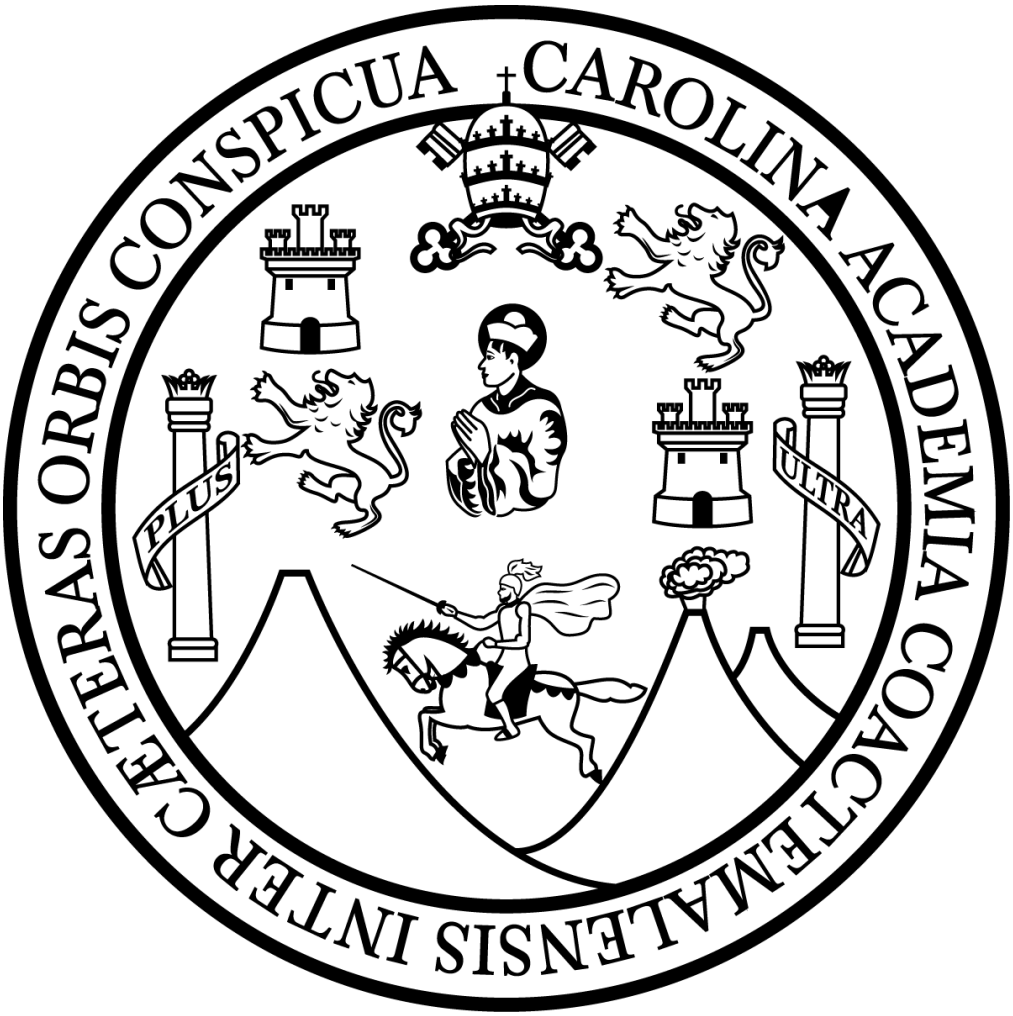
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA EN SISTEMA

INGENIERIA EN CIENCIAS Y SISTEMAS

INTRODUCCION A LA PROGRAMACION Y LA COMPUTACION II



TEMA:

PROYECTO No. 1

FECHA DE ENTREGA: 16/06/24

Introducción

En este proyecto, hemos desarrollado una aplicación de escritorio denominada IPCmarket. Esta aplicación actúa como una plataforma de comercio electrónico (appCommerce) diseñada para facilitar a los usuarios la compra de productos de consumo diario mediante una interfaz amigable e intuitiva. A continuación, se detalla el proceso de desarrollo, la documentación técnica y los diagramas que representan los principales algoritmos y estructuras de la aplicación.

