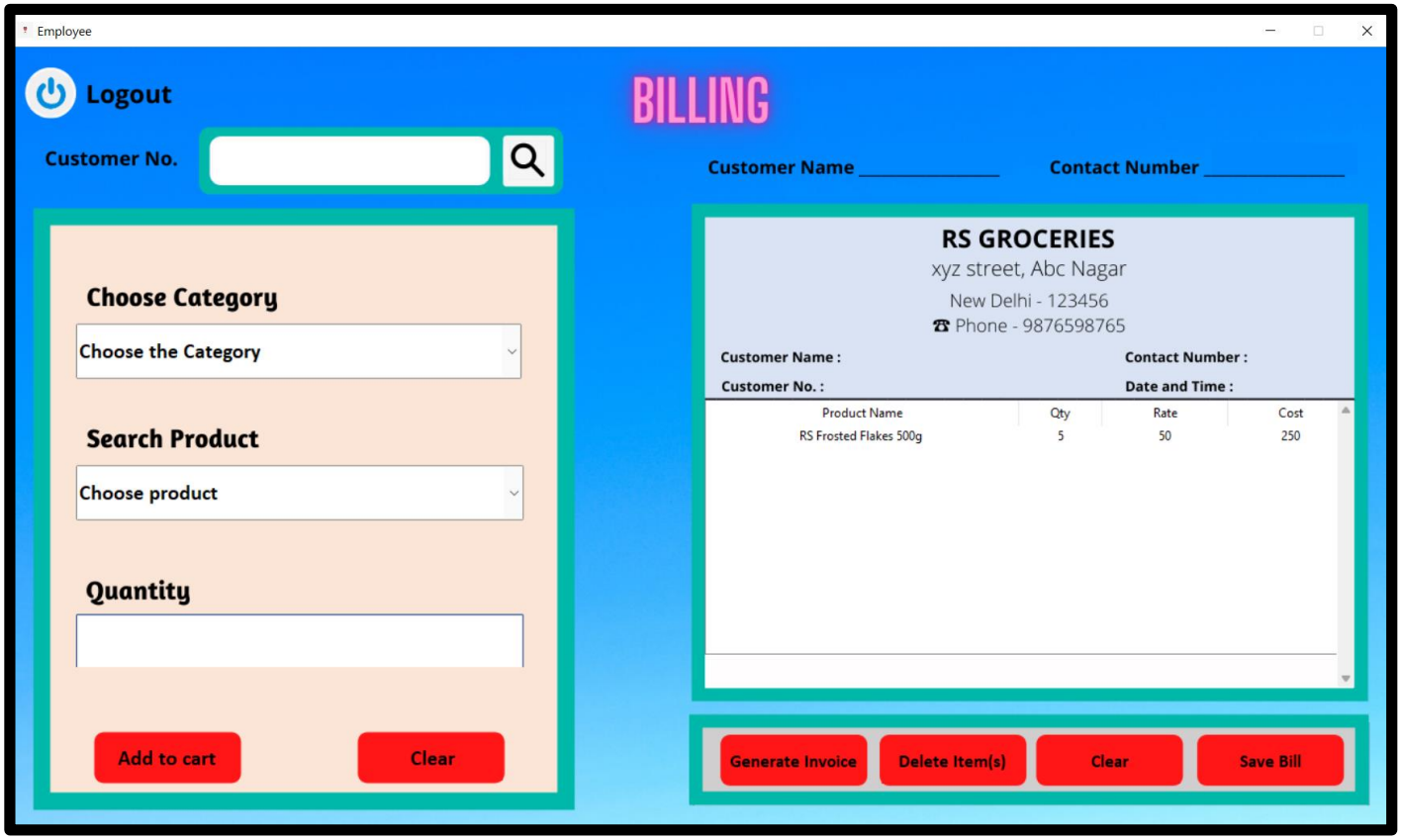
Selecting Category



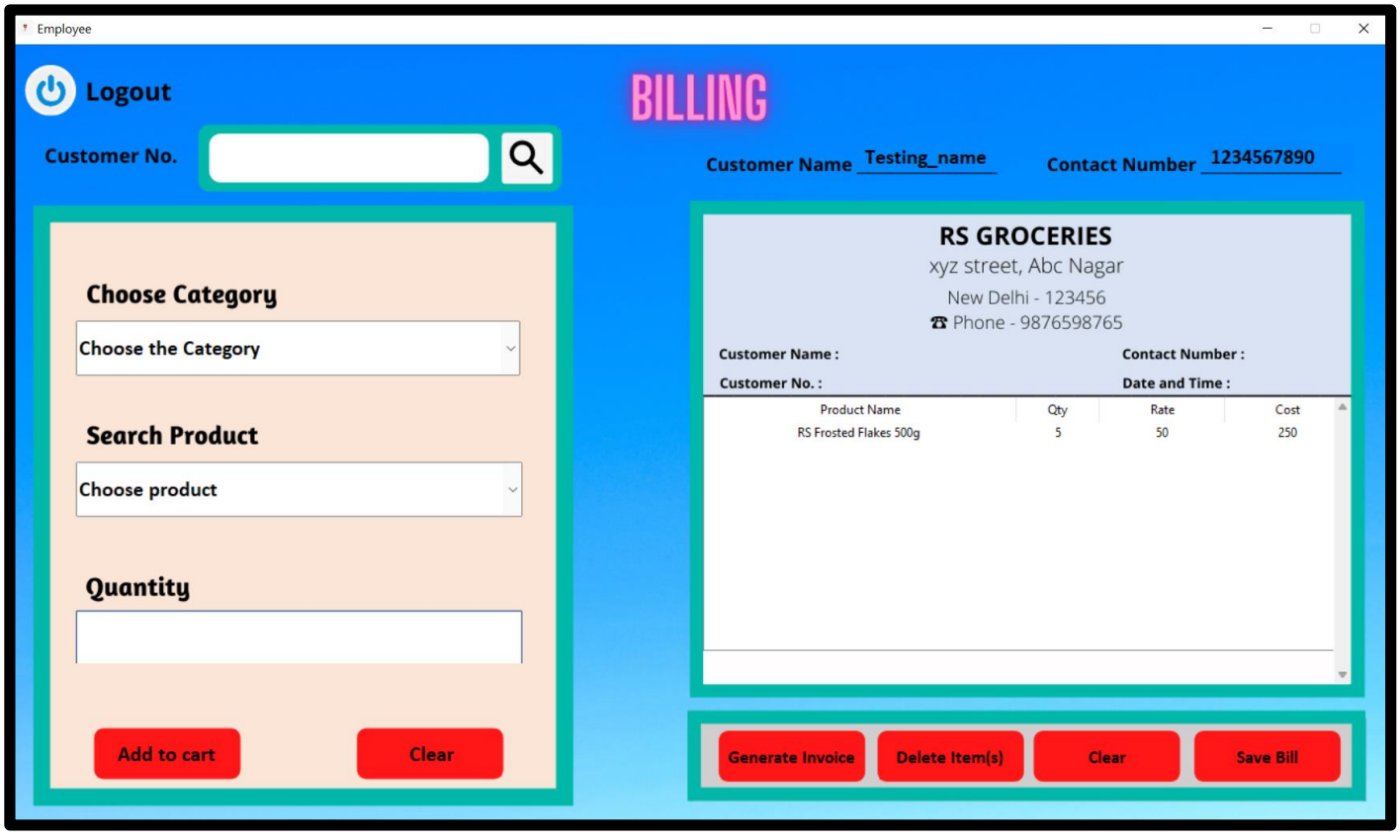
