Datos a Saber:

Donde este una clase o un código y se encuentre esta forma.....

Quiere decir que se esta llamando al modulo correspondiente con el método antes mencionado resaltado en fondo amarillo

ConfigManafer:

Url config/configmanager.py

Desglose de la Clase ConfigManager

1. Importación de Módulos

import configparser
Se importa el módulo configparser, que permite leer y escribir archivos de configuración en formato .ini.

2. Definición de la Clase

class ConfigManager:
Se define la clase ConfigManager, que será responsable de la gestión de la configuración.

3. Método Constructor __init__

```
def __init__(self, config_file='config.ini'):
    self.config = configparser.ConfigParser()
    self.config.read(config_file)
```

Parámetro: config_file (por defecto 'config.ini').

Se crea una instancia de ConfigParser y se lee el archivo de configuración especificado.
Esto permite que la clase acceda a las configuraciones definidas en el archivo .ini.

4. Método get_database_config

```
def get_database_config(self):
    try:
        return {
            'host': self.config.get('database', 'host'),
            'user': self.config.get('database', 'user'),
            'password': self.config.get('database', 'password'),
            'database': self.config.get('database', 'database'),
            # 'use_pure': self.config.getboolean('database', 'use_pure')
        }
    except configparser.NoSectionError:
        print("Error: La sección 'database' no existe en el archivo de configuración.")
        return None
    except configparser.NoOptionError as e:
        print(f"Error: Falta la opción {e} en la sección 'database'.")
        return None
```

Este método intenta recuperar la configuración de la base de datos de la sección database.
Retorna: Un diccionario con las claves host, user, password y database.

Manejo de Errores:
Si la sección database no existe, se captura la excepción NoSectionError y se imprime un mensaje de error.
Si falta alguna opción dentro de la sección database, se captura la excepción NoOptionError y se imprime el nombre de la opción faltante.

5. Método set_database

```
def set_database(self, new_database):
    if not self.config.has_section('database'):
        self.config.add_section('database')

    self.config.set('database', 'database', new_database)
    with open('config.ini', 'w', encoding='utf-8') as configfile:
        self.config.write(configfile)
        print(f"Base de Datos cambiada a {new_database}")
```

Este método permite establecer un nuevo valor para la base de datos.

Comprobación: Si la sección database no existe, se crea una nueva sección.
Configuración: Se establece el nuevo nombre de la base de datos utilizando self.config.set.

Escritura en Archivo: Se abre el archivo config.ini en modo escritura y se guardan los cambios realizados en la configuración.

Se imprime un mensaje confirmando el cambio.

Resumen
La clase ConfigManager es una herramienta útil para gestionar configuraciones de bases de datos de manera sencilla. Permite leer y escribir configuraciones desde un archivo .ini, manejando errores comunes que pueden surgir al acceder a configuraciones faltantes o incorrectas. Esto proporciona una forma organizada y accesible de manejar configuraciones en aplicaciones Python.

```python
import configparser

class ConfigManager:
  def __init__(self, config_file='config.ini'):
    self.config = configparser.ConfigParser()
    self.config.read(config_file)

  def get_database_config(self):
    try:
      return {
        'host': self.config.get('database', 'host'),
        'user': self.config.get('database', 'user'),
        'password': self.config.get('database', 'password'),
        'database': self.config.get('database', 'database'),
        # 'use_pure': self.config.getboolean('database', 'use_pure')
      }
    except configparser.NoSectionError:
      print(
        "Error: La sección 'database' no existe en el archivo de configuración.")
      return None
    except configparser.NoOptionError as e:
      print(f"Error: Falta la opción {e} en la sección 'database'.")
      return None

  def set_database(self, new_database):
    if not self.config.has_section('database'):
      self.config.add_section('database')

    self.config.set('database', 'database', new_database)
    with open('config.ini', 'w', encoding='utf-8') as configfile:
      self.config.write(configfile)
      print(f"Base de Datos cambiada a {new_database}")
```

Modulos:

Botones: btn_config

Url: modulos/botones/btn_config

1. Importación de Módulos

```
import tkinter as tk
import modulos.botones.btn_hover as btn_hover
import config.config as colores
from tkinter import font
import modulos.ejecucion_click.click_btn_insertData_BD as ejecutar
```

tkinter: Se importa para crear la interfaz gráfica.
btn_hover Módulos personalizados que manejan eventos de hover
colores: Un módulo que contiene definiciones de colores para la interfaz.
font: Se importa para manejar fuentes en los botones.
ejecutar: Módulo que contiene funciones que se ejecutarán al hacer clic en los botones.

2. Función configuracion_btn_menu_lateral

```
def configuracion_btn_menu_lateral(self, boton, text, icono, activo):
    """Configura los parámetros del botón en el menú lateral."""
    boton.config(
        text=f" {icono}  {text}",
        anchor="w",
        font=font.Font(family="FontAwesome", size=15),
        width=20,
        height=1,
        pady=5,
        bg=colores.COLOR_BTN,
        fg="white",
        bd=1,
        relief="raised",
        highlightbackground=colores.COLOR_MENU_LATERAL,
        highlightcolor=colores.COLOR_MENU_LATERAL,
        highlightthickness=2
    )
    boton.pack(side=tk.TOP, pady=8)
    if not activo:
        btn_hover.hover_event(self, boton)
    else:
        self.boton_activo = boton
```

Propósito: Configura un botón que se ubicará en el menú lateral de la GUI.

Parámetros:
self: Referencia a la instancia de la clase.
boton: El botón que se va a configurar.
text: El texto que se mostrará en el botón.
icono: Un icono que se mostrará junto al texto.
activo: Un booleano que indica si el botón está activo.
Configuración del Botón: Se establece el texto, fuente, colores, dimensiones y otros estilos.
Empaquetado: Se coloca el botón en la parte superior del contenedor.
Eventos de Hover: Si el botón no está activo, se aplica un evento de hover.

3. Función configuracion_btn_menu_superior

```
def configuracion_btn_menu_superior(self, boton, text, icono, activo):
    """Configura los parámetros del botón en el menú superior."""
    boton.config(
        text=f" {icono}  {text}",
        anchor="w",
        font=font.Font(family="FontAwesome", size=12),
        width=15,
        height=1,
        pady=5,
        bg=colores.COLOR_BTN,
        fg="white",
        bd=1,
        relief="raised",
        highlightbackground=colores.COLOR_BARRA_SUPERIOR,
        highlightcolor=colores.COLOR_BARRA_SUPERIOR,
        highlightthickness=2
    )
    boton.pack(side=tk.RIGHT, padx=15)
    btn_hover.hover_event_sup(self, boton)
```

Propósito: Similar a la función anterior, pero para botones en el menú superior.
Configuración: Se ajustan las propiedades del botón de manera similar, pero con diferentes dimensiones y posición (a la derecha).
Eventos de Hover: Se aplica un evento de hover específico para el menú superior.

4. Función crear_boton_sub_panel

```
def crear_boton_sub_panel(self, tipo_boton):
    """Crea un botón en el subpanel según el tipo especificado."""
    texto = {
        "Actualizar": "Actualizar",
        "Insertar": "Insertar"
    }.get(tipo_boton)

    if not texto:
        return  # Si el tipo de botón no es válido, salir del método

    # Crear el botón
    boton = tk.Button(
        self.panel_acciones_cuerpo,
        text=texto,
        padx=20,
        bg=colores.COLOR_BTN,
        font=("Arial", 12, "bold"),
        fg="white",
        command=lambda: ejecutar.acciones_botones_sub_panel(
            self, self.titulo_panel_administracion, boton)
    )

    # Empaquetar el botón
    boton.pack(side="right", padx=20)
    btn_hover.hover_event(self, boton)
```

Propósito: Crea un botón en un subpanel basado en el tipo de botón especificado.
Parámetro: tipo_boton: Determina qué tipo de botón se va a crear (por ejemplo, "Actualizar" o "Insertar").

