

Universidad Mayor de San Andres
Facultad de Ciencias Puras y Naturales
Carrera de Informática



MINIPROYECTO 2
APLICACIONES CLIENTE / SERVIDOR

Integrantes : Univ. Altamirano Monasterios Felipe Ernesto
Univ. Quinaquina Limachi Alejandra
Univ. Quino Chipana Gueylor Amed

Docente: Lic. Gallardo Portanda Franz Ramiro

Paralelo: "A"

Asignatura: INF-273

Fecha: 08 de Mayo de 2023

LA PAZ - BOLIVIA

1. Introducción

Los sockets TCP (Transmission Control Protocol) son una herramienta fundamental en la programación de redes, estos nos permiten la comunicación bidireccional y confiable entre dos dispositivos en una red por medio de una conexión virtual punto a punto.

El protocolo TCP proporciona una comunicación confiable entre los dispositivos, ya que garantiza que los datos se entreguen correctamente y en el orden correcto, para lograr esto el protocolo TCP es quien se encarga de la verificación de errores y además de las retransmisiones de paquetes en caso de pérdida o daño de los datos durante la transmisión.

Los sockets TCP son una herramienta fundamental en la programación de redes que permiten la comunicación confiable entre dispositivos en una red, gracias al uso del protocolo TCP y su sistema de verificación de errores y retransmisión de paquetes.

La creación de programas o aplicaciones de facturación, es una tarea importante para cualquier empresa que desee llevar un control y registro de alguna transacción. En la actualidad, el uso de sockets TCP en Python es una opción y elección popular para la comunicación en red, ya que esta permite el intercambio de datos de manera confiable, eficiente y segura entre diferentes sistemas.

En este sentido, la creación de un programa de facturación utilizando sockets TCP en Python puede ser una solución efectiva para empresas que requieren la gestión de sus transacciones en red. Al utilizar sockets TCP, es posible enviar y recibir datos de manera segura y confiable, lo que garantiza la integridad de la información y la precisión en el registro de las transacciones.

Se busca crear una solución eficiente y escalable para la gestión de facturación en un supermercado. Esta aplicación permitirá a los usuarios generar facturas de manera automatizada, almacenarlas en una base de datos relacional y gestionarlas a través de tecnologías para la experiencia de usuario en tres niveles relevantes, como se el uso de Flask, para la vista y comparación a una aplicación web; python con Tkinter, simulando una experiencia de usuario para escritorio de un SO y consola, interacción mediante comandos y solicitudes como comandos de instrucción y ejecución . Utilizando sockets TCP para establecer la comunicación entre el cliente/servidor de manera eficiente y segura.

2. Objetivos propuestos

- Desarrollar una aplicación de generación de facturas que sea fácil de usar y eficiente en el procesamiento de datos.
- Proponer tres opciones para el proceso de facturación (web, programa de escritorio y consola) mediante una conexión entre cliente/servidor usando sockets TCP.
- Permitir a los usuarios generar facturas personalizadas con información relevante y opciones de configuración.
- Almacenar las facturas generadas en una base de datos para su posterior consulta y gestión.
- Establecer una comunicación eficiente entre el cliente y el servidor utilizando sockets TCP.

3. Software y bibliotecas del lenguaje requeridas (lado del cliente y del servidor)

Para el desarrollo de la aplicación se tomaron en cuenta las siguientes tecnologías:

3.1. Lado del cliente

- Python: Es un lenguaje de programación interpretado, de alto nivel y multiparadigma.
- Bibliotecas de Python:
 - **socket:** El módulo socket de Python proporciona funcionalidades para la comunicación mediante sockets. Permite establecer conexiones y enviar/recibir datos a través de redes utilizando el protocolo TCP/IP
 - **psycopg2:** es una biblioteca de Python utilizada para interactuar con bases de datos PostgreSQL. Proporciona una interfaz para conectarse a una base de datos PostgreSQL, ejecutar consultas SQL y obtener los resultados.
 - **Flask:** Se utiliza para crear aplicaciones web y proporciona funciones para manejar rutas, solicitudes y respuestas HTTP, y generar plantillas HTML dinámicas.
 - **requests:** Es una biblioteca de Python utilizada para realizar solicitudes HTTP. Proporciona una interfaz sencilla para enviar solicitudes HTTP GET, POST y otras, y recibir respuestas del servidor.
 - **Tkinter:** Tkinter es un módulo de la biblioteca estándar de Python que proporciona una interfaz gráfica de usuario (GUI) para aplicaciones de escritorio. Con Tkinter, se pueden crear ventanas, botones, cuadros de texto, menús desplegables, gráficos y otros widgets que permiten al usuario interactuar con el programa de manera visual e intuitiva.

3.2. Lado del servidor

- **Python:** Es un lenguaje de programación interpretado, de alto nivel y multiparadigma.
- **Postgres:** PostgreSQL es un sistema de gestión de bases de datos relacional (RDBMS, por sus siglas en inglés) de código abierto y robusto. Es conocido por su confiabilidad, escalabilidad y capacidad para manejar grandes volúmenes de datos. PostgreSQL se basa en el modelo relacional y utiliza el lenguaje SQL para gestionar y manipular los datos almacenados en la base de datos.
- **Mysql (Programa de Escritorio y Consola):** MySQL sistema de gestión de bases de datos relacional (RDBMS) de código abierto, es una herramienta esencial para la gestión de datos en aplicaciones web y de servidor.
- **Bibliotecas**
 - **socket:** El módulo socket de Python proporciona funcionalidades para la comunicación mediante sockets. Permite establecer conexiones y enviar/recibir datos a través de redes utilizando el protocolo TCP/IP.

- **psycopg2:** Es una biblioteca de Python utilizada para interactuar con bases de datos PostgreSQL. Proporciona una interfaz para conectarse a una base de datos PostgreSQL, ejecutar consultas SQL y obtener los resultados.
- **threading:** El módulo threading de Python proporciona clases y funciones para la programación concurrente. Permite crear y administrar hilos (threads) independientes dentro de un programa, lo que permite ejecutar múltiples tareas de forma simultánea y aprovechar mejor los recursos del sistema.

4. Descripción de la solución

La aplicación de generación de facturas se desarrollará utilizando Python como lenguaje principal. En el lado del servidor, se utilizará el framework Flask para desarrollar la interfaz web que permitirá a los usuarios acceder a la aplicación y gestionar sus facturas. Se utilizará Postgres como base de datos para almacenar las facturas generadas.

El proceso de generación de facturas se realizará de manera automatizada, permitiendo a los usuarios personalizar las facturas con la información relevante, como el nombre del cliente, los productos adquiridos y los precios correspondientes. La comunicación entre el cliente y el servidor se establecerá mediante sockets TCP. Esto permitirá una transferencia de datos eficiente y segura entre ambos extremos.