La aplicación IPCmarket se desarrolló utilizando Python, una de las bibliotecas estándar para la creación de interfaces gráficas (tkinter), y SQLite para el almacenamiento de datos

1. Descripción General

La aplicación IPCmarket es una aplicación de escritorio que actúa como una plataforma de comercio electrónico (appCommerce). Esta aplicación permite a los usuarios visualizar una lista de productos disponibles y añadirlos a un carrito de compras. La interfaz de usuario es amigable e intuitiva, diseñada para facilitar el proceso de compra de productos de consumo diario.

2. Requisitos

Requisitos del Sistema

Python 3.x
Biblioteca `tkinter` (incluida por defecto en Python)
Biblioteca `sqlite3` (incluida por defecto en Python)

Instalación de Requisitos

No es necesario instalar bibliotecas adicionales ya que `tkinter` y `sqlite3` vienen incluidas con Python por defecto.

3. Estructura del Código

Creación de la Base de Datos

```python
from node import Node

class SimpleList():
    def __init__(self):
        self.head = None
        self.size = 0

    def __len__(self):
        return self.size

    def insert(self, data):
        newNode = Node(data)
        current = self.head
        if self.head == None:
            self.head = newNode
        else:
            current = self.head
            while current != None:
                if current.next == None:
                    current.next = newNode
                    break
                current = current.next
        self.size += 1
```

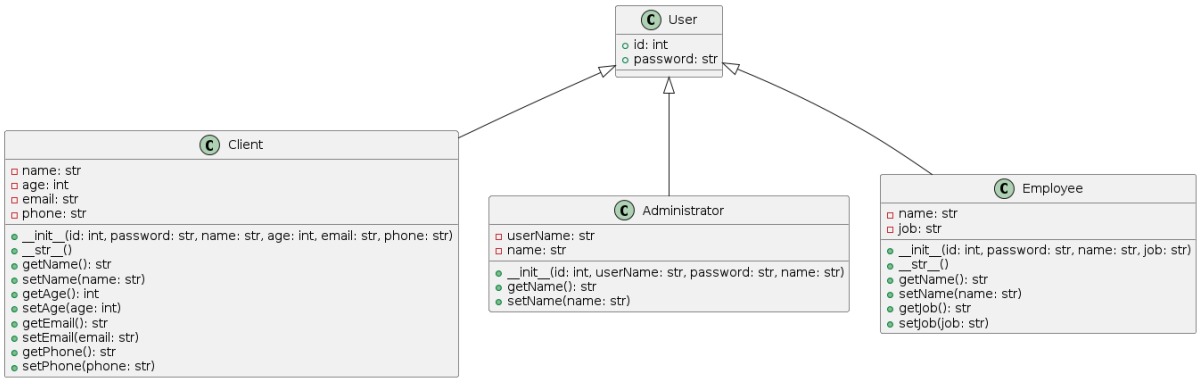


Diagrama de Actividades para "Clientes"

```python
class Client(User):
    def __init__ (self, id, password, name, age, email, phone):
        super().__init__(id, password)
        self.name = name
        self.age = age
        self.email = email
        self.phone = phone

    def __str__(self):
        return f'''
        ID: {self.id}
        Contraseña: {self.password}
        Nombre: {self.name}
        Edad: {self.age}
        Correo: {self.email}
        Telefono: {self.phone}
        '''

    #Getters and Setters
    def getName(self):
        return self.name
    def setName(self, name):
        self.name = name

    def getAge(self):
        return self.age
    def setAge(self, age):
        self.age = age

    def getEmail(self):
        return self.email
    def setEmail(self, email):
        self.email = email

    def getPhone(self):
        return self.phone
    def setPhone(self, phone):
        self.phone = phone
```