Validación: Si el tipo de botón no es válido, se sale de la función.
Creación del Botón: Se define un botón con texto, color, fuente y una acción que se ejecutará al hacer clic.
Empaquetado: Se coloca el botón en el subpanel a la derecha.
Eventos de Hover: Se aplica un evento de hover al botón creado.

Resumen
Este archivo es parte de una aplicación GUI que utiliza tkinter para gestionar la interfaz. Se centra en la configuración y creación de botones en diferentes menús (lateral, superior y subpanel). Utiliza módulos personalizados para manejar eventos y configuraciones específicas, y permite personalizar los botones con texto, iconos, colores y acciones al hacer clic.

```python
import tkinter as tk
import modulos.botones.btn_hover as btn_hover
import config.config as colores
from tkinter import font
import modulos.ejecucion_click.click_btn_insertData_BD as ejecutar

def configuracion_btn_menu_lateral(self, boton, text, icono, activo):
  """Configura los parámetros del botón en el menú lateral."""
  boton.config(
    text=f" {icono}  {text}",
    anchor="w",
    font=font.Font(family="FontAwesome", size=15),
    width=20,
    height=1,
    pady=5,
    bg=colores.COLOR_BTN,
    fg="white",
    bd=1,
    relief="raised",
    highlightbackground=colores.COLOR_MENU_LATERAL,
    highlightcolor=colores.COLOR_MENU_LATERAL,
    highlightthickness=2
  )
  boton.pack(side=tk.TOP, pady=8)
  if not activo:
    btn_hover.hover_event(self, boton)
  else:
    self.boton_activo = boton


def configuracion_btn_menu_superior(self, boton, text, icono, activo):
  """Configura los parámetros del botón en el menú superior."""
  boton.config(
    text=f" {icono}  {text}",
    anchor="w",
    font= font.Font(family="FontAwesome", size=12),
    width=15,
    height=1,
    pady=5,
    bg=colores.COLOR_BTN,
    fg="white",
    bd=1,
    relief="raised",
    highlightbackground=colores.COLOR_BARRA_SUPERIOR,
    highlightcolor=colores.COLOR_BARRA_SUPERIOR,
    highlightthickness=2
  )
  boton.pack(side=tk.RIGHT, padx=15)
  btn_hover.hover_event_sup(self, boton)
```

```python
def crear_boton_sub_panel(self, tipo_boton):
  """Crea un botón en el subpanel según el tipo especificado."""
  texto = {
    "Actualizar": "Actualizar",
    "Insertar": "Insertar"
  }.get(tipo_boton)

  if not texto:
    return  # Si el tipo de botón no es válido, salir del método

  # Crear el botón
  boton = tk.Button(
    self.panel_acciones_cuerpo,
    text=texto,
    padx=20,
    bg=colores.COLOR_BTN,
    font=("Arial", 12, "bold"),
    fg="white",

    #este metodo se ejecutara cuando se presione el boton
    #Necesario aquiya que los botones se crean automaticamente
    command=lambda: ejecutar.acciones_botones_sub_panel(
      self, self.titulo_panel_administracion, boton)
  )

  # Empaquetar el botón
  boton.pack(side="right", padx=20)
  btn_hover.hover_event(self, boton)
```

btn_hover

Url modulos/botones/btn_hover

Este archivo maneja los hover que tendran los botones y los label del programa, se han creado varios para casos en especificos.

```
import config.config as colores

def hover_event(self, boton):
  # Verifica si el botón es el activo para aplicar hover
  def on_enter(e):
    if self.boton_activo != boton:  # Evita hover en el botón activo
      boton.config(bg=colores.COLOR_MENU_CURSOR_ENCIMA,
             cursor="hand2", fg="white")

  def on_leave(e):
    if self.boton_activo != boton:  # Evita restaurar en el botón activo
      boton.config(bg=colores.COLOR_BTN, fg="white")
  boton.bind("<Enter>", on_enter)
  boton.bind("<Leave>", on_leave)


def hover_event_sup(self, boton_sup):
  # Verifica si el botón es el activo para aplicar hover
  def on_enter(e):
    if self.boton_activo_sup != boton_sup:  # Evita hover en el botón activo
      boton_sup.config(bg=colores.COLOR_MENU_CURSOR_ENCIMA,
               cursor="hand2", fg="white")

  def on_leave(e):
    if self.boton_activo_sup != boton_sup:  # Evita restaurar en el botón activo
      boton_sup.config(bg=colores.COLOR_BTN, fg="white")
  boton_sup.bind("<Enter>", on_enter)
  boton_sup.bind("<Leave>", on_leave)



def hover_event_Exit(self, boton_sup):
  # Verifica si el botón es el activo para aplicar hover
  def on_enter(e):
    boton_sup.config(cursor="hand2")
    boton_sup.config(image=self.exit2)

  def on_leave(e):
    boton_sup.config(cursor="arrow")
    boton_sup.config(image=self.exit)

  boton_sup.bind("<Enter>", on_enter)
  boton_sup.bind("<Leave>", on_leave)
```

```
def hover_event_minimizar(self, boton_sup):
  # Verifica si el botón es el activo para aplicar hover
  def on_enter(e):
    boton_sup.config(cursor="hand2")
    boton_sup.config(image=self.min2)

  def on_leave(e):
    boton_sup.config(cursor="arrow")
    boton_sup.config(image=self.min)

  boton_sup.bind("<Enter>", on_enter)
  boton_sup.bind("<Leave>", on_leave)


def hover_event_maximizar(self, boton_sup):
  # Verifica si el botón es el activo para aplicar hover
  def on_enter(e):
    if self.winfo_toplevel().state() == "zoomed" and
self.winfo_toplevel().overrideredirect():
      boton_sup.config(image=self.max)
      boton_sup.config(cursor="hand2")
    else:
      boton_sup.config(image=self.maxmax)
      boton_sup.config(cursor="hand2")


  def on_leave(e):
    if self.winfo_toplevel().state() == "zoomed" and
self.winfo_toplevel().overrideredirect():
      boton_sup.config(image=self.maxmax)
      boton_sup.config(cursor="arrow")
    else:
      boton_sup.config(image=self.max)
      boton_sup.config(cursor="arrow")

  boton_sup.bind("<Enter>", on_enter)
  boton_sup.bind("<Leave>", on_leave)



def hover_event_label(self, label):
  def on_enter(e):
    label.configure(cursor="hand2", foreground="red")

  def on_leave(e):
    label.configure(foreground=colores.COLOR_MENU_CURSOR_ENCIMA,)

  label.bind("<Enter>", on_enter)
  label.bind("<Leave>", on_leave)
```

btn_selected

Url: modulos/btn/btn_selected.py

Este arcivo contiene el codigo que se encarga de la selección de los botones al hacer click y cambiarles la propiedad active a True para que el hover no tenga efecto en ese boton, ademas incorpora un metodo de resetear el boton activo que se encuentra en la barra superior y desactivarlo.

```
import config.config as colores

def marcar_boton(self, boton, btn_info, es_superior=False):
  # Determinar el botón activo y la lista de botones según el tipo
  if es_superior:
    boton_activo = self.boton_activo_sup
    btn_info_lista = self.btn_info_sup
  else:
    boton_activo = self.boton_activo
    btn_info_lista = self.btn_info

  # Si hay un botón activo, restaurar su color
  if boton_activo:
    boton_activo.config(bg=colores.COLOR_BTN)
    # Actualiza el estado del botón anterior
    for btn in btn_info_lista:
      if btn["text"] == boton_activo.cget("text").strip():
        btn["activo"] = False  # Desmarcar el botón anterior

  # Marca el botón seleccionado
  boton.config(bg=colores.COLOR_MENU_CURSOR_ENCIMA)
  if es_superior:
    self.boton_activo_sup = boton  # Actualiza el botón activo superior
  else:
    self.boton_activo = boton  # Actualiza el botón activo lateral

  btn_info["activo"] = True  # Marca el botón actual como activo


def reset_btn_sup(self):
  if self.boton_activo_sup:
    self.boton_activo_sup.config(bg=colores.COLOR_BTN)
    for btn in self.btn_info_sup:
      if btn["text"] == self.boton_activo_sup.cget("text").strip():
        btn["activo"] = False
```

