M6_p2_Sebastian Cabrera

Mejoras de entrega

1.Cambiar y mejorar la interfaz gráfica

Implemente cambios de colores de fondo para generar mayor contraste El color de fondo de la ventana y los widgets varía entre "SteelBlue4" y "pale turquoise".

```
self.etiqueta_categoria = Label(frame,
                                text="Categoria: ",
                                font=('Calibri', 11),
                                padx=15,
                                pady=10,
                                background="SteelBlue4")
self.etiqueta_categoria.grid(row=5, column=0)

self.nombre = Entry(frame,
                    border=4,
                    background="pale turquoise")
self.nombre.focus()  # foco del raton
```

A cada botón le sumé una imagen al texto que hace referencia a su funcionalidad. además de buscar colores a los que se asocia esa función
Para las imagenes se creo un directorio recursos donde se alojaron

```
# IMAGENES
imagen_buscar = self.formato_imagen("recursos/buscar.png")
self.imagen_salir = self.formato_imagen("recursos/salir.png")
imagen_eliminar = self.formato_imagen("recursos/eliminar.ico")
imagen_editar = self.formato_imagen("recursos/editar-4.png")
imagen_guardar = self.formato_imagen("recursos/guardar-2.png")
self.imagen_excel = self.formato_imagen("recursos/excel.png")
```

Utilice el método pack() para los widget principales. Si bien es más “rígido” que el método grid(), es responsivo y mantiene las alineaciones si se modifican las dimensiones de la ventana.

```
# VENTANA PRINCIPAL
frame = Frame(self.ventana, background="SteelBlue4", bd=2)
frame.pack()
```

2. Añadir un campo Categoría Stock para el producto

Sume los campo Categoria y stock para los productos. A los que se les dio limitaciones. deben ser de tipo flotante con un maximo de 2 decimales

```
            precio = round(float(precio.get()), 2)
            stock = round(float(stock.get()), 2)
```

```
__tablename__ = 'producto'
__table_args__ = {'sqlite_autoincrement': True}
id = Column(Integer, primary_key=True)
nombre = Column(String(100), nullable=False)
precio = Column(Integer, nullable=False)
stock = Column(Integer)
categoria = Column(String(100))

def __init__(self, nombre, precio, stock=0, categoria=None):
    self.nombre = nombre
    self.precio = precio
    self.stock = stock
    self.categoria = categoria
    print("Producto creado con éxito")
```

3. Implementar SQLAlchemy sobre SQLite para poder cambiar de base de datos sin necesidad de tocar el código.

Cree un archivo db.py para incorporar el ORM e importe la biblioteca.
Esta creado el motor para conectarse con una base de datos sqlite3 que es plausible de ser modificado.

app.py models.py db.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base

engine = create_engine("sqlite:///database/productos.db",
                       connect_args={
                           "check_same_thread": False
                       })

Session = sessionmaker(engine)
Base = declarative_base()
session = Session()
```

Creada esta sesión las utilice en los métodos para que trabaje con los objetos “producto”.

```
def get_productos(self):
    """def get_productos(self):
        Método de la clase VentanaProducto.
        Solicita a la tabla producto de productos.bd todos los registros
        y los renderiza en: self.tabla = ttk.Treeview. """
    # Borra lo existete actualmente en la tabla.
    registro_tabla = self.tabla.get_children()
    for fila in registro_tabla:
        self.tabla.delete(fila)
    # Consulta a base de datos
    registros = db.session.query(Producto).order_by(desc("nombre"))
    # print(registros) devuelve un objeto
    # insert de registros en tabla
    for registro in registros:
        print("registros:", registro)
        reg = self.tabla.insert( parent: "", index: 0,
                                values=(registro.nombre, registro.precio, registro
                                iid=registro.id)
```

4.Agregar algún nuevo widget gráfico proporcionado por Tkinter, que no sea una label, un botón, un cajón de texto o una tabla.

A) **Scrollbar**: Una barra de desplazamiento junto a la tabla producto

```
barra_scroll = Scrollbar(frame_tabla_buscar)
barra_scroll.grid(row=0, column=1, sticky='ns')
self.tabla_buscar = ttk.Treeview(frame_tabla_buscar,
                                 height=10,
                                 columns=columnas_a_mostrar,
                                 show='headings',
                                 style="mystyle.Treeview",
                                 yscrollcommand=barra_scroll.set)
self.tabla_buscar.grid(row=0, column=0)
```

B) **Checkbutton**: Un boon tipo check cuya función es reordenar los elementos de la tabla a partir de su atributo precio