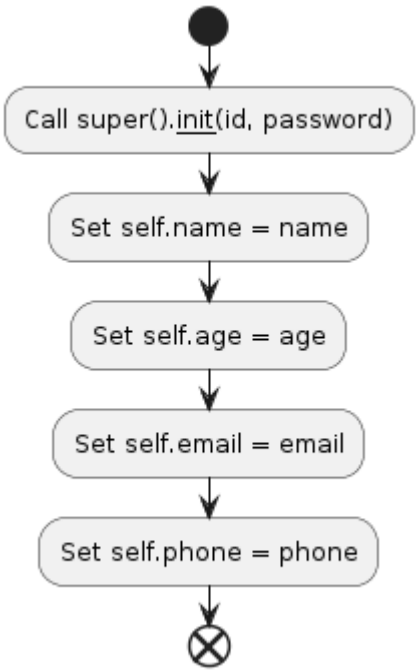


Diagrama de Actividades para "Administrador"

```python
from user import User
class Administrator(User):
    def __init__ (self, id, userName, password,
name):
        super().__init__(id, userName, password)
        self.name = name

    #Getters and Setters
    def getName(self):
        return self.name
    def setName(self, name):
        self.name = name
```

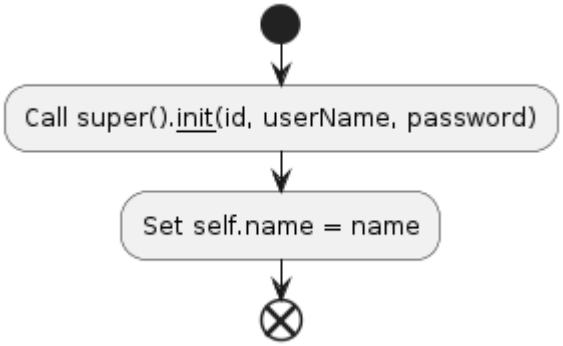


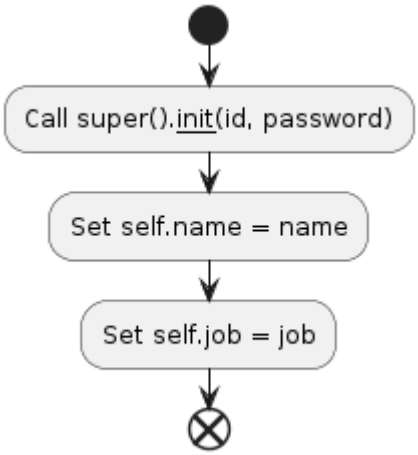
Diagrama de Actividades para "empleado"

```python
from user import User

class Employee(User):
    def __init__ (self, id, password, name, job):
        super().__init__(id, password)
        self.name = name
        self.job = job

    def __str__(self):
        return f'''
        ID: {self.id}
        Contraseña: {self.password}
        Nombre: {self.name}
        Puesto: {self.job}
        '''

    #Getters and Setters
    def getName(self):
        return self.name
    def setName(self, name):
        self.name = name
    def getJob(self):
        return self.job
    def setJob(self, job):
        self.job = job
```



4. **Ejecución de la Aplicación**

Crear la Base de Datos

Ejecuta el script de creación de la base de datos:
python create_database.py

Iniciar la Aplicación

Ejecuta el script principal de la aplicación:
python ipcmarket_app.py

5. **Funcionalidades**

Visualización de Productos

Los productos disponibles se muestran en una lista a la izquierda de la interfaz.

Añadir Productos al Carrito

Selecciona un producto de la lista y haz clic en "Añadir al carrito" para agregar el producto al carrito de compras.

Visualización del Carrito

Los productos añadidos al carrito se muestran en una lista a la derecha de la interfaz.

6. **Ampliaciones Futuras**

Este proyecto puede ser ampliado con las siguientes funcionalidades:
Eliminación de productos del carrito.
Procesamiento de pagos.
Gestión de usuarios y autenticación.
Integración con una base de datos remota.
Funcionalidades de búsqueda y filtrado de productos.

Explicación del Diagrama de Clases

Clase `Product`:
Atributos:
`id`: Identificador único del producto.
`name`: Nombre del producto.
`price`: Precio del producto.
`description`: Descripción del producto.
Métodos:
`__init__()`: Inicializa un objeto de la clase `Product`.
Clase `Database`:
Atributos:
`connection`: Conexión a la base de datos SQLite.
`cursor`: Cursor para ejecutar comandos SQL.
Métodos:
`create_database()`: Crea la base de datos y la tabla `products` si no existen.
`get_products()`: Recupera una lista de productos de la base de datos.
Clase `IPCmarketApp`:
Atributos:
`root`: Ventana principal de la aplicación Tkinter.
`product_listbox`: Listbox para mostrar los productos disponibles.
`add_button`: Botón para añadir productos al carrito.
`cart_listbox`: Listbox para mostrar los productos añadidos al carrito.
Métodos:
`__init__(root: tk.Tk)`: Inicializa la aplicación y configura la ventana principal.
`create_widgets()`: Crea y posiciona los widgets en la interfaz de usuario.
`load_products()`: Carga los productos desde la base de datos y los muestra en la lista de productos.
`add_to_cart()`: Añade el producto seleccionado al carrito.