Para implementar la aplicación de facturación automatizada que almacene las facturas en una base de datos Postgres y las gestione a través de una interfaz web desarrollada con Flask utilizando sockets TCP, seguiremos los siguientes pasos:

Desarrollo consola.

Para el desarrollo de la aplicación basada en una facturación automatizada por consola se hizo uso de sockets TCP el cual nos ayuda establecer una comunicación entre cliente servidor, además la aplicación cuenta con una base de datos en sql y almacenado en phpmyadmin el cual maneja los datos de los productos ofrecidos al cliente, para la elaboración de tal aplicación se describira el proceso realizado.

Descripción de aplicación por consola por lado del cliente

Para comenzar con la implementación primeramente se iniciará con una modelo de conexión mediante sockets TCP entre cliente servidor.

Programa del lado del servidor

```
Users > E > Documents > Universidad > LAB273 > proy_1 > serverTCP.py > ...  
from socket import*                                #importando socket  
  
addr = ("localhost", 7777)                          #creando direccion y puerto  
serverSock = socket(AF_INET, SOCK_STREAM)           #creando socket  
serverSock.bind((addr))                             #vinculando dir y puerto con socket  
serverSock.listen()                                 #iniciando modo escucha //esperando conexiones  
  
while True:  
    sockClient, add = serverSock.accept()           #aceptando cliente  
    petition= sockClient.recv(512)                 #recibiendo petition  
    mensaje = petition.decode()                    #decodificando mensaje  
  
    sockClient.send(mensaje.encode())              #enviando mensaje codificado  
    sockClient.close()
```

Programa del lado del cliente

```
from socket import *                                #importando socket  
  
addr = ("localhost", 7777)                          # creando direccion y puerto  
sockClient = socket(AF_INET, SOCK_STREAM)           # creando socket  
sockClient.connect((addr))                          # conectando al servidor  
  
solicitud = input("Intro mensaje: ")                #leyendo mensaje  
sockClient.send(solicitud.encode())                 #enviando emnsaje y codificandolo con encode  
  
resp = sockClient.recv(512)                         #reccibiendo mensaje del servidor  
print(resp.decode())                                #decodificando y mostrando  
sockClient.close()
```

Esta es la representación de una conexión cliente/servidor mediante sockets TCP, como sabemos que una factura requiere de los datos personales del usuario que realiza la compra, como primer paso pediremos al cliente que nos otorgue tales datos para la emisión de la factura.

```
import datetime  
from socket import *                                #importando socket  
  
addr = ("localhost", 7777)                          # creando direccion y puerto  
sockClient = socket(AF_INET, SOCK_STREAM)           # creando socket  
sockClient.connect((addr))                          # conectando al servidor  
print("\n")  
print("*****Ingrese datos del Usuario*****")  
usuario =input("Nombre del Usuario: "+"\\n"+"\\t")  
ci= input("Carnet de identidad"+"\\n"+"\\t")  
fecha=input("fecha emision de factura"+"\\n"+"\\t")  
hora = datetime.datetime.now()
```

Como siguiente paso se hizo la implementación de los productos en oferta del supermercado, el cual nos mostrará los productos que se ofrecen al cliente, el programa también pedirá que se digite un número con la opción del producto a comprar y la cantidad deseada por el usuario.

Con el siguiente código mostramos una tabla de los productos disponibles para la compra del usuario del usuario

```
print("*****SUPERMERCADO*****")
productos = [
    {
        "opcion": "1",
        "producto": "Macarrones",
        "precio": 9.5,
    },
    {
        "opcion": "2",
        "producto": "Leche",
        "precio": 5.00,
    },
    {
        "opcion": "3",
        "producto": "Arroz",
        "precio": 6.00,
    },
    {
        "opcion": "4",
        "producto": "mantequilla",
        "precio": 10.00,
    },
    {
        "opcion": "5",
        "producto": "sal",
        "precio": 7.5,
    },
    {
        "opcion": "6",
        "producto": "galletas",
        "precio": 12.5,
    },
    {
        "opcion": "7",
        "producto": "refresco",
        "precio": 13.00,
    },
    {
        "opcion": "8",
        "producto": "frituras",
    },
]
```

```

        "precio": 8.00,
    },
    {
        "opcion": "9",
        "producto": "doritos",
        "precio": 15.00,
    },
    {
        "opcion": "10",
        "producto": "aceitunas",
        "precio": 9.50,
    },
    {
        "opcion": "11",
        "producto": "azucar",
        "precio": 6.5,
    },
    {
        "opcion": "12",
        "producto": "pollo",
        "precio": 50.00,
    },
    {
        "opcion": "13",
        "producto": "pescado",
        "precio": 30.00,
    },
    {
        "opcion": "14",
        "producto": "atun",
        "precio": 15.00,
    },
    {
        "opcion": "15",
        "producto": "mortadela",
        "precio": 12.00,
    },
]

print("+-----*-----+-----+")
print("| OPCION | PRODUCTO          | PRECIO    |")
print("+-----*-----+-----+")
for dato in productos:

```

```

        product = dato["producto"]
        precio = dato["precio"]
        opcion=dato["opcion"]
        cadena =
"|{:<8}|{:<16}|{:>10.2f}|".format(opcion,product,
precio)
        print(cadena)
        print("+-----*-----+-----+")

print("-----SI DESEA TERMINAR LA COMPRA PRECIONE
0-----"+"\\n")

```

para la opción que reciba el producto y la cantidad a comprar se inicializaran dos variables con el mismo el de opción y cantidad, los dos datos enviamos al servidor con el `socketClient.send((opcion+" "+cantidad).encode())` nos ayudarán a la decodificación de los datos enviados y como tal recibiremos una respuesta del lado del servidor.

```

while True:
    opc = input(" Elija alguna opcion para su compra
----->> ")
    while True:
        if opc == "1":
            producto = "macarrones"
            break
        elif opc == "2":
            producto = "leche"
            break
        elif opc == "3":
            producto = "arroz"
            break
        elif opc == "4":
            producto = "mantequilla"
            break
        elif opc == "5":
            producto = "sal"
            break
        elif opc == "6":
            producto = "galletas"
            break
        elif opc == "7":