Selecting Product and Entering Quantity



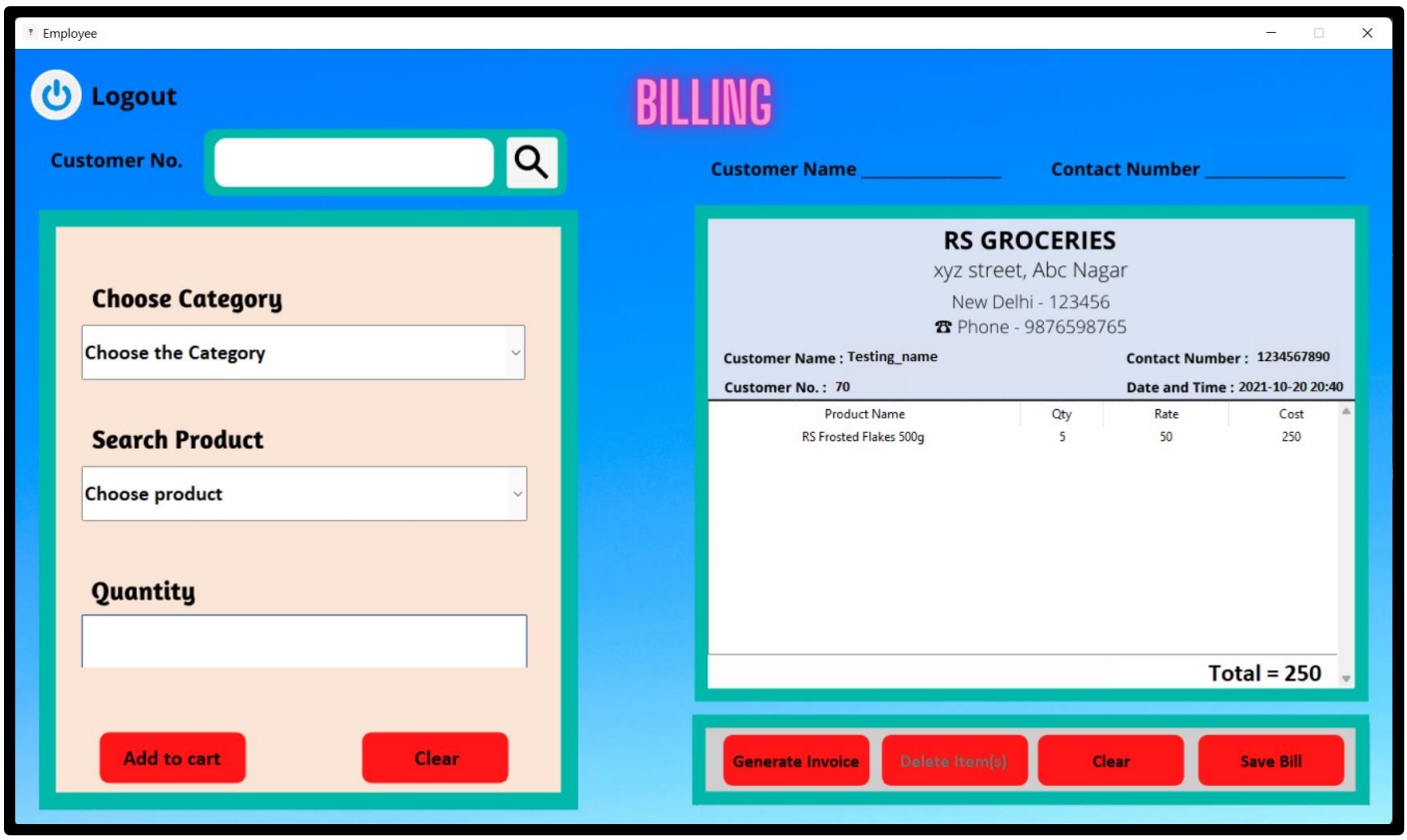