Relación entre Clases

`IPCmarketApp` utiliza la clase `Database` para interactuar con la base de datos y obtener los productos.
`IPCmarketApp` muestra los productos utilizando la clase `Product` como modelo de datos.
Diagrama de Actividades para "Cargar Productos"
Este diagrama muestra el flujo del proceso de carga de productos desde la base de datos y su visualización en la interfaz de usuario.

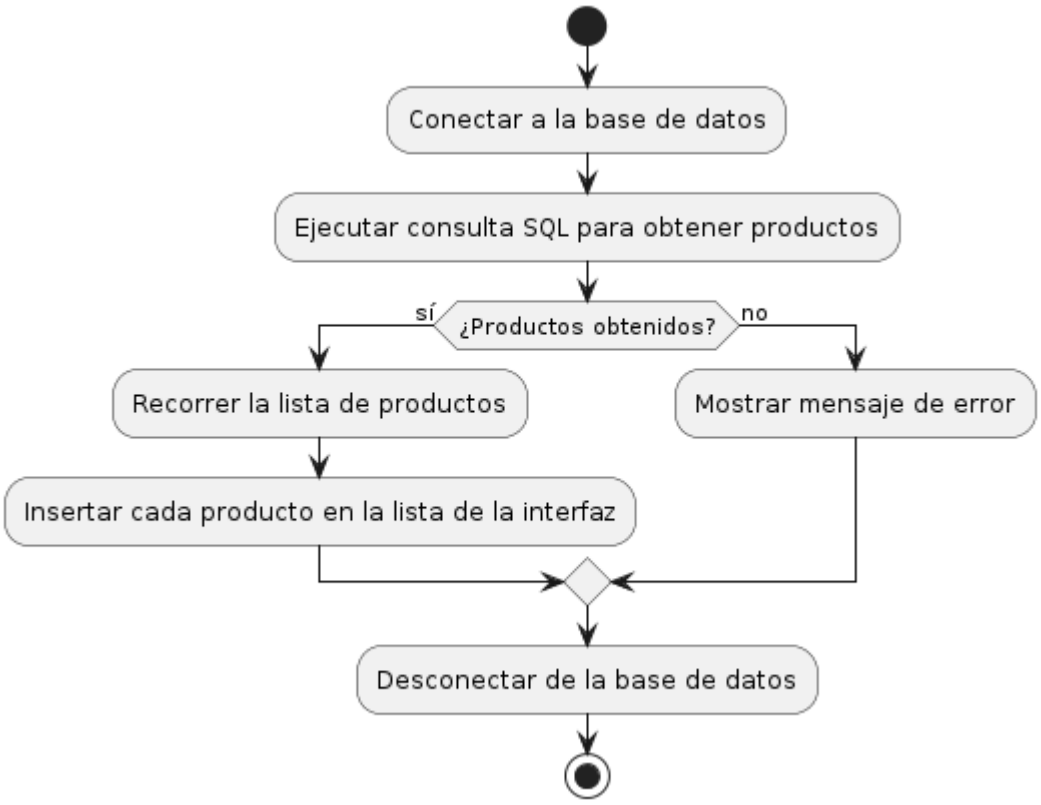
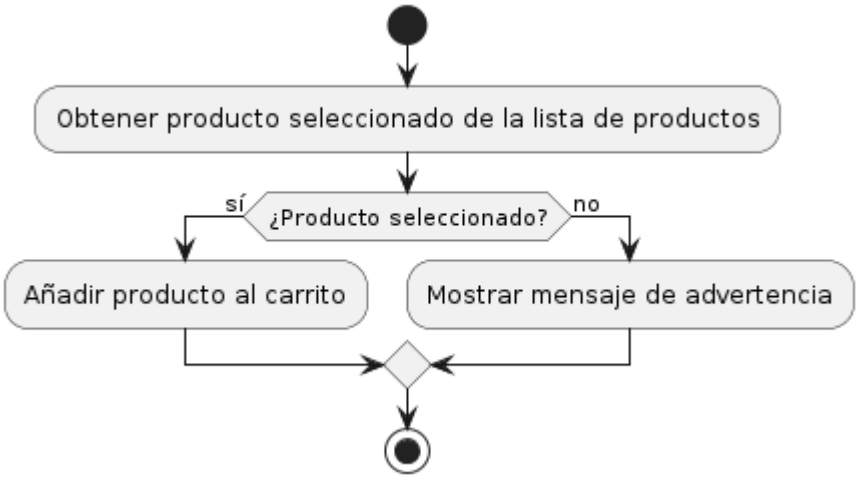