```



```

        producto = "refresco"
        break
    elif opc == "8":
        producto = "frituras"
        break
    elif opc == "9":
        producto = "doritos"
        break
    elif opc == "10":
        producto = "aceitunas"
        break
    elif opc == "11":
        producto = "azucar"
        break
    elif opc == "12":
        producto = "pollo"
        break
    elif opc == "13":
        producto = "pescado"
        break
    elif opc == "14":
        producto = "atun"
        break
    elif opc == "15":
        producto = "mortadela"
        break
    elif opc == "0":
        producto = "0"
        break
    else:
        opc = input("elija una opcion correcta")

    if producto == "0":
        sockClient.send((opc).encode())
        break
    else:
        cantidad = input("Introduzca la cantidad deseada ---->> ")
        print("\t"+"Registrado --->> "+ producto+" "+cantidad)
        sockClient.send((producto + " "+cantidad).encode())

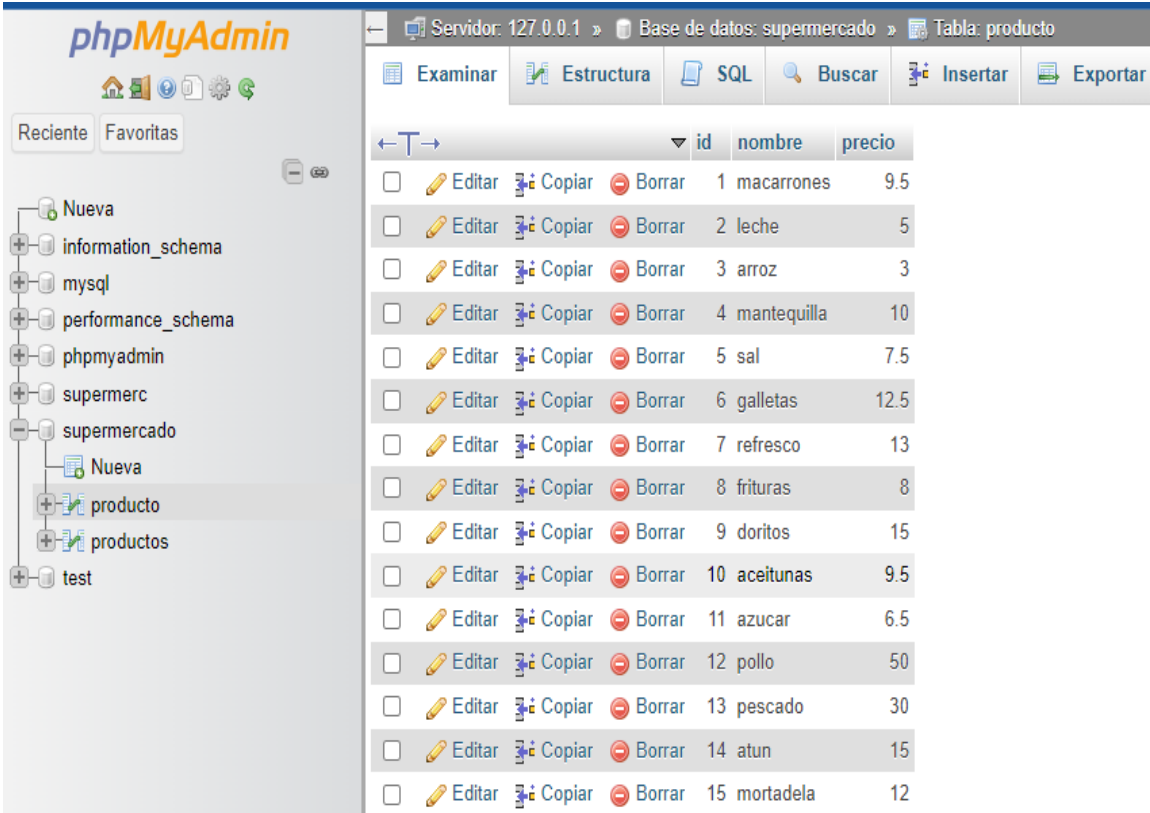
```

Esta parte del código nos ayudará a registrar las compras del usuario, así también nos ayudará con la verificación de las opciones introducidas para la comprar y también verifica si la opción es válida, caso contrario está pedirá que se introduzca un valor correcto.

Para terminar una compra se pide se digite el número 0, tal acción conlleva a que el socketClient envíe y decodifique el dato opción al servidor.

Descripción de aplicación por consola por lado del servidor

Para el lado del servidor primero esperamos la conexión con el cliente y recibimos una petición además mensaje decodifica los datos de opción y una cantidad en un vector los cuales los separaremos por un espacio para separar en un index [0] el producto y en un index [1] la cantidad, para hacer el respectivo cálculo de un total de la compra y además de conseguir el precio unitario del producto se hará uso de una base de datos, la herramienta que utilizaremos es el phpMyAdmin en el cual tendremos una tabla con el nombre de los productos, código y precio unitario.



	id	nombre	precio
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	1	macarrones	9.5
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	2	leche	5
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	3	arroz	3
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	4	mantequilla	10
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	5	sal	7.5
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	6	galletas	12.5
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	7	refresco	13
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	8	frituras	8
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	9	doritos	15
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	10	aceitunas	9.5
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	11	azucar	6.5
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	12	pollo	50
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	13	pescado	30
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	14	atun	15
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	15	mortadela	12

Para trabajar con esta base de datos tendremos que instalar una librería que permite trabajar a python mediante una conexión con mysql, para la instalación se deberá usar el comando:

```
pip install mysql-connector-python
```

También tendremos un archivo punto py el cual tendrá por nombre conexión, la función de esta clase nos permitirá inicializar la función que contenga nuestra base de datos, además tendrá dos clases una recuperar y otra con el nombre datosProducto, la primera función nos ayudará a realizar una consulta de todos los datos de nuestra base de datos, la siguiente función nos permite realizar una consulta con el nombre del producto dentro de nuestra base de datos y nos devuelve el registro del producto seleccionado.

Código de la clase Conexion para el uso de la base de datos

```

import mysql.connector

class Conexion:

    def __init__(self):
        self.conexion=mysql.connector.connect(
            host="localhost",
            user="root",
            passwd="",
            database="supermercado"
        )
        self.cursor1=self.conexion.cursor()

    def recuperar(self):
        consulta = "select * from producto"
        self.cursor1.execute(consulta)
        return self.cursor1.fetchall()

    def datosProducto(self,nom):
        consulta = f"select * from producto where nombre = '{nom}'"
        self.cursor1.execute(consulta)
        return self.cursor1.fetchone()

```

Con la conexión de nuestro servidor con la base de datos podremos hacer uso de los datos del precio unitario de los productos y todos los datos que contenga, con los datos ya generados tendremos la disposición de hacer un total del precio con todos los productos de la compra del usuario, además que se mandara toda la información al cliente para generar la factura

Código de la clase servidor

```

from socket import * #importarndo
socket
from Conexion import *

addr = ("localhost", 7777) #creando direccion y puerto
serverSock = socket(AF_INET,SOCK_STREAM) #creando socket
serverSock.bind((addr)) #vinculando dir y puerto con
socket
serverSock.listen() #iniciando modo escucha
//esperando conexiones