Datos: definir_btns.py

Url: modulos/datos/definir_btns.py

Este archivo solo contiene la definicion de los botones superiores y laterales.

```
from panel_Principal.form_maestro_design import *

# definir_botones laterales y superiores
 # Agregar estado a cada botón del menu lateral y superior para marcar el botón
activo con el metodo marcar_boton
def definir_btn_menu_lateral(self):
  return [
    {"text": "Inicio", "icon": "\uf0e4", "activo": False},
    {"text": "Libros", "icon": "\uf0f6", "activo": False},
    {"text": "Autores", "icon": "\uf0f6", "activo": False},
    {"text": "Editoriales", "icon": "\uf0f6", "activo": False},
    {"text": "Autor-Libro", "icon": "\uf0f6", "activo": False},
    {"text": "Crear Tabla", "icon": "\uf0f6", "activo": False},
  ]

def definir_btn_menu_superior(self):
  return [
    {"text": "Actualizar", "icon": "\uf021", "activo": False},
    {"text": "Eliminar", "icon": "\uf2ed", "activo": False},
    {"text": "Insertar", "icon": "\uf067", "activo": False},
    {"text": "Buscar", "icon": "\uf002", "activo": False},
    ]
```

datos_para_insertar.py

Url: modulos/datos/datos_para _insertar.py

En este archivo se han colocado los datos que se van a pintar en el panel de insercion dependiendo del nombre del panel en el que se encuentra.

```
def datos_llenar_insertar(self, tipo_panel):
  """Devuelve los campos a llenar según el tipo de panel especificado."""
  campos = {
    "Libros": ("titulo", "año", "autor", "editorial"),
    "Editoriales": ("nombre", "direccion", "telefono"),
    "Autores": ("nombre", "apellido", "nacionalidad"),
    "CambioAutorLibro": ("libro", "autor")
  }

  return campos.get(tipo_panel, campos["CambioAutorLibro"])
```

Ejemplo en el panel Libros se pintara asi:

En el Panel Editoriales asi:

Etc...

transisiones.py

Url: modulos/efectos_visuales/transisiones.py

Este modulo se encarga de crear las transiciones entre subpaneles insertar ,actualizar y buscar

slide_out(self, ventana):

Esta función toma una ventana como argumento y la desliza hacia arriba hasta que desaparece de la pantalla.

Utiliza una función auxiliar llamada mover_ventana(i) que se encarga de mover la ventana hacia arriba en incrementos de 5 píxeles cada 10 milisegundos.

Cuando la ventana se ha deslizado completamente hacia arriba, se oculta usando ventana.place_forget().

slide_in(self, ventana, tiempo_espera=800):

Esta función toma una ventana y un tiempo de espera (en milisegundos) como argumentos.

Primero, guarda las coordenadas y dimensiones originales de la ventana.
Luego, coloca la ventana fuera de la vista, por encima de la pantalla.

Utiliza una función auxiliar llamada mover_ventana(i) que se encarga de mover la ventana hacia abajo en incrementos de 5 píxeles cada 10 milisegundos.

Después de un tiempo de espera (especificado por tiempo_espera), se inicia el movimiento de la ventana hacia abajo hasta que alcanza su posición original.

La función toggle(self, ventana), que no está implementada en el código se encargaría de mostrar u ocultar el panel lateral, llamando a slide_in o slide_out según el estado actual del panel

```
def slide_out(self, ventana):
  def mover_ventana(i):
    if i <= 200:  # Continuar hasta que haya deslizado completamente
      # Mover hacia arriba
      ventana.place(x=ventana.winfo_x(), y=ventana.winfo_y() - i)
      self.update_idletasks()
      # Llama a sí mismo con el nuevo valor
      self.after(10, mover_ventana, i + 5)
    else:
```

```
            ventana.place_forget()  # Ocultar la ventana al final del movimiento
        # Iniciar el movimiento
        mover_ventana(0)


    def slide_in(self, ventana, tiempo_espera=800):  # tiempo_espera en milisegundos
        original_x = self.coordenadas[0]
        original_y = self.coordenadas[1]
        original_width = self.ancho_cuerpo
        original_height = self.alto_cuerpo

        # Desactivar el ajuste automático de tamaño
        ventana.update_idletasks()  # Asegúrate de que el tamaño se calcule correctamente
        # Coloca la ventana fuera de la vista inicialmente
        ventana.place(x=original_x, y=original_y - original_height, width=original_width)

        # Función para mover la ventana hacia abajo
        def mover_ventana(i):
            if i <= original_height:
                ventana.place(x=original_x, y=original_y - original_height + i,
    width=original_width)
                self.update_idletasks()
                self.after(10, mover_ventana, i + 5)
            else:
                # Asegúrate de que esté en la posición original al final
                ventana.place(x=original_x, y=original_y, width=original_width,
    height=original_height)

        # Esperar antes de iniciar el movimiento
        self.after(tiempo_espera, mover_ventana, 0)


    ## Efecto de mostrar ocultar panel al hacer clic a un boton
        # def toggle(self, ventana):
        #     if self.ventanas.get(ventana) is None:
        #         # Guardar información detallada
        #         self.ventanas[ventana] = self.get_window_details(ventana)

        #     if self.visible:
        #         self.slide_out(ventana)
        #     else:
        #         self.slide_in(ventana)
```

Ejecucion_click

click_btn_menu_lateral

Url: modulos/ejecucion_click/click_btn_menu_lateral.py

Este modulo es el encargado de al hacer click en un boton del menu lateral instanciar la clase y mostrar los datos en el panel_datos

Importaciones:

modulos.botones.btn_selected: Importa el módulo que contiene funciones relacionadas con la selección de botones y para reiniciar los botones superiores si estan marcados.

modulos.paneles.crear_panel_admin: Importa el módulo que contiene funciones para crear y cargar datos en un panel de administración.

clases.libros, clases.editoriales, clases.autores, clases.autorlibro: Importa las clases que representan las entidades principales del sistema (libros, editoriales, autores y la relación entre autores y libros).

modulos.paneles.crear_tabla_bd: Importa el módulo que contiene funciones para crear la tabla que se mostrara.

Método instanciar_y_marcar(self, boton, btn_info):

Este método se encarga de asociar diferentes métodos al hacer clic en cualquier botón del menú lateral.
Llama al método marcar_boton() del módulo btn_selected para marcar el botón seleccionado.
Luego llama al método instanciar() pasando el texto del botón seleccionado.

Método instanciar(self, clase):

Este método se encarga de instanciar la clase correspondiente al nombre del botón seleccionado y cargar los datos en el panel de administración.

Primero, llama al método reset_btn_sup() del módulo btn_selected para restablecer el estado de los botones superiores.
Luego, crea un diccionario clases_mapping que asocia los nombres de las clases con las instancias de las clases y los atributos que contienen los registros.
Si la clase seleccionada está en el diccionario clases_mapping, intenta instanciar la clase correspondiente, obtener los registros y cargarlos en el panel de administración utilizando la función cargarDatos() del módulo crear_panel_admin.
Si la clase seleccionada es "Inicio", llama a cargarDatos() del módulo crear_panel_admin sin parámetros.