Diagrama de Actividades para "Añadir Producto al Carrito"

Este diagrama muestra el flujo del proceso para añadir un producto seleccionado al carrito de compras.



Explicación de los Diagramas

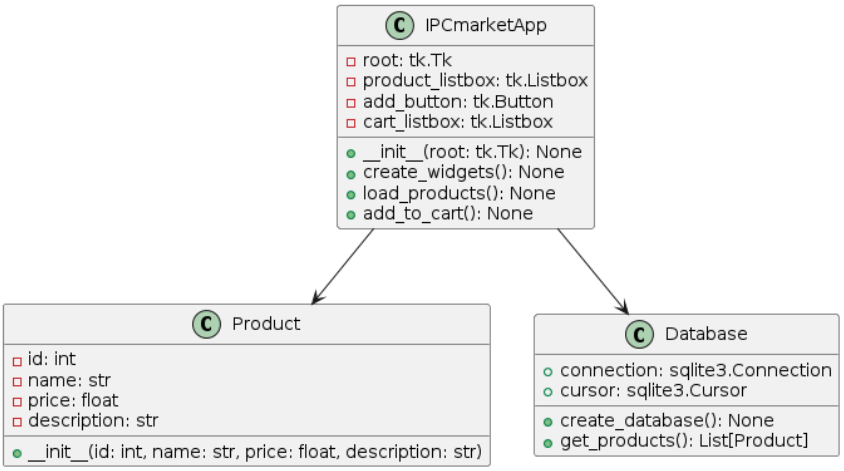
Diagrama de Actividades para "Cargar Productos"

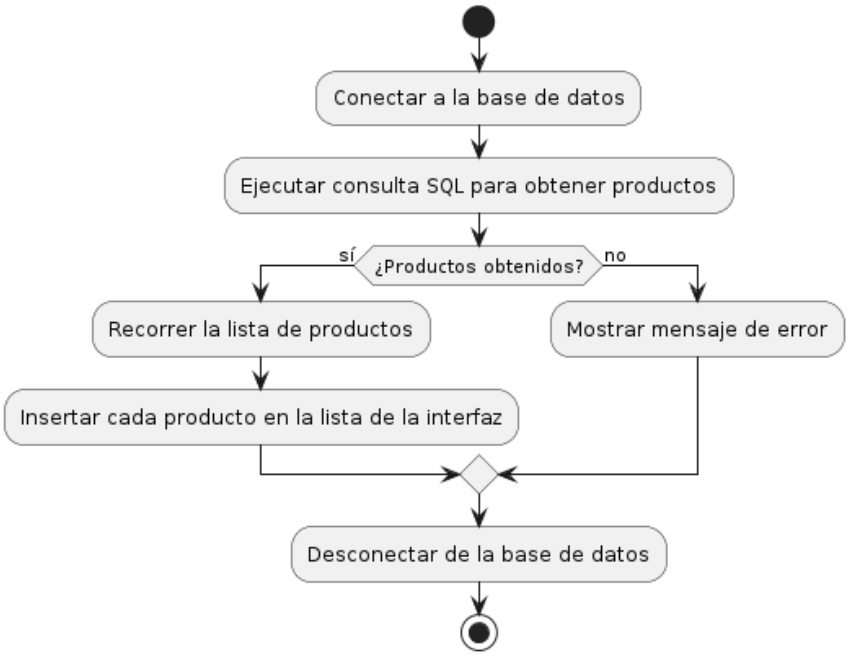
Conectar a la base de datos: Se establece una conexión con la base de datos SQLite.
Ejecutar consulta SQL para obtener productos: Se ejecuta una consulta SQL para recuperar los productos de la tabla `products`.
¿Productos obtenidos?: Se verifica si la consulta ha devuelto productos.
Sí: Se recorre la lista de productos y se insertan en la lista de la interfaz de usuario.
No: Se muestra un mensaje de error.
Desconectar de la base de datos: Se cierra la conexión con la base de datos.