```

```

while True:
    sockClient, add = serverSock.accept()
    conex = Conexion()
    total =0
    parcial=0
    mensajeParcial=""
    mensajeFactura=""

    while True:
        peticion = sockClient.recv(512)
        mensaje=peticion.decode()
        vec= mensaje.split(" ")
        producto = vec[0]
        if producto != "0":
            cantidad = vec[1]

registroProducto=conex.datosProducto(producto)
            precioUnitario = int(registroProducto[2])
            parcial = float(cantidad)*precioUnitario
            mensajeParcial = "ID:
"+str(registroProducto[0])+"|Producto: "+producto+ " = "+
str(precioUnitario)+" | Cantidad: "+ str(cantidad)+"| Total:
"+str(parcial)

            mensajeFactura= mensajeFactura +
mensajeParcial+"\n"
            total=total+parcial

        else:
            mensaje=mensajeFactura+"\n"+"| TOTAL=
"+str(total)

            sockClient.send(mensaje.encode())
            sockClient.close()
            break

```

Ejecución de la aplicación por consola

```
C:\Users\E\Documents\Universidad\LAB273\prueba\factura>py cli
```

```
*****Ingrese datos del Usuario*****
```

```
Nombre del Usuario:
```

```
Ana
```

```
Carnet de identidad
```

```
1526282
```

```
fecha emision de factura
```

```
20/03/2023
```

```
*****SUPERMERCADO*****
```

OPCION	PRODUCTO	PRECIO

1	Macarrones	9.50

2	Leche	5.00

3	Arroz	6.00

4	mantequilla	10.00

5	sal	7.50

6	galletas	12.50

7	refresco	13.00

8	frituras	8.00

9	doritos	15.00

10	aceitunas	9.50

11	azucar	6.50

12	pollo	50.00

13	pescado	30.00

14	atun	15.00

15	mortadela	12.00

```
-----SI DESEA TERMINAR LA COMPRA PRECIONE 0-----
```

```
*****REGISTRO*****
```

```
Elija alguna opcion para su compra ---->> 2
Introduzca la cantidad deseada ----->> 4
Registrado --->> leche 4
Elija alguna opcion para su compra ---->> 4
Introduzca la cantidad deseada ----->> 2
Registrado --->> mantequilla 2
Elija alguna opcion para su compra ---->> 6
Introduzca la cantidad deseada ----->> 3
Registrado --->> galletas 3
Elija alguna opcion para su compra ---->> 11
Introduzca la cantidad deseada ----->> 2
Registrado --->> azucar 2
Elija alguna opcion para su compra ---->> 14
Introduzca la cantidad deseada ----->> 2
Registrado --->> atun 2
Elija alguna opcion para su compra ---->> 15
Introduzca la cantidad deseada ----->> 2
Registrado --->> mortadela 2
Elija alguna opcion para su compra ---->> 0
```

Emisión de la factura

```
*****REGISTRO*****

Elija alguna opcion para su compra ---->> 2
Introduzca la cantidad deseada ----->> 4
    Registrado --->> leche 4
Elija alguna opcion para su compra ---->> 4
Introduzca la cantidad deseada ----->> 2
    Registrado --->> mantequilla 2
Elija alguna opcion para su compra ---->> 6
Introduzca la cantidad deseada ----->> 3
    Registrado --->> galletas 3
Elija alguna opcion para su compra ---->> 11
Introduzca la cantidad deseada ----->> 2
    Registrado --->> azucar 2
Elija alguna opcion para su compra ---->> 14
Introduzca la cantidad deseada ----->> 2
    Registrado --->> atun 2
Elija alguna opcion para su compra ---->> 15
Introduzca la cantidad deseada ----->> 2
    Registrado --->> mortadela 2
Elija alguna opcion para su compra ---->> 0

*****DATOS DE LA FACTURA*****

-----Datos del Cliente-----
Usuario: Ana      CI: 1526282
Fecha: 20/03/2023      Hora: 04:10:53
-----Datos de la compra-----
ID: 2|Producto: leche = 5 | Cantidad: 4| Total: 20.0
ID: 4|Producto: mantequilla = 10 | Cantidad: 2| Total: 20.0
ID: 6|Producto: galletas = 12 | Cantidad: 3| Total: 36.0
ID: 11|Producto: azucar = 6 | Cantidad: 2| Total: 12.0
ID: 14|Producto: atun = 15 | Cantidad: 2| Total: 30.0
ID: 15|Producto: mortadela = 12 | Cantidad: 2| Total: 24.0

| TOTAL= 142.0

C:\Users\E\Documents\Universidad\LAB273\prueba\factura>
```

Desarrollo escritorio

Para integrar el programa que funcionara como aplicacion de Escritorio, implementaremos la libreria Tkinter, primero importamos la librería en ambos archivos de Python (cliente y servidor), y luego crear la interfaz de usuario utilizando widgets de Tkinter.

A continuación, se muestra como el el código del lado del servidor y del Cliente:

LADO CLIENTE

```
import tkinter as tk
from tkinter import ttk
import mysql.connector
import socket
import json
from tkinter import messagebox
```

```

diccionario_prod={}
a=""
root = tk.Tk()
root.title("LAB273, MINIPROYECTO 2")

root.geometry("750x400")
my_connect = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="",
    database="supermercado"
)
my_conn = my_connect.cursor()
my_conn.execute("SELECT * FROM productos")
productos_bd = my_conn.fetchall()
tree = ttk.Treeview(root, columns=("Cantidad",))
tree.heading("#0", text="Producto")
tree.heading("Cantidad", text="Cantidad")

def agregar_producto():
    seleccion = tree.selection()
    producto = tree.item(seleccion)['text']
    cantidad = entry.get()
    if cantidad == "":
        tk.messagebox.showerror("Error", "Ingrese una
cantidad válida.")
    elif(cantidad.isnumeric()):
        tree.set(seleccion, "Cantidad", cantidad)
        diccionario_prod[producto]=cantidad
        #print(f"{producto}: {cantidad}")
        #print(diccionario_prod)

def enviar_servidor():
    if not diccionario_prod:
        tk.messagebox.showerror("Error", "No se han
seleccionado productos.")
    else:
        print(diccionario_prod,"<-----")
        client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        client_socket.connect(("localhost", 12345))
        aux=json.dumps(diccionario_prod).encode()
        #aux=str(diccionario_prod)
        #aux=aux.replace("{", "")
        #aux=aux.replace("}", "")
        #print(aux)
        client_socket.send(aux)
        client_socket.close()
        root.destroy()
productos = []