Si la clase seleccionada no está en el diccionario clases_mapping, llama a la función nueva_tabla_Base_Datos() del módulo crear_tabla para crear una nueva tabla la cual se creara en la base de datos.

```
import modulos.botones.btn_selected as btn_selected
import modulos.paneles.crear_panel_admin as crear_panel_admin
from clases.libros import Libros
```

```python
from clases.editoriales import Editoriales
from clases.autores import Autores
from clases.autorlibro import AutorLibro
import modulos.paneles.crear_tabla_bd as crear_tabla


#metodo para asocial diferentes metodos llamado al hacer clic en cualquier boton del menu lateral
def instanciar_y_marcar(self, boton, btn_info):
  # Llama a los métodos deseados
  btn_selected.marcar_boton(self, boton, btn_info)  # Marca el botón
  instanciar(self, btn_info["text"])   # Llama a instanciar

#metodo el cual instancia la clase dependiendo del nombre del boton y carga los datos en el panel
def instanciar(self, clase):
  btn_selected.reset_btn_sup(self)
  self.campo_selected_table = {}
  clases_mapping = {
    "Libros": (Libros, "libros_con_autor_y_editorial"),
    "Autores": (Autores, "autores"),
    "Editoriales": (Editoriales, "editoriales"),
    "Autor-Libro": (AutorLibro, "autor_libros")
  }

  if clase in clases_mapping:
    try:
      self.registros = None
      clase_obj, atributo = clases_mapping[clase]
      instancia = clase_obj()
      self.registros = getattr(instancia, atributo)
      self.indice_actual = 0
      self.titulo_panel_administracion = clase
      crear_panel_admin.cargarDatos(self)
    except Exception as e:
      print(f"Error al instanciar {clase.lower()}: {e}")
  elif clase == "Inicio":
    crear_panel_admin.cargarDatos(self, "Inicio")
  else:
    crear_tabla.nueva_tabla_Base_Datos(self)
```

click_btn_menu_sup

Url: modulos/ejecucion_click/click_btn_menu_sup.py

Importaciones:
clases.libros, clases.editoriales, clases.autores, clases.autorlibro: Importa las clases que representan las entidades principales del sistema (libros, editoriales, autores y la relación entre autores y libros).
modulos.botones.btn_selected: Importa un módulo que contiene funciones relacionadas con la selección de botones.
tkinter.messagebox: Importa el módulo de mensajes de Tkinter para mostrar cuadros de diálogo.
modulos.efectos_visuales.transisiones: Importa un módulo que contiene funciones para realizar transiciones visuales.
modulos.ejecucion_click.click_btn_menu_lateral: Importa el módulo que contiene funciones relacionadas con la interacción con los botones del menú lateral.
modulos.paneles.tabla: Importa un módulo que contiene funciones para actualizar la tabla de datos.

Método acciones(self, buton, btn_info_sup):
Este método se encarga de manejar las acciones realizadas al hacer clic en los botones superiores.
Llama al método marcar_boton() del módulo btn_selected para marcar el botón seleccionado.
Obtiene el texto del botón seleccionado y realiza diferentes acciones según el texto.

Si el texto del botón es "Actualizar" o "Eliminar" y no hay una fila seleccionada en la tabla, muestra un mensaje de error.
Si el texto del botón es "Eliminar", muestra un cuadro de diálogo de confirmación y, si el usuario acepta, llama al método eliminar_registro().
Luego, llama a los métodos transicion_paneles_if_true() y creacion_acciones_cuerpo_datos() para realizar transiciones y cargar acciones en el panel de acciones.

Finalmente, llama a la función slide_in() del módulo transisiones para mostrar el panel de acciones.
Método eliminar_registro(self):

Este método se encarga de eliminar el registro seleccionado en la tabla.
Crea un diccionario model_mapping que asocia los nombres de las clases con las instancias de las clases.
Obtiene la clase correspondiente al título del panel de administración y llama al método eliminar_registro() de esa clase, pasando el ID del registro seleccionado.
Si la eliminación es exitosa, muestra un mensaje de información; de lo contrario, muestra un mensaje de error.

Finalmente, llama a la función instanciar() del módulo click_btn_menu_lateral para actualizar el panel de administración.

Resumen:
Este código maneja la lógica de las acciones realizadas al hacer clic en los botones superiores, como actualizar, eliminar, insertar y buscar Cuando se elimina un registro, se muestra un cuadro de diálogo de confirmación y, si el usuario acepta, se elimina el registro y se actualiza el panel de administración.

```python
from clases.libros import Libros
from clases.editoriales import Editoriales
from clases.autores import Autores
from clases.autorlibro import AutorLibro
import modulos.botones.btn_selected as btn_selected
from tkinter import messagebox
import modulos.efectos_visuales.transisiones as transition
import modulos.ejecucion_click.click_btn_menu_lateral as click_btn_menu_lateral
import modulos.paneles.tabla as tabla

def acciones(self, buton, btn_info_sup):
  btn_selected.marcar_boton(self, buton, btn_info_sup, True)

  # Manejo de acciones según el texto del botón
  accion_texto = btn_info_sup["text"]


  if self.titulo_panel_administracion == "Bienvenido a eDe-Lib":
    messagebox.showinfo("Error", f"Debe seleccionar en el menu lateral que desea
{accion_texto}")
    return

  # Verificar si hay una fila seleccionada antes de procesar otras acciones
  if accion_texto in ["Actualizar", "Eliminar"] and not self.campo_selected_table:
    tabla.actualizar_tabla(self, self.registros)
    messagebox.showinfo("Error", "Seleccione una fila en la tabla para actualizar")
    return

  if accion_texto == "Eliminar":
    # Mensaje de confirmación basado en el título del panel
    if self.titulo_panel_administracion == "Libros":
      mensaje = f"¿Está seguro de que desea eliminar el libro
'{self.campo_selected_table.get('titulo', 'desconocido')}'?"
    elif self.titulo_panel_administracion == "Editoriales":
      mensaje = f"¿Está seguro de que desea eliminar la editorial
'{self.campo_selected_table.get('nombre', 'desconocido')}'?"
    elif self.titulo_panel_administracion == "Autores":
      nombre_autor = self.campo_selected_table.get('nombre', 'desconocido')
      apellido_autor = self.campo_selected_table.get('apellido', 'desconocido')
      mensaje = f"¿Está seguro de que desea eliminar el autor '{nombre_autor}
{apellido_autor}'?"
    else:
      mensaje = "¿Está seguro de que desea eliminar el registro seleccionado?"

    respuesta = messagebox.askyesno("Confirmar Eliminación", mensaje)
    if respuesta:  # Si el usuario acepta
      eliminar_registro(self)
    return

  self.transicion_paneles_if_true()
```

```
  self.creacion_acciones_cuerpo_datos(accion_texto)
  transition.slide_in(self, self.panel_acciones_cuerpo)



def eliminar_registro(self):
  model_mapping = {
    "Libros": Libros,
    "Editoriales": Editoriales,
    "Autores": Autores,
    "AutorLibro": AutorLibro
  }

  modelo = model_mapping.get(self.titulo_panel_administracion, AutorLibro)
  respuesta = modelo().eliminar_registro(self.campo_selected_table["id"])

  if respuesta is not None:
    messagebox.showinfo("Informacion", f"{self.titulo_panel_administracion}
eliminado correctamente")
  else:
    messagebox.showerror("Error", f"Error al eliminar el
{self.titulo_panel_administracion.lower()}")

  click_btn_menu_lateral.instanciar(self, self.titulo_panel_administracion)
```

click_btn_insertData_BD

Url: modulos/ejecucion_click/click_btn_insertData_BD.py

Este modulo es el encargado de tomar las acciones de los botones de los subpaneles insertar, actualizar y buscar

Importaciones:

tkinter.messagebox: Importa el módulo de mensajes de Tkinter para mostrar cuadros de diálogo.
clases.libros, clases.editoriales, clases.autores: Importa las clases que representan las entidades principales del sistema (libros, editoriales y autores).
modulos.ejecucion_click.click_btn_menu_lateral: Importa el módulo que contiene funciones relacionadas con la interacción con los botones del menú lateral.