```
    self.boton_check = Checkbutton(frame, text="Ordenar por precio",
                                   variable=self.opcion_precio,
                                   command=self.listar_por_precio,
                                   background="light grey")
    self.boton_check.deselect()
    self.boton_check.grid(row=8, sticky=W + E, padx=5, pady=5)
    # TABLA
```

C) **Menu**: Barra Menu que incluye las opciones de exportar la tabla a un archivo excel y un botón para salir de la aplicación

```
MENU
self.menu = Menu(self.ventana)
self.ventana.config(menu=self.menu)
self.boton_menu = Menu(self.menu)
self.menu.add_cascade(label="Menu", menu=self.boton_menu)
self.boton_menu.add_command(label="Exportar a Excel", command=self.crear_excel, image=self.imagen_excel,
                            compound='left')
self.boton_menu.add_command(label="Salir", command=self.salir, image=self.imagen_salir, compound='left')
```

5. Otras mejoras

-Implemente **notas** en todas las funciones y clases para facilitar la documentación y mostrar los propósitos de cada una de ellas.

-**manejo de errores** que puedan surgir durante la ejecución lo hice a través de implementar try/except, mostrando mensajes de error al usuario.

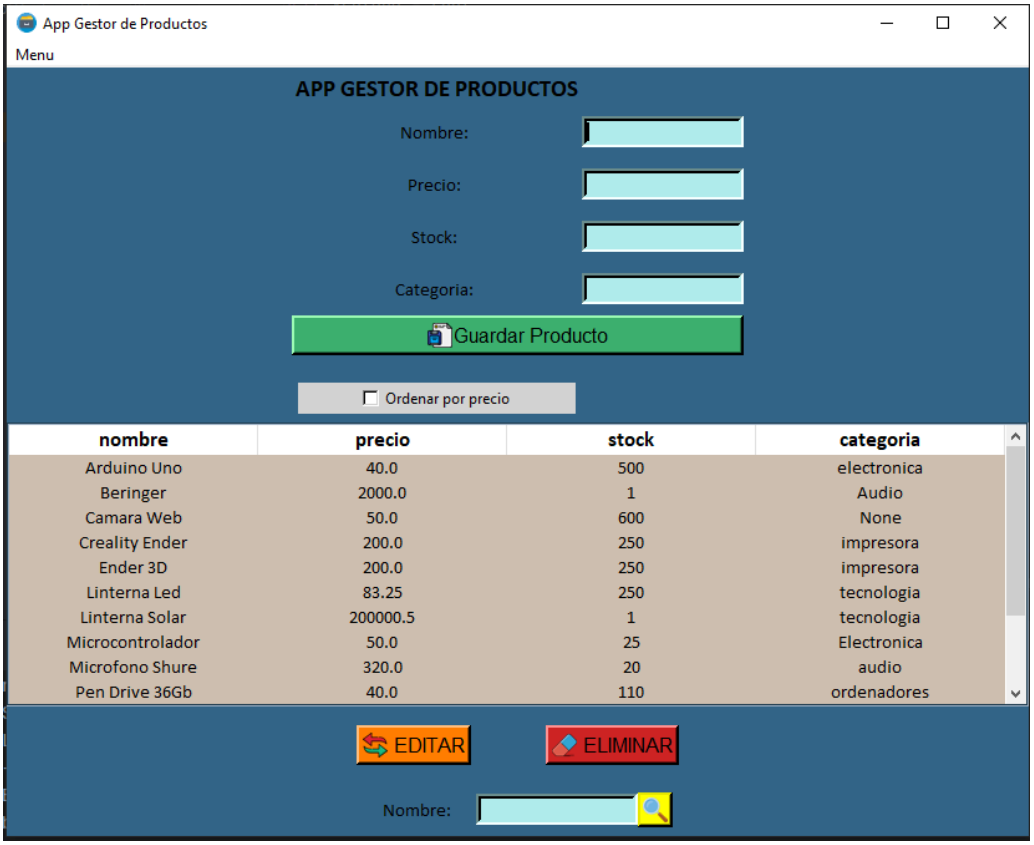
- **rollback**() de sqlalchemy para revertir una transacción que esta sujeta a un posible error.