Adding it to the cart



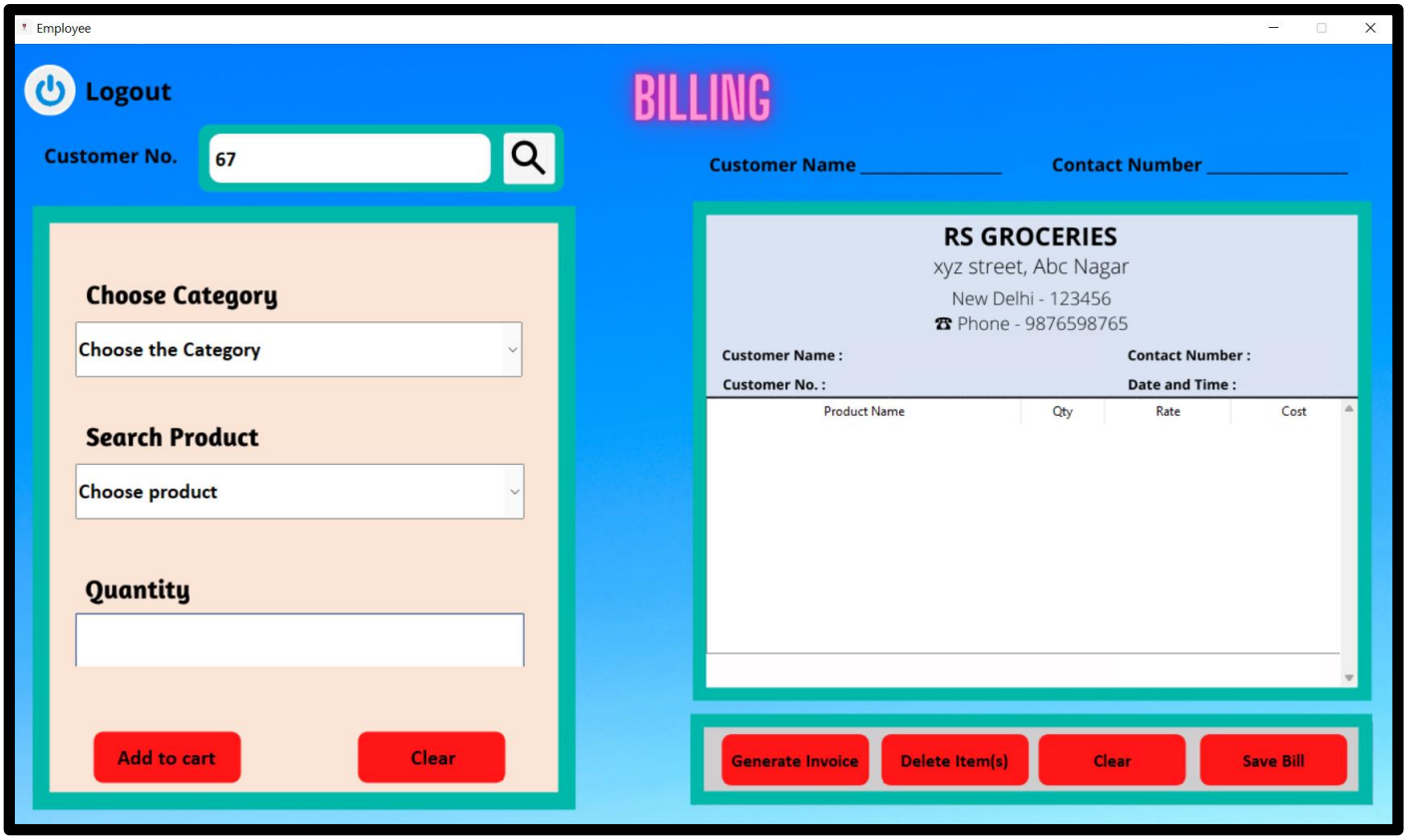
Entering Customer Name and Contact Number