```

```

for i in productos_bd:
    print(i)
    productos.append(i[1])
for producto in productos:
    tree.insert("", "end", text=producto)

entry = tk.Entry(root)
entry.grid(row=1, column=1)
boton_agregar = tk.Button(root, text="Agregar",
command=agregar_producto)
boton_agregar.grid(row=1, column=2)
boton_agregar = tk.Button(root, text="Enviar",
command=enviar_servidor)
boton_agregar.grid(row=1, column=3)
tree.grid(row=0, column=0)
root.mainloop()

```

LADO SERVIDOR

```

import socket
import threading
import mysql.connector
import tkinter as tk
from tkinter import ttk
import json

def fac():
    # Crear ventana principal
    root = tk.Tk()
    root.title("Factura")
    # Crear etiquetas para mostrar la factura
    cliente_label = tk.Label(root, text="Cliente: localhost: 127.0.0.1")
    cliente_label.grid(row=0, column=0, padx=5, pady=5)
    fecha_label = tk.Label(root, text="Fecha: 08/05/2023")
    fecha_label.grid(row=1, column=0, padx=5, pady=5)
    descripcion_label = tk.Label(root, text="Descripción")
    descripcion_label.grid(row=2, column=0, padx=5, pady=5)
    cantidad_label = tk.Label(root, text="Cantidad")
    cantidad_label.grid(row=2, column=1, padx=5, pady=5)

    precio_label = tk.Label(root, text="Precio")
    precio_label.grid(row=2, column=2, padx=5, pady=5)

    # Conectarse a la base de datos
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="",

```



```

        database="supermercado"
    )
    cursor = conn.cursor()
    # Obtener los productos de la base de datos
    cursor.execute("SELECT * FROM productos")
    productos = cursor.fetchall()
    # Mostrar los productos en la tabla
    fila = 3 # empezamos en la fila 3 para dejar espacio para las etiquetas
    for producto in productos:
        nombre = producto[1]
        precio_unitario = producto[3]
        existencia = producto[2]
        # crear etiquetas para el producto
        nombre_label = tk.Label(root, text=nombre)
        nombre_label.grid(row=fila, column=0, padx=5, pady=5)
        cantidad_label = tk.Label(root, text=str(precio_unitario))
        cantidad_label.grid(row=fila, column=1, padx=5, pady=5)
        precio_label = tk.Label(root, text="$" + str(existencia))
        precio_label.grid(row=fila, column=2, padx=5, pady=5)
        fila += 1
    # Calcular y mostrar el total de la factura
    total = sum([producto[2]*producto[3] for producto in productos])
    total_label = tk.Label(root, text="Total: $" + str(total))
    total_label.grid(row=fila, column=2, padx=5, pady=5)
    # Cerrar la conexión a la base de datos
    cursor.close()
    conn.close()
    # Iniciar bucle de eventos
    root.mainloop()

def mostrar_factura(detalle,total,diccionario):
    # Conectarse a la base de datos
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="",
        database="supermercado"
    )
    cursor = conn.cursor()
    # Obtener los productos de la base de datos
    cursor.execute("SELECT fecha FROM factura WHERE detalle=%s AND
total=%s", (detalle,total))
    factura = cursor.fetchall()
    print(factura,"Aquí es la Factura" )
    root = tk.Tk()
    root.title("Factura")
    fecha="Fecha: 08/05/2023"
    # Crear etiquetas para mostrar la factura
    cliente_label = tk.Label(root, text="Cliente: localhost")
    cliente_label.grid(row=0, column=0, padx=5, pady=5)
    fecha_label = tk.Label(root, text=fecha)
    fecha_label.grid(row=1, column=0, padx=5, pady=5)
    descripcion_label = tk.Label(root, text="Nombre Producto")
    descripcion_label.grid(row=2, column=0, padx=5, pady=5)
    cantidad_label = tk.Label(root, text="Cantidad")
    cantidad_label.grid(row=2, column=1, padx=5, pady=5)

```

```

precio_label = tk.Label(root, text="Precio Unitario")
precio_label.grid(row=2, column=2, padx=5, pady=5)
precio_label = tk.Label(root, text="Total")
precio_label.grid(row=2, column=3, padx=5, pady=5)
cursor.close()
fila = 3 # empezamos en la fila 3 para dejar espacio para las etiquetas
for i in diccionario:
    nombre_label = tk.Label(root, text=i["nombre"])
    nombre_label.grid(row=fila, column=0, padx=5, pady=5)
    cantidad_label = tk.Label(root, text=str(i["cantidad"]))
    cantidad_label.grid(row=fila, column=1, padx=5, pady=5)
    precio_label = tk.Label(root, text="Bs" + str(i["precio_unitario"]))
    precio_label.grid(row=fila, column=2, padx=5, pady=5)
    precio_label = tk.Label(root, text="Bs" + str(i["total"]))
    precio_label.grid(row=fila, column=3, padx=5, pady=5)
    fila+=1

total_label = tk.Label(root, text="Total: Bs" +
str(total),bg="lightblue")
total_label.grid(row=fila, column=2, padx=5, pady=5)
conn.close()
# Iniciar bucle de eventos
root.mainloop()
def handle_client(client_socket):
    data = client_socket.recv(1024).decode()
    mi_dicc_resc=json.loads(data)
    print(mi_dicc_resc)
    print(type(mi_dicc_resc))
    my_connect = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="",
    database="supermercado"
    )
    my_conn = my_connect.cursor()
    detalle=""
    vec_tot=[]
    au=[]
    v=0
    vec_dicc=[]
    var_det=""
    for i in mi_dicc_resc:
        new_dicc={}
        my_conn.execute("SELECT nombre,(precio_unitario*%s) as
total,precio_unitario FROM productos WHERE nombre=%s",(mi_dicc_resc[i],i))
        au.append(my_conn.fetchall())
        vec_tot.append(au[v][0][1])
        new_dicc["nombre"]=i
        new_dicc["cantidad"]=mi_dicc_resc[i]
        new_dicc["precio_unitario"]=au[v][0][2]
        new_dicc["total"]=au[v][0][1]
        vec_dicc.append(new_dicc)
        detalle+=f"{i} \t\t\t\t {mi_dicc_resc[i]}\t\t\t\t\t
{au[v][0][2]}\t\t\t\t\t {au[v][0][1]} \n"
        v+=1

```

```

    var_total=sum(vec_tot)
    my_conn.execute("INSERT INTO factura(detalle,total) VALUES
(%s,%s)", (detalle,float(var_total)))
    my_conn.execute("SELECT * FROM factura")
    print(my_conn.fetchall())
    print("Vector Total ",vec_tot)
    print(vec_dicc)
    print("-----")
    print(detalle)
    my_connect.commit()
    my_conn.close()
    #fac()
    mostrar_factura(detalle,float(var_total),vec_dicc)
    client_socket.close()

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(("localhost", 12345))
server_socket.listen(5)
print("Servidor iniciado. Esperando conexiones...")

while True:
    client_socket, address = server_socket.accept()
    print(f"Conexión establecida con {address}")
    client_thread = threading.Thread(target=handle_client,
args=(client_socket,))
    client_thread.start()