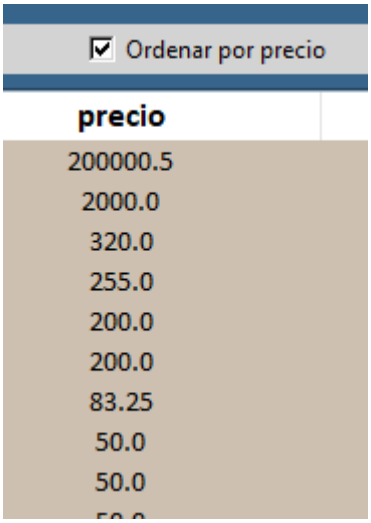
-**Caja de mensajes**, que crean ventanas emergentes para volver más dinámica y amigable con el usuario la aplicación

-**buscar_producto**, método que permite filtrar por nombre de producto creando una tabla nueva, emergente de la principal.

-Exportar a **Excel**, un método que crea un archivo excel al que se envia la tabla productos.

-**Validaciones**, en el codigo para asegurar la creación de cada objeto producto conforme a cierto estándar.





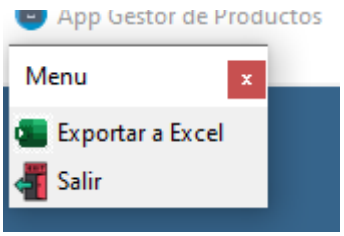
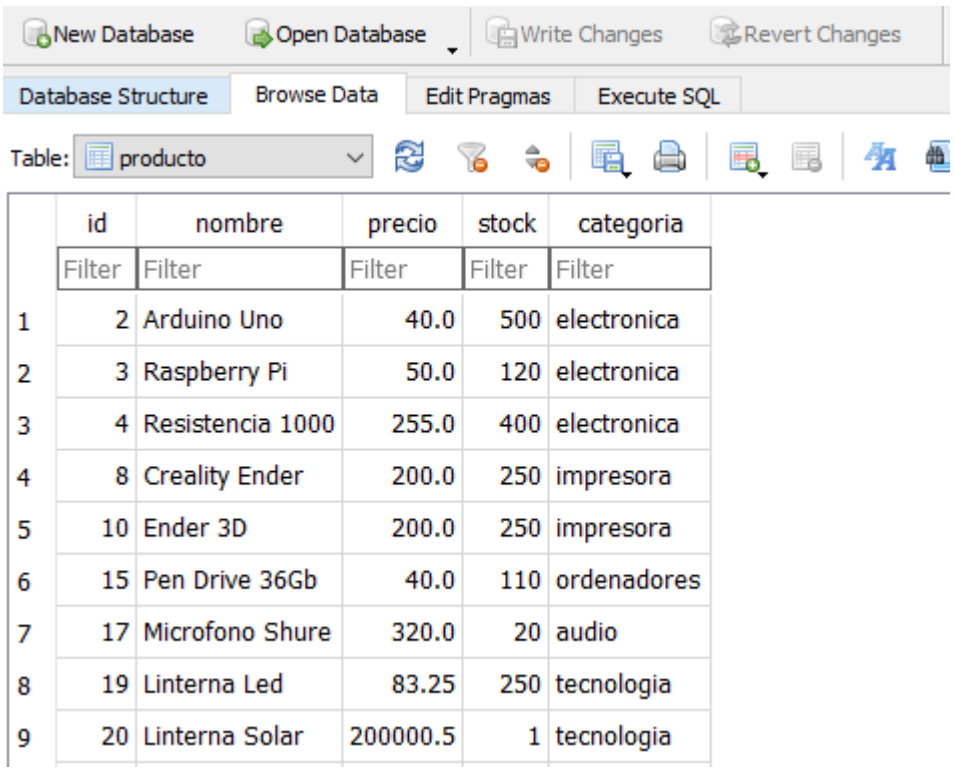


Tabla en DB Browser

producto		CREATE TABLE "producto" ("id" INTEGER NOT NULL, "nombre" TEXT NOT NULL, "precio" REAL NOT NULL, "stock" INTEGER, "categoria'
id	INTEGER	"id" INTEGER NOT NULL
nombre	TEXT	"nombre" TEXT NOT NULL
precio	REAL	"precio" REAL NOT NULL
stock	INTEGER	"stock" INTEGER
categoria	TEXT	"categoria" TEXT
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)



	id	nombre	precio	stock	categoria
	Filter	Filter	Filter	Filter	Filter
1	2	Arduino Uno	40.0	500	electronica
2	3	Raspberry Pi	50.0	120	electronica
3	4	Resistencia 1000	255.0	400	electronica
4	8	Creality Ender	200.0	250	impresora
5	10	Ender 3D	200.0	250	impresora
6	15	Pen Drive 36Gb	40.0	110	ordenadores
7	17	Microfono Shure	320.0	20	audio
8	19	Linterna Led	83.25	250	tecnologia
9	20	Linterna Solar	200000.5	1	tecnologia