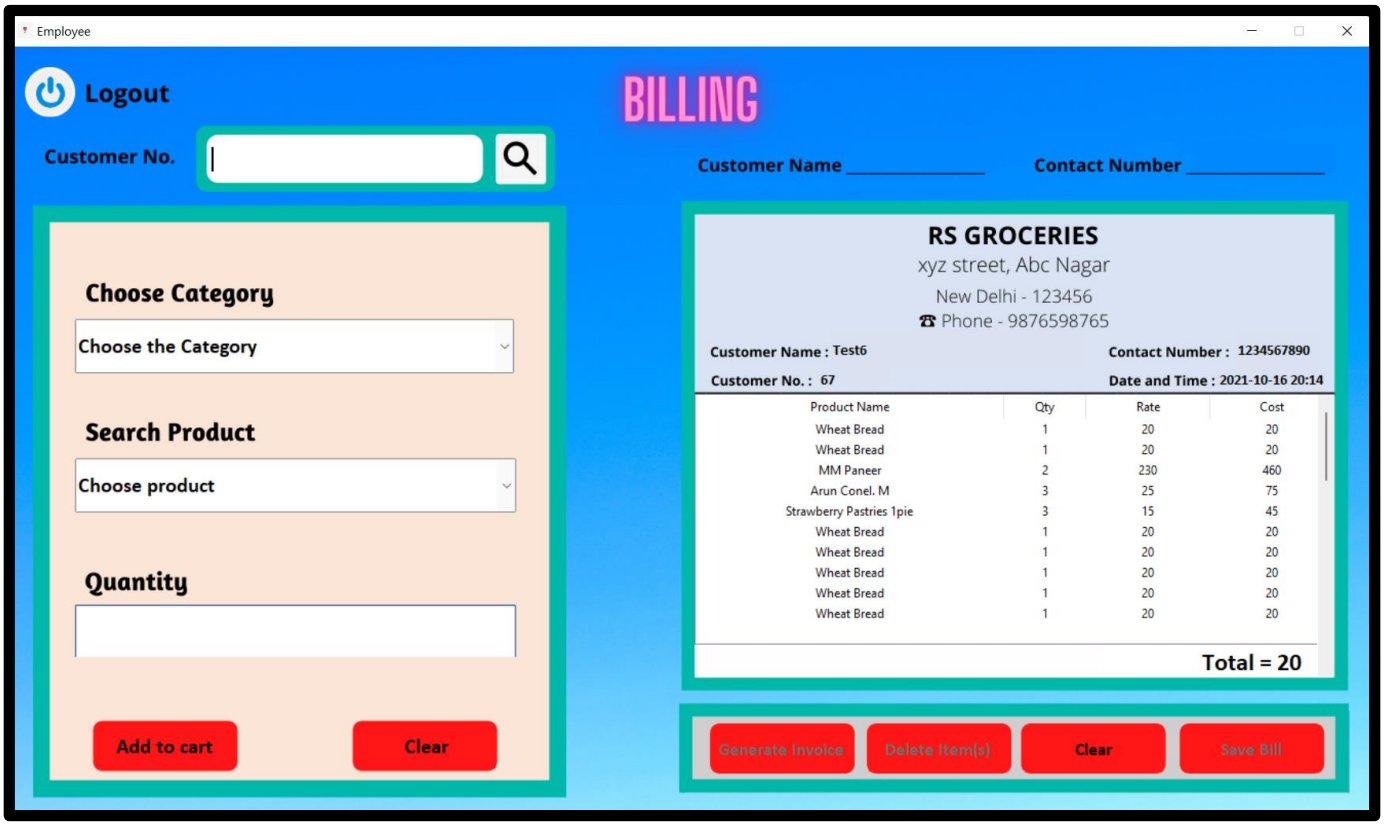
Generating Invoice



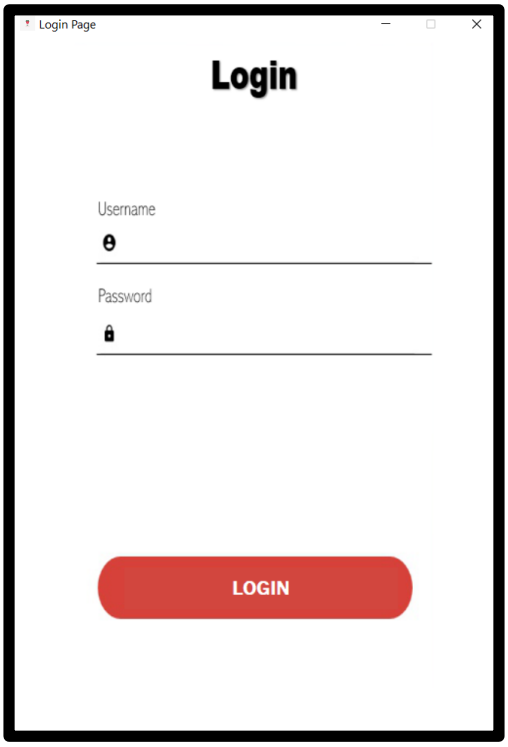
Searching the old bill



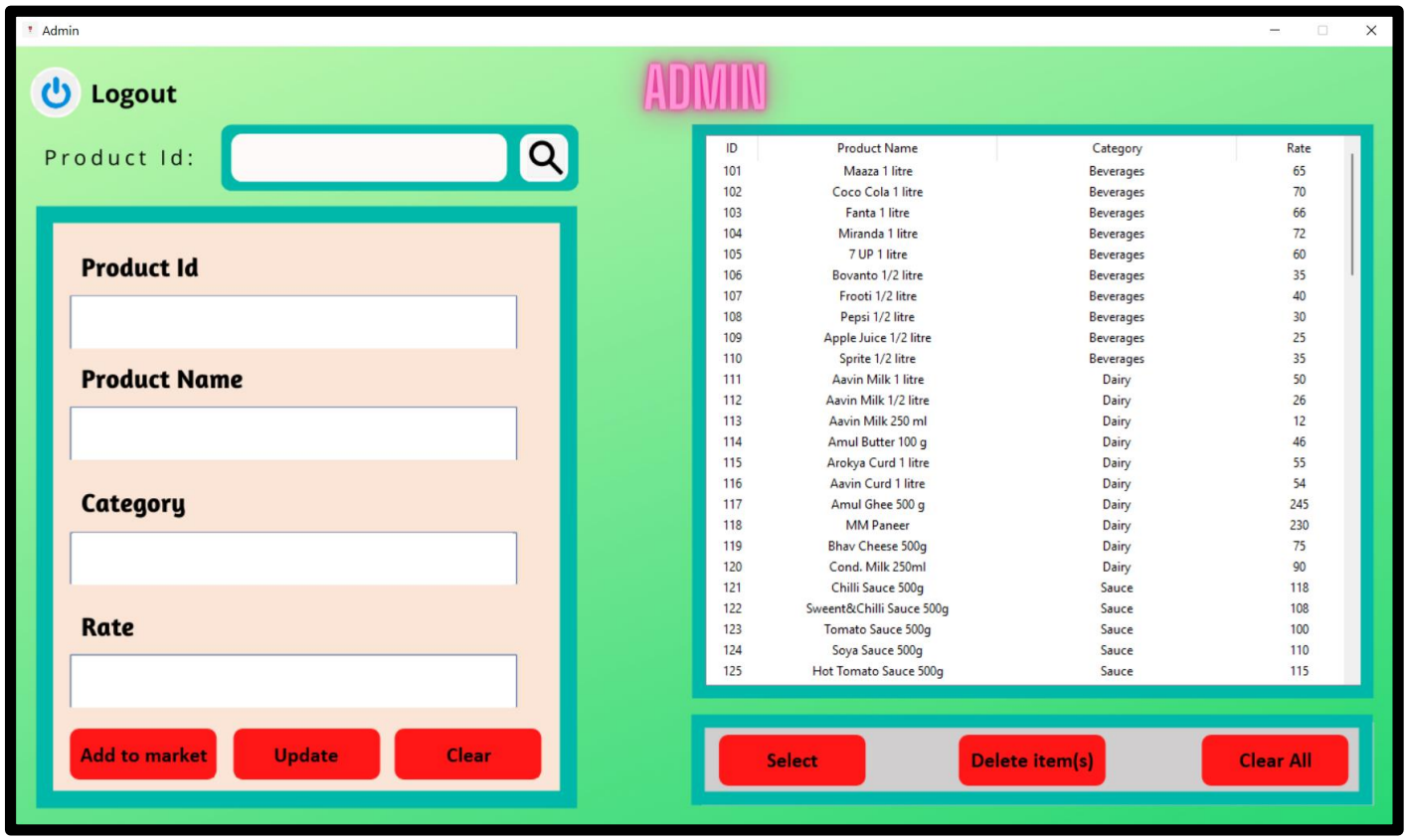