Función acciones_botones_sub_panel(self, tabla, boton):

Esta función se encarga de manejar las acciones realizadas al hacer clic en los botones del subpanel.
Si el texto del botón es "Actualizar", llama a la función actualizar().
Si el texto del botón es "Insertar", llama a la función insertar().
Función obtener_datos_editorial(cadena_busqueda):

Esta función se encarga de buscar una editorial en la base de datos utilizando una cadena de búsqueda.
Crea una instancia de la clase Editoriales y llama al método filtrar() para buscar la editorial.
Devuelve el ID de la editorial encontrada o None si no se encuentra ninguna.

Función mostrar_mensaje(resultado, entidad, accion):

Esta función se encarga de mostrar un mensaje de información o error según el resultado de una acción.
Si el resultado es válido, muestra un mensaje de información indicando que la acción (actualización o inserción) se realizó correctamente.
Si el resultado es None, muestra un mensaje de error indicando que hubo un problema al realizar la acción.

Función actualizar(self, tabla):

Esta función se encarga de actualizar los datos de un registro existente en la base de datos.
Obtiene los nuevos datos de los campos de actualización y los almacena en el diccionario nuevos_datos.
Según la tabla seleccionada (Libros, Autores o Editoriales), llama al método modificar_registro() de la clase correspondiente, pasando el ID del registro y los nuevos datos.
Llama a la función mostrar_mensaje() para mostrar un mensaje de información o error según el resultado de la actualización.
Finalmente, llama a la función instanciar() del módulo click_btn_menu_lateral para actualizar el panel de administración.
Función insertar(self, tabla):

Esta función se encarga de insertar un nuevo registro en la base de datos.
Obtiene los nuevos datos de los campos de inserción y los almacena en el diccionario nuevos_datos.
Según la tabla seleccionada (Libros, Autores o Editoriales), llama al método crear_registro() de la clase correspondiente, pasando los nuevos datos.
Llama a la función mostrar_mensaje() para mostrar un mensaje de información o error según el resultado de la inserción.
Finalmente, llama a la función instanciar() del módulo click_btn_menu_lateral para actualizar el panel de administración.

Resumen:
Este código maneja la lógica de actualización e inserción de registros en la base de datos, mostrando mensajes de información o error según el resultado de las operaciones. También incluye una función para buscar una editorial en la base de datos utilizando una cadena de búsqueda. Y actualiza el panel despues de cada accion si la hay.

```
from tkinter import messagebox
from clases.libros import Libros
from clases.editoriales import Editoriales
from clases.autores import Autores
import modulos.ejecucion_click.click_btn_menu_lateral as click_btn_menu_lateral


def acciones_botones_sub_panel(self, tabla, boton):
  if boton['text'] == "Actualizar":
    actualizar(self, tabla)
  elif boton['text'] == "Insertar":
    insertar(self, tabla)


def obtener_datos_editorial(cadena_busqueda):
  edit = Editoriales()
  campos_a_buscar = ['nombre']
  resul = edit.filtrar(campos_a_buscar, cadena_busqueda)
  return resul[0]['id'] if resul else None



def mostrar_mensaje(resultado, entidad, accion):
  if resultado is not None:
    messagebox.showinfo("Informacion", f"{entidad} {'actualizado' if accion ==
'update' else 'creado'} correctamente" + (f" con id: {resultado}" if accion ==
'insert' else ""))
  else:
    messagebox.showerror("Error", f"Error al {'actualizar' if accion == 'update'
```

```
else 'crear'} el {entidad}")

def actualizar(self, tabla):
  nuevos_datos = {}
  id = self.campos_actualizar["id"].get()

  if tabla == "Libros":
    nuevos_datos['titulo'] = self.campos_actualizar["titulo"].get()
    nuevos_datos['anio'] = self.campos_actualizar["anio"].get()
    cadena_busqueda = self.campos_actualizar["editorial"].get()
    nuevos_datos['id_editorial'] = obtener_datos_editorial(cadena_busqueda)
    resultado = Libros().modificar_registro(id, nuevos_datos)
    mostrar_mensaje(resultado, "Libro", "update")
    click_btn_menu_lateral.instanciar(self, "Libros")

  elif tabla == "Autores":
    nuevos_datos['nombre'] = self.campos_actualizar["nombre"].get()
    nuevos_datos['apellido'] = self.campos_actualizar["apellido"].get()
    nuevos_datos['nacionalidad'] = self.campos_actualizar["nacionalidad"].get()
    resultado = Autores().modificar_registro(id, nuevos_datos)
    mostrar_mensaje(resultado, "Autor", "update")
    click_btn_menu_lateral.instanciar(self, "Autores")

  elif tabla == "Editoriales":
    nuevos_datos['nombre'] = self.campos_actualizar["nombre"].get()
    nuevos_datos['direccion'] = self.campos_actualizar["direccion"].get()
    nuevos_datos['telefono'] = self.campos_actualizar["telefono"].get()
    resultado = Editoriales().modificar_registro(id, nuevos_datos)
    mostrar_mensaje(resultado, "Editorial", "update")
    click_btn_menu_lateral.instanciar(self, "Editoriales")

  elif tabla == "Autor-Libro":
    # Lógica para Autor-Libro si es necesario
    pass

def insertar(self, tabla):
  nuevos_datos = {}

  if tabla == "Libros":
    nuevos_datos['titulo'] = self.campos_insertar["titulo"].get()
    nuevos_datos['anio'] = self.campos_insertar["año"].get()
    cadena_busqueda = self.campos_insertar["editorial"].get()
    nuevos_datos['id_editorial'] = obtener_datos_editorial(cadena_busqueda)
    resultado = Libros().crear_registro(nuevos_datos)
    mostrar_mensaje(resultado, "Libro", "insert")
    click_btn_menu_lateral.instanciar(self, "Libros")

  elif tabla == "Autores":
    nuevos_datos['nombre'] = self.campos_insertar["nombre"].get()
    nuevos_datos['apellido'] = self.campos_insertar["apellido"].get()
    nuevos_datos['nacionalidad'] = self.campos_insertar["nacionalidad"].get()
```

```
        resultado = Autores().crear_registro(nuevos_datos)
        mostrar_mensaje(resultado, "Autor", "insert")
        click_btn_menu_lateral.instanciar(self, "Autores")

    elif tabla == "Editoriales":
        nuevos_datos['nombre'] = self.campos_insertar["nombre"].get()
        nuevos_datos['direccion'] = self.campos_insertar["direccion"].get()
        nuevos_datos['telefono'] = self.campos_insertar["telefono"].get()
        resultado = Editoriales().crear_registro(nuevos_datos)
        mostrar_mensaje(resultado, "Editorial", "insert")
        click_btn_menu_lateral.instanciar(self, "Editoriales")


    elif tabla == "Autor-Libro":
        # Lógica para Autor-Libro si es necesario
        pass
```

Paneles:

panel_bienvenida.py

Url: modulos/paneles/panel_bienvenida.py

Una vez creada la interfaz en la clase principal de los Paneles **FormMaestro** y creado el menu lateral y superior el panel siguiente es el de bienvenida el cual se crea dentro del frame cuerpo_principal.

Importaciones:

tkinter.ttk: Importa el módulo de widgets mejorados de Tkinter.
tkinter: Importa el módulo principal de Tkinter.
config.config: Importa el módulo de configuración que contiene variables de color.
modulos.botones.btn_hover: Importa un módulo que contiene funciones relacionadas con el efecto de hover en los botones.

Función crear_panel_bienvenida(self):

Esta función se encarga de crear el panel de bienvenida de la aplicación.
Establece el título del panel de administración como "Bienvenido a eDe-Lib".
Crea un Frame principal llamado panel_inicio y lo coloca en el cuerpo_principal de la aplicación.

Crea un Canvas dentro del panel_inicio para mostrar la imagen de fondo.
Llama al método redimensionar_imagen_fondo_panel_bienvenida() para cargar y redimensionar la imagen de fondo inicialmente.