```

Funcionamiento de la Aplicación de Escritorio:

Producto	Cantidad
Leche	
Maple de Huevo	
Papas Fritas	
Salchicha	
Vino	
Sal	
Arroz	
Mantequilla	
Fideo	
Queso	

LAB273, MINIPROYECTO 2

Producto	Cantidad
Leche	20
Maple de Huevo	
Papas Fritas	
Salchicha	10
Vino	
Sal	
Arroz	2
Mantequilla	
Fideo	15
Queso	

2

Factura

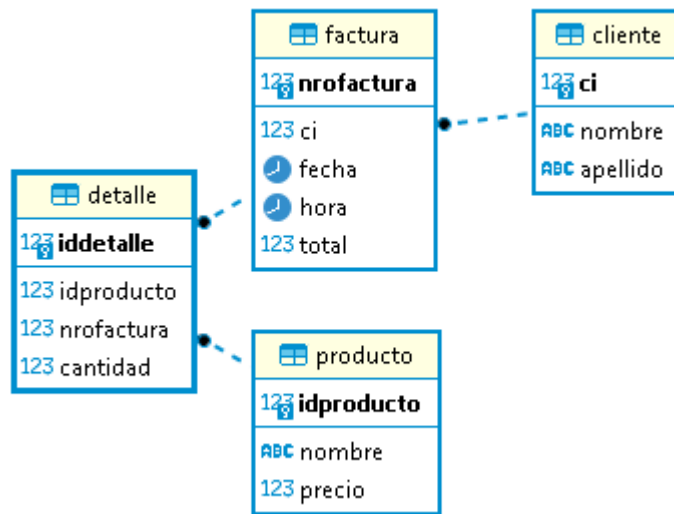
Cliente: localhost

Fecha: 08/05/2023

Nombre Producto	Cantidad	Precio Unitario	Total
Salchicha	10	Bs15.5	Bs155.0
Leche	20	Bs6.0	Bs120.0
Fideo	15	Bs11.2	Bs168.0
Arroz	2	Bs13.5	Bs27.0
Total: Bs470.0			

Desarrollo web

Se desarrolló el diseño de una base de datos relacional para facturación con los datos mas esenciales



Diseño de la base de datos:

Código Cliente: Desde el lado del cliente se configura la comunicación mediante sockets con el servidor, para ello hacemos referencia al host del como al puerto del servidor TCP, una vez establecida la conexión empleando sockets mediante nuestra interfaz grafica, desde ella enviamos mensajes al servidor que dependiendo el mensaje lo resuelve y nos da la información solicitada.

```
from flask import Flask, render_template, request, redirect, url_for
import requests
import socket

app = Flask(__name__)
TCP_SERVER_HOST = 'localhost'
TCP_SERVER_PORT = 5001

def send_message(message):
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.connect((TCP_SERVER_HOST, TCP_SERVER_PORT))
            s.sendall(message.encode())
            response = s.recv(1024).decode()
        return response
    except socket.error as e:
        # Manejar la excepción de conexión aquí
```

```

        print(f"Error de conexión: {e}")
        return "Error de conexión"

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/cliente', methods=['GET', 'POST'])
def cliente():
    if request.method == 'POST':
        ci = request.form['ci']
        nombre = request.form['nombre']
        apellido = request.form['apellido']

        message = f"INSERT cliente {ci} {nombre} {apellido}"
        response = send_message(message)

        return redirect(url_for('cliente'))

    message = "SELECT cliente"
    response = send_message(message)
    rows = response.split('\n')
    return render_template('cliente.html', clientes=rows)

@app.route('/producto', methods=['GET', 'POST'])
def producto():
    if request.method == 'POST':
        idproducto = request.form['idproducto']
        nombre = request.form['nombre']
        precio = request.form['precio']

        message = f"INSERT producto {idproducto} {nombre}
{precio}"
        response = send_message(message)

        return redirect(url_for('producto'))

    message = "SELECT producto"
    response = send_message(message)
    rows = response.split('\n')

    return render_template('producto.html', productos=rows)

```

```

@app.route('/factura', methods=['GET', 'POST'])
def factura():
    if request.method == 'POST':
        ci = request.form['ci']
        fecha = request.form['fecha']
        nrofactura = request.form['nrofactura']
        hora = request.form['hora']
        cantidad = request.form['total']
        prueba=request.form['cant']

        productos = request.form.getlist('producto')    # Obtener
una lista de los productos seleccionados

        message = f"INSERT factura {ci} {fecha} {hora}
{cantidad}"
        response = send_message(message)

        cont=0
        array=prueba.split(",")
        for producto in productos:
            if array[cont]!="0":
                idproducto, b1,cantidad1 = producto.split(',')
                # Generar el código de inserción en la base de
datos para cada producto
                message = f"INSERT detalle {idproducto}
{nrofactura} {array[cont]}"
                response = send_message(message)
                cont=cont+1
        return redirect(url_for('factura'))

message = "SELECT factura"
response = send_message(message)
rows = response.split('\n')

message = "SELECT cliente"
response = send_message(message)
rows1 = response.split('\n')

message = "SELECT producto"
response = send_message(message)
rows2 = response.split('\n')

message1 = "SELECT p1"
response1 = int(send_message(message1))+1

```

```

print(responsel)

message = "SELECT detalle"
response = send_message(message)
rows3 = response.split('\n')

        return render_template('factura.html', facturas=rows,
clientes=rows1, productos=rows2, detalles=rows3,nroa=responsel)

if __name__ == '__main__':
    app.run(host='0.0.0.0')

```

Codigo Servidor: El servidor esta configurado de la siguiente manera, el host de manera local, el puerto configurado para su funcionamiento es el 5001, la conexion a la base de datos en postgres esta realizada de manera eficiente. Dependiendo de las solicitudes que reciba el servidor este se comunicara con la base de datos para extraer o insertar información.

```

import socket
import psycopg2
import threading

HOST = 'localhost'
PORT = 5001
DB_NAME = 'supermercado'
DB_USER = 'postgres'
DB_PASSWORD = 'c4rp1nch0'

def handle_message(message):
    conn = psycopg2.connect(database=DB_NAME, user=DB_USER,
password=DB_PASSWORD, host=HOST, port=5432)
    cursor = conn.cursor()

    # Ejemplo de manejo de mensajes
    if message.startswith("INSERT cliente"):
        parts = message.split(" ")
        ci = int(parts[2])
        nombre = parts[3]
        apellido = parts[4]
        cursor.execute("INSERT INTO cliente (ci, nombre,
apellido) VALUES (%s, %s, %s)", (ci, nombre, apellido))
        conn.commit()
        response = "Cliente insertado correctamente"
    elif message.startswith("SELECT cliente"):