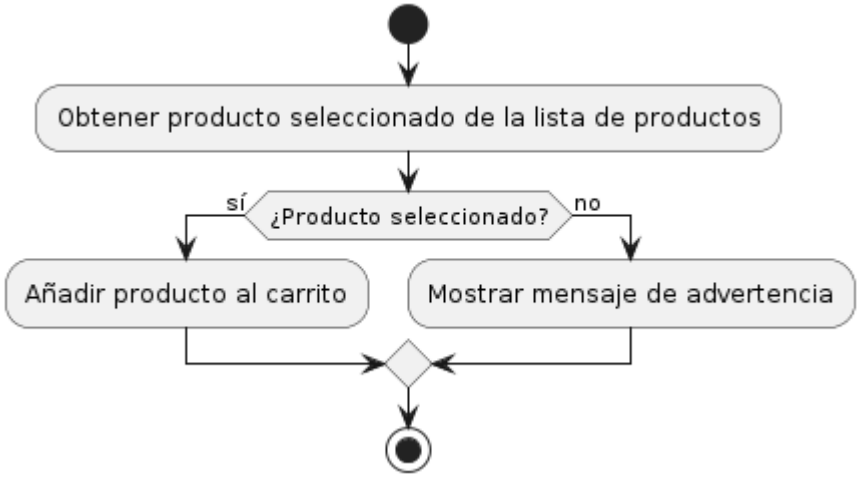
Diagrama de Actividades para "Añadir Producto al Carrito"

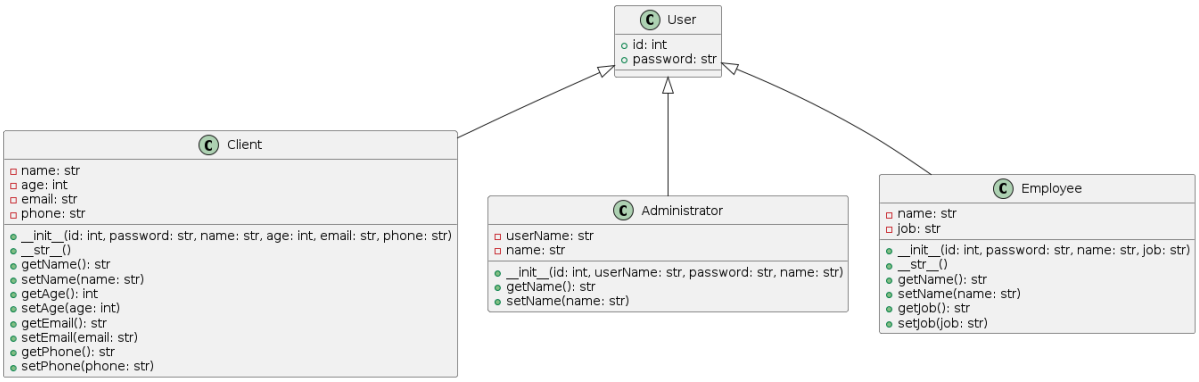
Obtener producto seleccionado de la lista de productos: Se obtiene el producto que el usuario ha seleccionado en la lista de productos.
¿Producto seleccionado?: Se verifica si hay un producto seleccionado.
Sí: Se añade el producto al carrito.
No: Se muestra un mensaje de advertencia indicando que no se ha seleccionado ningún producto.