Vincula el evento de redimensionamiento del panel_inicio al método redimensionar_imagen_fondo_panel_bienvenida() para que la imagen se redimensione cuando se cambie el tamaño del panel.

Crea varios Label dentro del Canvas para mostrar el título, información, autor, teléfono, correo electrónico y derechos de autor.

Aplica el efecto de hover a los Label del teléfono y el correo electrónico utilizando la función hover_event_label() del módulo btn_hover.

Resumen: este código se encarga de crear el panel de bienvenida de la aplicación, que incluye una imagen de fondo redimensionable, un título, información sobre la plataforma, detalles del autor y derechos de autor. También agrega un efecto de hover a los elementos de contacto (teléfono y correo electrónico) para mejorar la experiencia del usuario.

```
from tkinter import ttk
import tkinter as tk
import config.config as colores
import modulos.botones.btn_hover as btn_hover


def crear_panel_bienvenida(self):
    self.titulo_panel_administracion = "Bienvenido a eDe-Lib"

    # Crear el panel principal
    self.panel_inicio = tk.Frame(self.cuerpo_principal, bg="white")
    self.panel_inicio.place(relwidth=1, relheight=1)

    # Crear un Canvas para la imagen de fondo
    self.canvas = tk.Canvas(self.panel_inicio, highlightthickness=0)
    self.canvas.place(relwidth=1, relheight=1)

    # Llamar a redimensionar_imagen para cargar la imagen inicialmente
    #el metodo redimensionamiento de la imagen de fondo se encuentra en
form_maestro
    self.redimensionar_imagen_fondo_panel_bienvenida()

    # Vincular el evento de redimensionamiento
    self.panel_inicio.bind("<Configure>",
self.redimensionar_imagen_fondo_panel_bienvenida)

    # Crear el Label con texto
    self.label_inicio = ttk.Label(self.canvas,
text=self.titulo_panel_administracion,
                  foreground=colores.COLOR_MENU_CURSOR_ENCIMA, font=("Arial", 36,
"bold"))
    self.label_inicio.place(relx=0.5, y=15, anchor="n")

    self.label_info= ttk.Label(self.canvas, text="Plataforma de gestión de
bibliotecas",
                  foreground=colores.COLOR_MENU_CURSOR_ENCIMA, font=("Arial", 16,
"bold"))
    self.label_info.place(relx=0.5, y=80, anchor="n")

    self.label_autor = ttk.Label(self.canvas, text="Desarrollado por Alexander
Galvez",
                 foreground=colores.COLOR_MENU_CURSOR_ENCIMA, font=("Arial", 10,
"bold"))
    self.label_autor.place(relx=1.0, rely=1.0, anchor='se', x=-30, y=-110)

    self.label_telefono = ttk.Label(self.canvas, text="Teléfono: +34 688 872 515",
                    foreground=colores.COLOR_MENU_CURSOR_ENCIMA, font=("Arial",
10, "bold"))
    self.label_telefono.place(relx=1.0, rely=1.0, anchor='se', x=-30, y=-90)

    self.label_correo = ttk.Label(self.canvas, text="Correo:
```

```
alexandergalvez880208@gmail.com",
                foreground=colores.COLOR_MENU_CURSOR_ENCIMA, font=("Arial", 10,
"bold"))
    self.label_correo.place(relx=1.0, rely=1.0, anchor='se', x=-30, y=-70)

    self.label_cpyright = ttk.Label(self.canvas, text="© 2025 eDe-Lib. Todos los
derechos reservados.",
                    foreground=colores.COLOR_MENU_CURSOR_ENCIMA, font=("Arial",
10, "bold"))
    self.label_cpyright.place(relx=1.0, rely=1.0, anchor='se', x=-30, y=-30)

    btn_hover.hover_event_label(self, self.label_telefono)
    btn_hover.hover_event_label(self, self.label_correo)
```

crear_panel_admin

Url: modulos/paneles/crear_panel_admin

Este panel se crea una vez creado el frame donde se contendran los datos en el padre **FormMaestro** Primeramente se creara panel_datos, tambien se creara panel_cuerpo donde iran reflejados los datos , tambien se llamara a la creacion de la tabla.

```
 def creacion_cuerpo_datos(self):
    self.panel_datos = tk.Frame(self.cuerpo_principal,
bg=colores.COLOR_CUERPO_PRINCIPAL)
    self.panel_datos.place(relwidth=1, relheight=1)

    self.panel_cuerpo = tk.Frame(self.panel_datos, bg=colores.COLOR_PANEL_INFO)

    # Metodo para centrar el panel_cuerpo
    def ajustar_panel():
      x, y = util_ventana.centrar_panel(self.panel_datos, self.ancho_cuerpo,
self.alto_cuerpo)
      self.panel_cuerpo.place(width=self.ancho_cuerpo, height=self.alto_cuerpo,
x=x, y=y)
      self.coordenadas = [x, y, self.ancho_cuerpo, self.alto_cuerpo]

    # Llama a ajustar_panel al configurar panel_datos
    self.panel_datos.bind("<Configure>", lambda event: ajustar_panel())

    tk.Label(self.panel_datos, text=f"Panel de Administración de
{self.titulo_panel_administracion}",
        bg=colores.COLOR_CUERPO_PRINCIPAL, fg=colores.COLOR_BARRA_SUPERIOR,
font=("Arial", 30, "bold")).place(relx=0.5, y=10, anchor="n")

    # Crear panel para la tabla
    self.panel_tabla = tk.Frame(self.panel_datos, bg="#FFFFFF")
    self.panel_tabla.place(relwidth=1, height=270, y=410)

    ajustar_panel()
```

Importaciones:

tkinter.ttk: Importa el módulo de widgets mejorados de Tkinter.
tkinter: Importa el módulo principal de Tkinter.
modulos.paneles.tabla: Importa el módulo que contiene funciones relacionadas con la creación de la tabla.
modulos.botones.btn_hover: Importa un módulo que contiene funciones relacionadas con el efecto de hover en los botones.
config.config: Importa el módulo de configuración que contiene variables de color.

Función cargarDatos(self, quepanel=None):

Esta función se encarga de cargar los datos en el panel principal de la aplicación.
Verifica si el panel actual es el panel de bienvenida ("Inicio") y, si es así, lo oculta y crea el panel de datos.

Crea un diccionario campos para almacenar los campos de entrada de cada columna.
Obtiene las columnas de los registros y crea un Frame para cada fila, con una etiqueta y un campo de entrada.

Crea dos botones "Siguiente" y "Anterior" para navegar entre los registros, y les aplica el efecto de hover.
Llama a la función mostrar_registro(self) para mostrar el registro actual en los campos de entrada.
Llama a la función crear_tabla(self) del módulo tabla para crear la tabla de datos.

Función mostrar_registro(self):

Esta función se encarga de mostrar el registro actual en los campos de entrada.
Verifica si hay registros disponibles y si el índice actual está dentro del rango.
Obtiene el registro actual y lo muestra en los campos de entrada, configurándolos como de solo lectura.

Función siguiente_registro(self):

Esta función se encarga de mostrar el siguiente registro.
Verifica si hay un siguiente registro disponible y, si es así, incrementa el índice actual y llama a mostrar_registro(self).

Función anterior_registro(self):

Esta función se encarga de mostrar el registro anterior.
Verifica si hay un registro anterior disponible y, si es así, decrementa el índice actual y llama a mostrar_registro(self).