```



```

        cursor.execute("SELECT ci, nombre, apellido FROM
cliente")
        rows = cursor.fetchall()
        response = "\n".join([".".join(map(str, row)) for row in
rows])
        elif message.startswith("INSERT producto"):
            parts = message.split(" ")
            nombre = parts[3]
            precio = float(parts[4])
            cursor.execute("INSERT INTO producto (nombre, precio)
VALUES (%s, %s)", (nombre, precio))
            conn.commit()
            response = "Producto insertado correctamente"
        elif message.startswith("SELECT producto"):
            cursor.execute("SELECT idproducto, nombre, precio FROM
producto")
            rows = cursor.fetchall()
            response = "\n".join([".".join(map(str, row)) for row in
rows])
        elif message.startswith("INSERT factura"):
            parts = message.split(" ")
            ci = int(parts[2])
            fecha = parts[3]
            hora = parts[4]
            total = float(parts[5])
            cursor.execute("INSERT INTO factura (ci, fecha, hora,
total) VALUES (%s, %s, %s, %s)",
                           (ci, fecha, hora, total))
            conn.commit()
            response = "Factura insertada correctamente"
        elif message.startswith("SELECT factura"):
            cursor.execute("SELECT nrofactura, ci, fecha, hora, total
FROM factura")
            rows = cursor.fetchall()
            response = "\n".join([".".join(map(str, row)) for row in
rows])
        elif message.startswith("SELECT p1"):
            cursor.execute("SELECT nrofactura FROM factura ORDER BY
nrofactura DESC LIMIT 1;")
            row = cursor.fetchone()
            if row:
                response = str(row[0])
            else:

```

```

        response = "No se encontraron registros."
    elif message.startswith("INSERT detalle"):
        parts = message.split(" ")
        idproducto = int(parts[2])
        nrofactura = int(parts[3])
        cantidad = int(parts[4])
        cursor.execute("INSERT INTO detalle (idproducto,
nrofactura, cantidad) VALUES (%s, %s, %s)",
                        (idproducto, nrofactura, cantidad))
        conn.commit()
        response = "Detalle insertado correctamente"
    elif message.startswith("SELECT detalle"):
        cursor.execute("SELECT iddetalle, idproducto, nrofactura,
cantidad FROM detalle")
        rows = cursor.fetchall()
        response = "\n".join(["", ".join(map(str, row)) for row in
rows])
    else:
        response = "Comando no reconocido"

    cursor.close()
    conn.close()
    return response
def start_server():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
server_socket:
        server_socket.bind((HOST, PORT))
        server_socket.listen()

        print(f"Servidor TCP escuchando en el puerto {PORT}")

        while True:
            client_socket, client_address =
server_socket.accept()
            with client_socket:
                print(f"Conexión establecida desde
{client_address}")
                data = client_socket.recv(1024).decode()
                response = handle_message(data)
                client_socket.sendall(response.encode())

def run_server():
    server_thread = threading.Thread(target=start_server)

```

```

server_thread.start()

if __name__ == '__main__':
    run_server()

```

Capturas de Ejecución:

Ejecucion del servidor: El servidor con socket TCP se encuentra escuchando desde el puerto 5001

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```

Downloading protobuf-3.20.3-py2.py3-none-any.whl (162 kB)
100% |#####| 162.1/162.1
Installing collected packages: protobuf, mysql-connector-python
Successfully installed mysql-connector-python-8.0.33 protobuf-3.20.3
PS C:\Users\Kalaris\Documents\273\Mini_proyecto_2\ba1> python server.py
Servidor iniciado. Esperando conexiones...

```

* Historial restaurado

```

Servidor TCP escuchando en el puerto 5001
[]

```

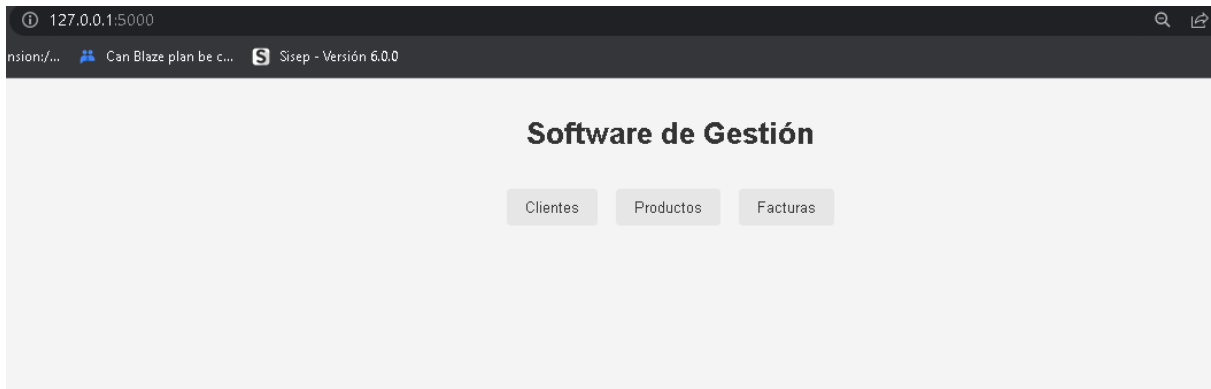
Ejecucion del cliente: Nuestro cliente esta funcionando desde la direccion local y mediante la direccion ip asignada a este por el puerto 5000 para el funcionamiento mediante web