After the search



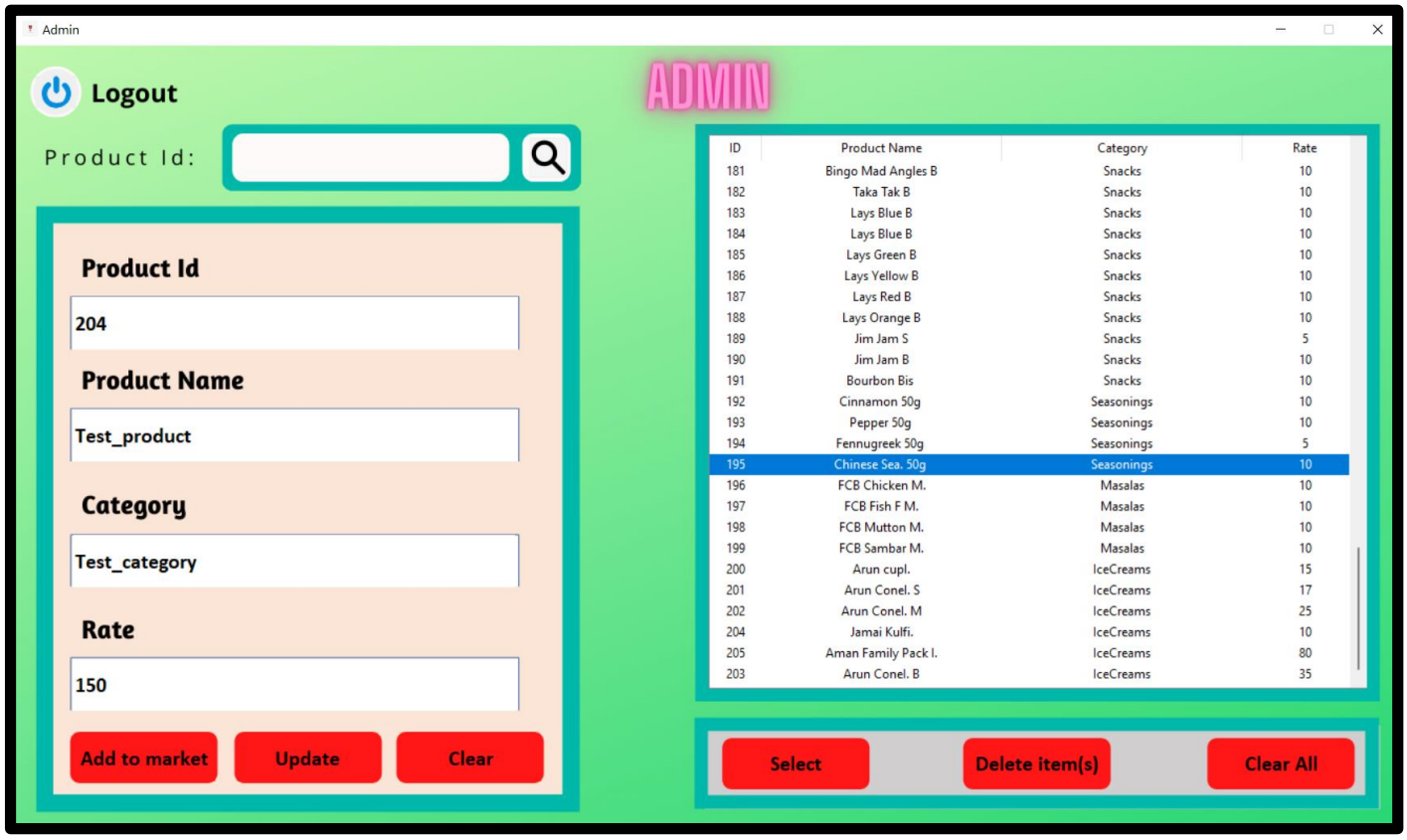
On Clicking Admin



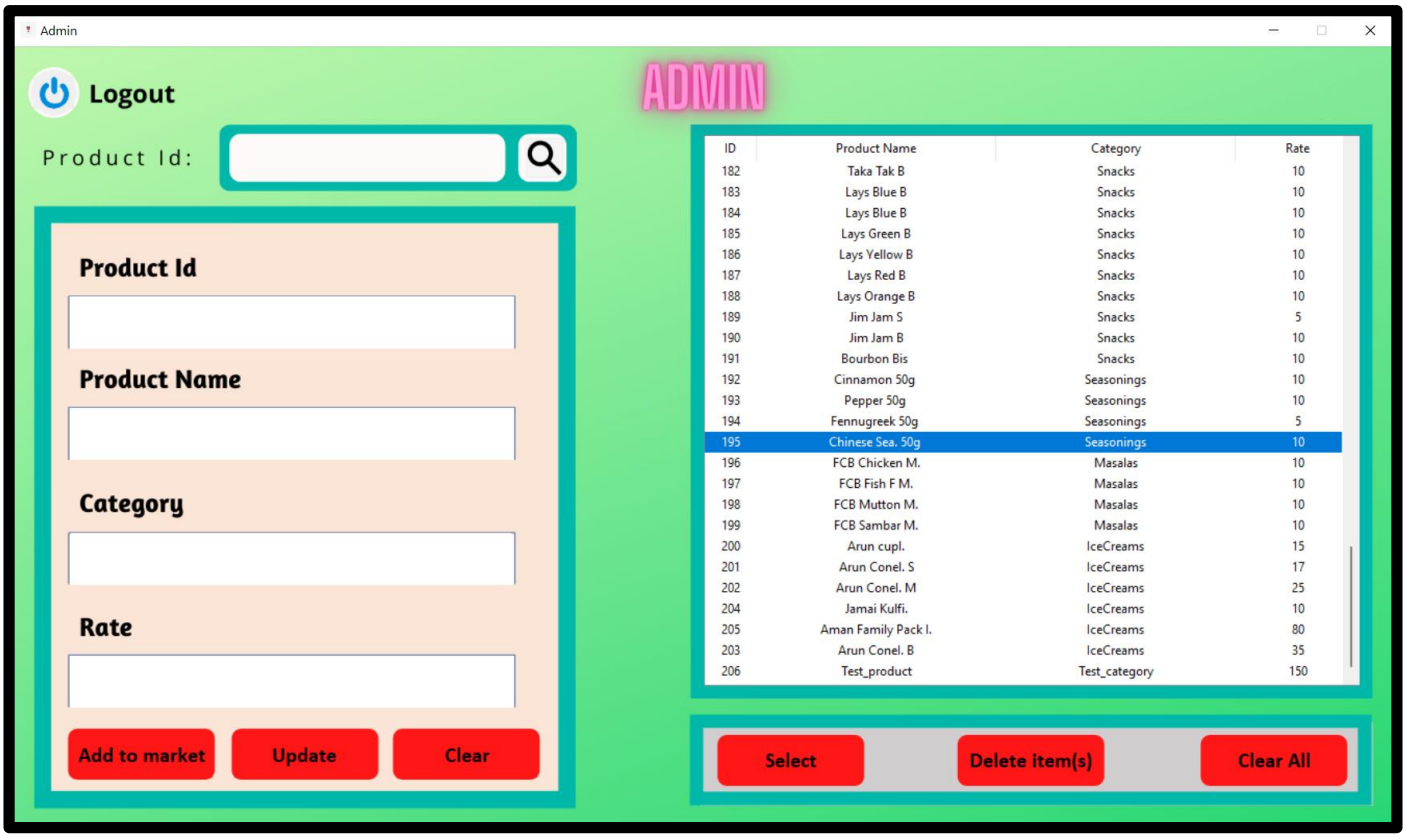
Admin Page



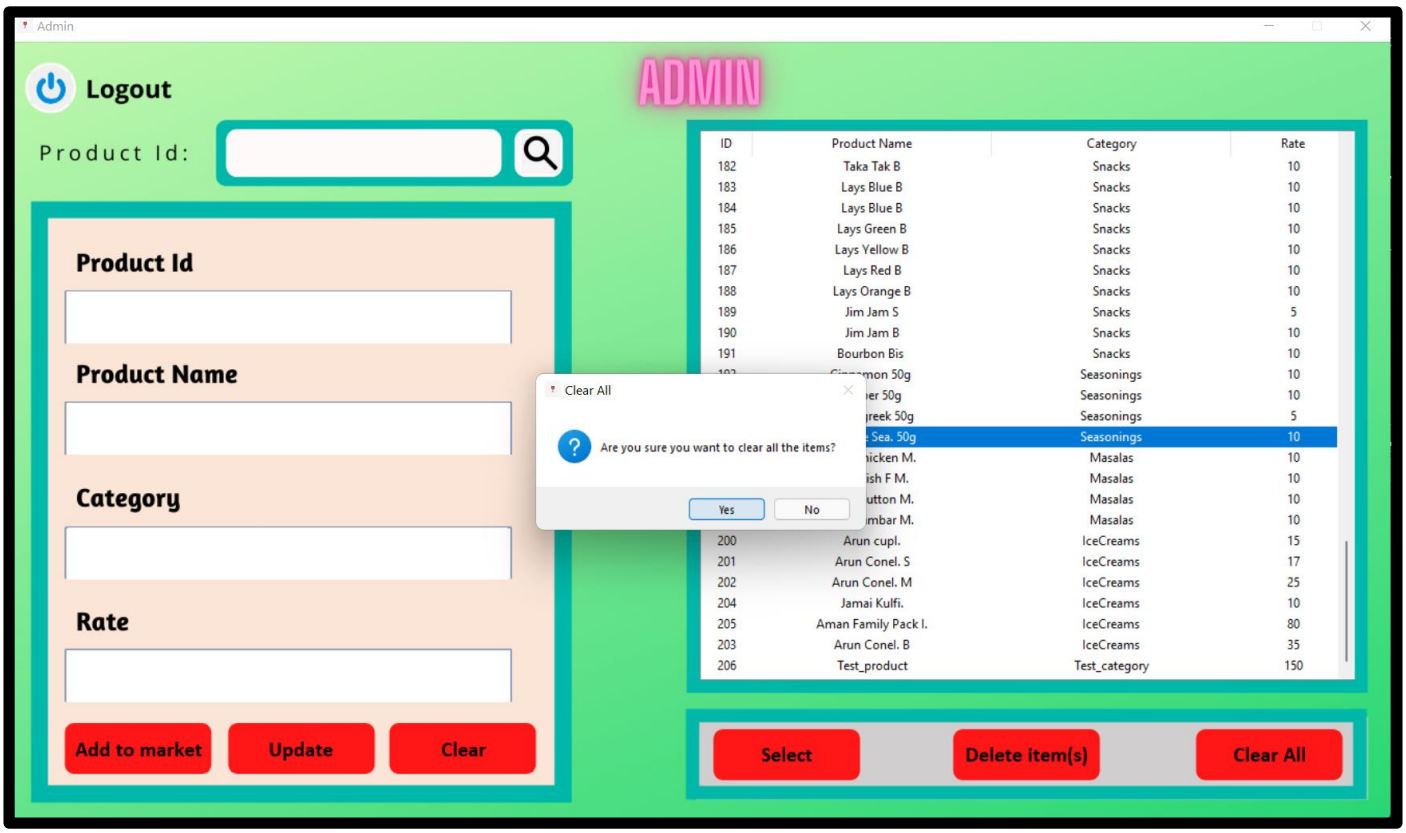