En resumen, este código se encarga de cargar los datos en el panel principal de la aplicación, mostrando los registros en campos de entrada y proporcionando botones para navegar entre ellos. También crea una tabla de datos utilizando el módulo tabla. La función cargarDatos(self, quepanel=None) es la principal responsable de esta funcionalidad.

```
import tkinter as ttk
import tkinter as tk
import modulos.paneles.tabla as tabla
import modulos.botones.btn_hover as btn_hover
import config.config as colores

# este metodo se encarga de cargar los datos cuando se hace click en uno de los
botones laterales
# llenando el panel cuerpo donde estan los botones mas y menos para ir visualizando
los elementos
# ademas llama a mostrar registro y crea la tabla

def cargarDatos(self, quepanel=None):
  # Esto lo hago para si clican en inicio al cargar el panel no se rompa el
programa
  if quepanel == "Inicio" and self.titulo_panel_administracion != "Bienvenido a
eDe-Lib":
    if self.titulo_panel_administracion == "Nueva Tabla en la Base de Datos":
      self.panel_nueva_tabla.pack_forget()
      self.crear_panel_bienvenida()
    else:
      self.panel_datos.pack_forget()
      self.crear_panel_bienvenida()

  elif quepanel == "Inicio" and self.titulo_panel_administracion == "Bienvenido a
eDe-Lib":
    return

  else:
    self.panel_inicio.pack_forget()
    self.creacion_cuerpo_datos()
    self.campos = {}
    self.columnas = list(self.registros[0].keys())

     # Diccionario para mapear columnas a sus etiquetas
    etiquetas = {
    "anio": "Año"
    }

    for columna in self.columnas:

      # Contenedor de cada fila (etiqueta + campo)
      frame_fila = tk.Frame(self.panel_cuerpo, bg=colores.COLOR_PANEL_INFO)
      frame_fila.pack(pady=10, fill="x")

       # Obtener la etiqueta correspondiente, o usar la columna en sí
      texto_label = etiquetas.get(columna, columna.title())

      self.label = tk.Label(frame_fila, text=texto_label, width=15,
          anchor="w", font=("Arial", 14, "bold"),
bg=colores.COLOR_PANEL_INFO).pack(side="left", padx=5)
```

```
      self.campos[columna] = ttk.Entry(
        frame_fila, font=("Arial", 14, "bold"))
      self.campos[columna].pack(
        side="left", expand=True, fill="x", padx=15)


    btnMas = tk.Button(self.panel_cuerpo, text="Siguiente", padx=20,
bg=colores.COLOR_BTN, font=("Arial", 12, "bold"), fg="white",
              command= lambda:
              siguiente_registro(self))
    btnMas.pack(side="right", padx=20)
    btn_hover.hover_event_sup(self, btnMas)
    btnMenos = tk.Button(self.panel_cuerpo, text="Anterior", padx=20,
bg=colores.COLOR_BTN, font=("Arial", 12, "bold"), fg="white",
               command= lambda:
               anterior_registro(self))
    btnMenos.pack(side="right", padx=20)
    btn_hover.hover_event_sup(self, btnMenos)

    mostrar_registro(self)
    tabla.crear_tabla(self)

def mostrar_registro(self):
  """ Muestra el registro actual en los campos de texto """
  if not self.registros or self.indice_actual >= len(self.registros):
    return

  registro_actual = self.registros[self.indice_actual]

  for columna, campo in self.campos.items():
    campo.config(state="normal")
    campo.delete(0, tk.END)

    # Verifica si el campo está vacío antes de insertar
    valor = registro_actual.get(columna, "")  # Usa get para evitar KeyError
    campo.insert(0, valor if valor else "")  # Inserta un valor vacío si no hay
    campo.config(state="readonly")


def siguiente_registro(self):
  """ Muestra el siguiente registro """
  if self.indice_actual < len(self.registros) - 1:
    self.indice_actual += 1
    mostrar_registro(self)

def anterior_registro(self):
  """ Muestra el registro anterior """
  if self.indice_actual > 0:
    self.indice_actual -= 1
    mostrar_registro(self)
```

panel_cuerpo:

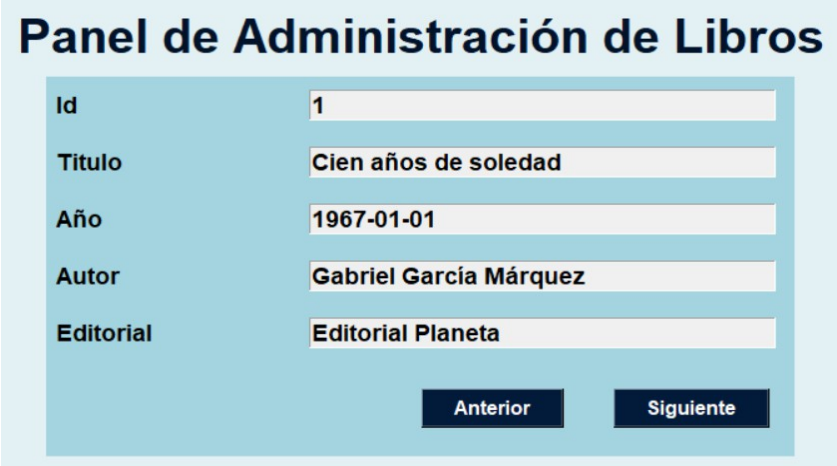


Tabla:

id	titulo	anio	autor	editorial
1	Cien años de soledad	1967-01-01	Gabriel García Márquez	Editorial Planeta
2	La casa de los espíritus	1982-01-01	Isabel Allende	Grupo Santillana
3	Harry Potter y la piedra filosofa	1997-01-01	J.K. Rowling	Penguin Random House
4	1984	1949-01-01	George Orwell	Editorial Planeta
5	La ciudad y los perros	1963-01-01	Mario Vargas Llosa	Editorial Anagrama
6	Rayuela	1963-01-01	Julio Cortázar	RBA Libros
7	Kafka en la orilla	2002-01-01	Haruki Murakami	Editorial Espasa
8	Orgullo y prejuicio	1813-01-01	Jane Austen	Alianza Editorial
9	Las aventuras de Tom Sawyer	1876-01-01	Mark Twain	Ediciones B
10	Al faro	1927-01-01	Virginia Woolf	Tusquets Editores
11	El amor en los tiempos del cól	1985-01-01	None	Editorial Planeta
12	Crónica de una muerte anunci	1981-01-01	None	Editorial Planeta

Panel_datos

Tabla:

Url: modulos/paneles/tabla.py

Este modulo es el encargado de la creacion de la tabla, cuyod parametros ya han sido cargados en la variable `self.registros = []` de nuestra clase Padre al clicar en los botones laterales
`self.registros = getattr(instancia, atributo)`

Importaciones:

panel_Principal.form_maestro_design: Importa la clase que contiene el diseño principal de la aplicación.
config.config: Importa el módulo de configuración que contiene variables de color.

Función crear_tabla(self):

Esta función se encarga de crear la tabla en el panel de la aplicación.
Obtiene las columnas de los registros y las almacena en la variable self.columnas.
Crea un estilo personalizado para la tabla, configurando los colores y las fuentes de las cabeceras y el contenido.

Crea un objeto Treeview de Tkinter.ttk y lo asigna a self.tabla.
Configura las cabeceras de la tabla, estableciendo el texto y el ancho de cada columna.
Inserta los datos de los registros en la tabla.
Agrega una barra de desplazamiento vertical a la tabla y la configura.
Utiliza el sistema de cuadrícula (grid) para posicionar la tabla y la barra de desplazamiento en el panel.
Configura la expansión de la cuadrícula para que la tabla se expanda y llene el espacio disponible.
Asocia un evento de clic a la tabla, que se encuentra en el módulo form_maestro.

Función borrar_tabla(self):

Esta función se encarga de eliminar todos los registros de la tabla.
Utiliza el método get_children() de la tabla para obtener todos los elementos y luego los elimina uno por uno.

Función actualizar_tabla(self, nuevos_registros):

Esta función se encarga de actualizar la tabla con nuevos registros.
Primero llama a la función borrar_tabla(self) para eliminar todos los registros existentes.
Luego, inserta los nuevos registros en la tabla utilizando el método insert().