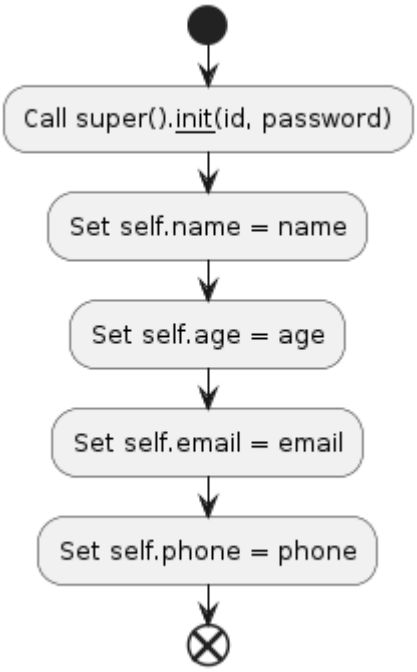
Anexos

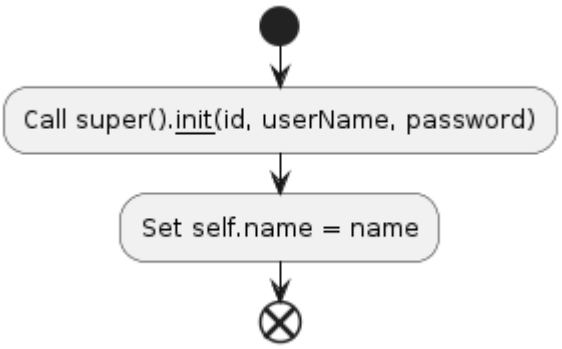

IPCmarketApp
root: tk.Tk
product_listbox: tk.Listbox
add_button: tk.Button
cart_listbox: tk.Listbox
__init__(root: tk.Tk): None
create_widgets(): None
load_products(): None
add_to_cart(): None
Product
id: int
name: str
price: float
description: str
__init__(id: int, name: str, price: float, description: str)
Database
connection: sqlite3.Connection
cursor: sqlite3.Cursor
create_database(): None
get_products(): List[Product]

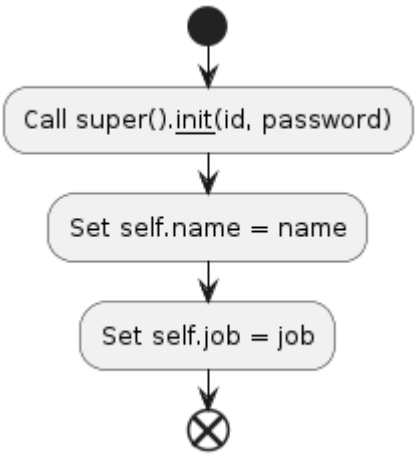

Conectar a la base de datos
Ejecutar consulta SQL para obtener productos
¿Productos obtenidos?
sí
no
Recorrer la lista de productos
Mostrar mensaje de error
Insertar cada producto en la lista de la interfaz
Desconectar de la base de datos


Obtener producto seleccionado de la lista de productos
¿Producto seleccionado?
sí
no
Añadir producto al carrito
Mostrar mensaje de advertencia


User
id: int
password: str
Client
name: str
age: int
email: str
phone: str
__init__(id: int, password: str, name: str, age: int, email: str, phone: str)
__str__()
getName(): str
setName(name: str)
getAge(): int
setAge(age: int)
getEmail(): str
setEmail(email: str)
getPhone(): str
setPhone(phone: str)
Administrator
userName: str
name: str
__init__(id: int, userName: str, password: str, name: str)
getName(): str
setName(name: str)
Employee
name: str
job: str
__init__(id: int, password: str, name: str, job: str)
__str__()
getName(): str
setName(name: str)
getJob(): str
setJob(job: str)


Call super().init(id, password)
Set self.name = name
Set self.age = age
Set self.email = email
Set self.phone = phone


Call super().init(id, userName, password)
Set self.name = name


Call super().init(id, password)
Set self.name = name
Set self.job = job

Conclusión

La aplicación IPCmarket es una solución básica pero funcional para el comercio electrónico, diseñada para ser amigable y fácil de usar. Con su implementación, hemos abordado aspectos clave como la creación de bases de datos, el desarrollo de interfaces gráficas y la implementación de lógica de negocio. Los diagramas de clases y de actividades proporcionan una representación clara de la estructura y el flujo de la aplicación, facilitando su comprensión y mantenimiento. Este proyecto puede servir como base para desarrollar soluciones de comercio electrónico más complejas en el futuro.