Entering all the Entries



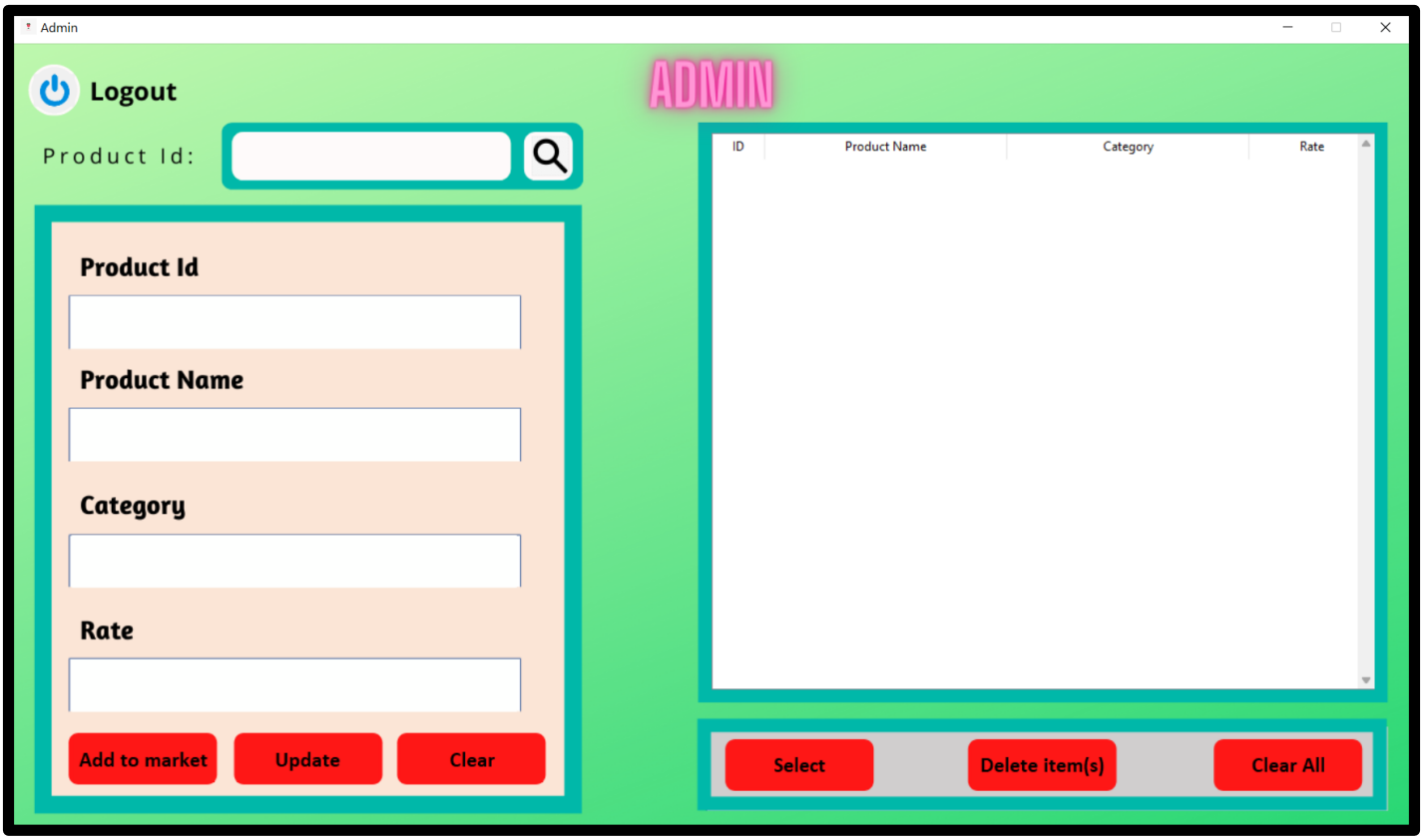
After adding



Clearing all the products



After Clearing



BIBLIOGRAPHY

- https://youtube.com/playlist?list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV (Course).
- https://youtube.com/playlist?list=PLu0W_9lII9ajLcqRcj4PoEihkukF_OTzA (Course).
- https://imagecolorpicker.com/ - Used this to get the hex color code.
- https://www.canva.com/ - Used this to design the backgrounds
- https://www.lucidchart.com/ - Used this to make flow chart

THANK YOU