```

* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.24.27.87:5000
Press CTRL+C to quit

```

Inicio de página: El inicio de nuestra cliente que funciona desde la web por la direccion local en el puerto 5000



Pestaña para registrar clientes: En esta sección realizamos el registro de los clientes y también podemos visualizar los registros ya existentes.

Clientes		
CI	Nombre	Apellido
10101010	Thimas	Kebec
123321	aaaa	nnnn
123737	ana	Huanca
1	A	B
12312	asasd	asda

Agregar Cliente

CI:

Nombre:

Apellido:

Pestaña para registrar productos: En esta página dedicada a productos realizamos el registro de los productos y también podemos visualizar los registros ya existentes

Productos		
ID Producto	Nombre	Precio
1	Uva	7.0
2	Manzana	10.0
3	arroz	2.0
4	Tartas	15.0
5	Frijoles	20.0
6	Uva	7.0

Agregar Producto

Nombre:

Precio:

Pestaña factura para registrar facturas: Aca tenemos un despliegue de las facturas ya registradas y el importe total en las mismas, se tiene tambien la lista de los clientes como el detalle de cada factura. Para registrar una nueva factura primero se debe seleccionar de la tabla clientes al cliente que deseamos registrar y en la tabla productos de la parte inferior colocamos en su respectiva celda de cantidad las unidades que deseamos. Una vez realizado esto podemos guardar la factura con el boton agregar y esta se registrara en la base de datos y los cambios apareceran en pantalla.

Facturacion

Facturas					Clientes			Detalles factura			
Nro Factura	CI Cliente	Fecha	Hora	Total	CI	Nombre	Apellido	ID Detalle	ID Producto	Nro Factura	Cantidad
1	10101010	2023-05-12	18:00:00	100.0	10101010	Thimas	Kebec	1	1	1	23
2	10101010	2023-05-12	18:00:00	0.0	123321	aaaa	nnnn	3	1	1	12
3	10101010	2023-05-12	18:00:00	0.0	123737	ana	Huanca	4	2	1	12
72	10101010	2023-05-08	00:29:34	495.0	1	A	B	5	3	1	12
5	10101010	2023-05-07	17:58:29	0.0	12312	asasd	asda	16	1	3	2
6	10101010	2023-05-07	18:18:19	0.0				17	2	3	3
73	10101010	2023-05-08	00:29:53	176.0				42	4	72	33
74	123321	2023-05-08	01:38:00	261.0				43	4	73	88
75	123321	2023-05-08	01:38:30	22220.0				44	2	74	3
76	123321	2023-05-08	18:15:45	10161.0				45	3	74	20
77	10101010	2023-05-08	18:19:00	331.0				46	4	74	10
71	10101010	2023-05-08	00:28:49	135.0				47	5	75	1111
								48	2	76	23
								49	3	76	1000
								50	3	77	23
								51	5	77	2

Agregar Factura

Nro Factura:

78

CI:

Fecha:

2023-5-8

Hora:

18:47:39

Total:

Productos				
ID Producto	Nombre	Precio	Cantidad	Total pagar
1	Uva	7.0	0	0.00
2	Manzana	10.0	0	0.00
3	arroz	2.0	0	0.00
4	Tartas	15.0	0	0.00
5	Frijoles	20.0	0	0.00
6	Uva	7.0	0	0.00

Agregar

- Diseño del esquema de la base de datos: Se diseño el esquema de la base de datos para almacenar las facturas. El esquema incluye las tablas necesarias para almacenar los datos de las facturas, como el número de factura, la fecha, el importe total y el extracto de los productos.
- Configuración del servidor de base de datos: Se debe configurar un servidor de base de datos relacional para almacenar las facturas. El servidor esta configurado para conexiones TCP.
- Implementación del servidor de sockets TCP: Se implemento un servidor de sockets TCP que escuche las conexiones entrantes y procese las solicitudes de los clientes. El servidor es capaz de recibir las facturas enviadas por los clientes y almacenarlas en la base de datos.
- Implementación del cliente de sockets TCP: Se implemento un cliente de sockets TCP que se conecte al servidor y envíe las facturas.
- Implementación de la interfaz web: Se implemento una interfaz web utilizando Flask para gestionar las facturas almacenadas en la base de datos. La interfaz permite la visualización de las facturas, así como los registros de productos y clientes..
- Integración del servidor de sockets y la interfaz web: Se integro el servidor de sockets TCP y la interfaz web para que la interfaz pueda mostrar las facturas enviadas al servidor y almacenadas en la base de datos.

ESTRUCTURA DE LA APLICACIÓN

Servidor de sockets TCP:

- El servidor está programado en un lenguaje de programación compatible con sockets TCP como lo es Python.
- El servidor debe aceptar conexiones entrantes y procesar las solicitudes de los clientes.
- El servidor debe ser capaz de recibir las facturas enviadas por los clientes y almacenarlas en la base de datos.

- El servidor debe poder enviar las facturas almacenadas en la base de datos a los clientes que lo soliciten.

Cliente de sockets TCP:

- El cliente esta programado en Python, que dicho lenguaje de programación es compatible con sockets TCP.
- El cliente puede leer las facturas de un archivo o recibir las facturas directamente del usuario y enviarlas al servidor.
- El cliente debe ser capaz de recibir las facturas almacenadas en la base de datos del servidor.

5. Conclusiones

Siendo que la utilización de sockets TCP para la comunicación entre el cliente y el servidor garantiza una transferencia eficiente y segura de datos. Los sockets TCP permiten una conexión estable y confiable entre los dispositivos, lo que es crucial para la generación y gestión de facturas en tiempo real. Además, la comunicación basada en sockets TCP facilita la escalabilidad de la aplicación, ya que permite manejar múltiples conexiones simultáneas y distribuir la carga de trabajo, puesto que gran parte de para el desarrollo del programa se lo realizó con python, se observó que mencionado programa implementa tecnologías para la experiencia de usuario en tres niveles relevantes; llegamos a ver la funcionalidad que presenta el uso de Flask y sus respectivos módulos para la experiencia de una aplicación o programa web, mostrando el eficiente uso y disposición de las utilidades a usar; el uso de Tkinter, que es un módulo de Python que proporciona una biblioteca para la creación de interfaces gráficas de usuario es decir el uso de una aplicación de escritorio y la implementación de un programa de facturación donde la interacción es por consola, siendo que esta nos muestra un uso óptimo entre la comunicación entre sockets e intercambio de data.

6. Bibliografía

- 6.1. Lutz, M. (2013). Learning Python: Powerful Object-Oriented Programming. O'Reilly Media.
- 6.2. Ramalho, L. (2015). Fluent Python. O'Reilly Media.
- 6.3. PostgreSQL Global Development Group. (2021). PostgreSQL Documentation. Recuperado de <https://www.postgresql.org/docs/>
- 6.4. Flask Pallets Team. (2021). Flask Documentation. Recuperado de <https://flask.palletsprojects.com/>
- 6.5. Grinberg, M. (2018). Flask Web Development with Python Tutorial. Recuperado de <https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>
- 6.6. Stevens, W. R., Fenner, B., & Rudoff, A. M. (2004). TCP/IP Illustrated, Volume 1: The Protocols. Addison-Wesley Professional.
- 6.7. Richard, S. (2015). Python Network Programming Cookbook. Packt Publishing.
- 6.8. Sockets Tutorial - Python Documentation. (2021). Recuperado de <https://docs.python.org/3/library/socket.html>
- 6.9. PostgreSQL vs. MySQL vs. SQL Server - Comparing SQL Databases. (2021). Recuperado de

<https://www.upgrad.com/blog/postgresql-vs-mysql-vs-sql-server-comparison-of-popular-relational-databases/>