En resumen, este código define las funciones necesarias para crear, borrar y actualizar una tabla en la interfaz de usuario de la aplicación. La función crear_tabla(self) es la más importante, ya que se encarga de la configuración y el diseño de la tabla, mientras que borrar_tabla(self) y actualizar_tabla(self, nuevos_registros) permiten manipular el contenido de la tabla.

```python
from panel_Principal.form_maestro_design import *
import config.config as btn_config

"""Creacion de la Tabla"""
def crear_tabla(self):
  self.columnas = list(self.registros[0].keys())

  # Crear un estilo para la tabla
  style = ttk.Style()
  style.configure("Heading", background=btn_config.COLOR_CABECERA_TABLA)
  style.configure("Treeview.Heading", font=(
    "Arial", 14, "bold"))  # Cabeceras en 14 bold
  style.configure("Treeview", font=("Arial", 12),
          background=btn_config.COLOR_PANEL_INFO)  # Contenido en 12

  # Crear la tabla
  self.tabla = ttk.Treeview(
    self.panel_tabla, columns=self.columnas, show="headings", style="Treeview")

  # Configurar las cabeceras de la tabla
  for col in self.columnas:
    self.tabla.heading(col, text=col)
    self.tabla.column(col, width=200)

  # Insertar los datos en la tabla
  for registro in self.registros:
    self.tabla.insert("", "end", values=[
              registro[col] for col in self.columnas])

  # Opcional: Agregar una barra de desplazamiento
  scrollbar = tk.Scrollbar(
    self.panel_tabla, orient="vertical", command=self.tabla.yview)
  self.tabla.configure(yscroll=scrollbar.set)

  # Usar grid para la tabla y el scrollbar
  self.tabla.grid(row=0, column=0, columnspan=4, sticky="nsew")
  scrollbar.grid(row=0, column=1, sticky="ns")

  # Configurar la expansión del grid
  self.panel_tabla.grid_rowconfigure(0, weight=1)
  self.panel_tabla.grid_columnconfigure(0, weight=1)

  # Asociar el evento de clic a la tabla
  #dicho evento se encuentra en form_maestro
  self.tabla.bind("<ButtonRelease-1>", self.on_item_tabla_click)

def borrar_tabla(self):
  # Eliminar todos los registros de la tabla
  for item in self.tabla.get_children():
    self.tabla.delete(item)
```

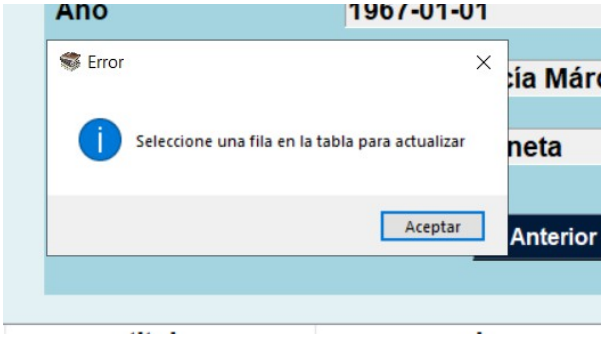
```
def actualizar_tabla(self, nuevos_registros):
  borrar_tabla(self)

  # Insertar nuevos registros en la tabla
  for registro in nuevos_registros:
    self.tabla.insert("", "end", values=[registro[col] for col in self.columnas])
```

subpanel_actualizar.py

Url: modulos/paneles/subpanel_actualizar.py

Este es el subpanel que se encarga de la Actualizacion de la clase que sea.
Modulo Automatico que se encarga de crear y llenar los elementos del panel_cuerpo una vez se selecciona una fila de la tabla y se hace click en el boton Actualizar del menu superior cogiendo esa selección y descomponiendola en columnas e imprimiendo los label correspondientes, los input u option mas el dato.



Si no se ah seleccionado nada en la tabla mostrara un mensaje de advertencia.

Importaciones:

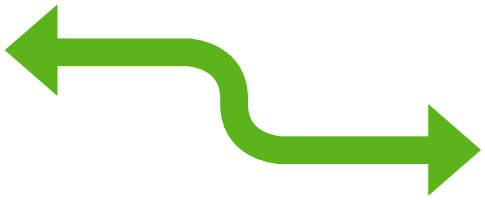
tkinter: Importa el módulo principal de Tkinter.
modulos.botones.btn_config: Importa el módulo que contiene la configuración de los botones.
config.config: Importa el módulo de configuración que contiene variables de color.
modulos.paneles.ccp_actualizar_libro: Importa el módulo que contiene la función para crear el panel de actualización de libros.
modulos.paneles.ccp_actualizar_autorlibro: Importa el módulo que contiene la función para crear el panel de actualización de autor-libro.

Función cargarDatosParaActualizar(self, tipo_boton):

Esta función se encarga de cargar los datos en el panel de actualización.
Establece el título del panel de actualización.
Crea una etiqueta con el título del panel.

Inicializa un diccionario self.campos_actualizar para almacenar los campos de entrada.
Verifica el título del panel de administración (self.titulo_panel_administracion) para determinar qué tipo de panel de actualización se debe crear.

Si es "Libros", llama a la función crear_cuerpo_panel_actualizar_libros() del módulo ccp_actualizar_libro.



ccp_actualizar_libro.py

Url:modulos/paneles/ccp_actualizar_libro.py

Importaciones:

tkinter.ttk: Importa el módulo de widgets mejorados de Tkinter.
tkinter: Importa el módulo principal de Tkinter.
config.config: Importa el módulo de configuración que contiene variables de color.
clases.editoriales: Importa la clase Editoriales que contiene información sobre las editoriales.
Función crear_cuerpo_panel_actualizar_libros(self, dat_filas):

Esta función se encarga de crear el cuerpo del panel de actualización de libros.
Inicializa un diccionario self.campos_actualizar para almacenar los campos de entrada, una vez en este panel cuando el usuario seleccione otro libro de la tabla se actualizara dicho panel.

Itera sobre cada columna y valor en dat_filas (que parece ser un diccionario con los datos de un libro).
Crea un Frame para cada fila, con una etiqueta y un campo de entrada.
Si la columna es "editorial", crea un Combobox (lista desplegable) con las opciones de las editoriales disponibles.

Para el resto de las columnas, crea un Entry (campo de entrada) y establece el valor inicial con el valor correspondiente de dat_filas.
Si la columna es "id" o "autor", configura el campo de entrada como de solo lectura y en color gris oscuro.

Agrega los campos de entrada al diccionario self.campos_actualizar para poder acceder a ellos posteriormente.

En resumen, este código crea el cuerpo del panel de actualización de libros, donde cada fila contiene una etiqueta y un campo de entrada (o un Combobox para la editorial) para que el usuario pueda modificar los datos del libro. Los campos "id" y "autor" se configuran como de solo lectura. El diccionario self.campos_actualizar se utiliza para almacenar y acceder a estos campos de entrada posteriormente si el usuario selecciona otro libro en la tabla.

```
from tkinter import ttk
import tkinter as tk
```

```
import config.config as colores
from clases.editoriales import Editoriales


def crear_cuerpo_panel_actualizar_libros(self,dat_filas):
  self.campos_actualizar = {}
  for columna, value in dat_filas.items():
    frame_fila = tk.Frame(
      self.panel_acciones_cuerpo, bg=colores.COLOR_PANEL_INFO)
    frame_fila.pack(pady=10, fill="x")
    tk.Label(frame_fila, text=columna, width=15,
        anchor="w", font=("Arial", 14, "bold"),
bg=colores.COLOR_PANEL_INFO).pack(side="left", padx=5)

    if columna == "editorial":
      editoriales = Editoriales()
      self.editorialesNombre = [
        editorial["nombre"] for editorial in editoriales.editoriales]
      # Lista de opciones para el autor o editorial
      opcionesEditoriales = self.editorialesNombre
      self.campos_actualizar[columna] = ttk.Combobox(
        frame_fila, values=opcionesEditoriales, font=("Arial", 14, "bold"))
      self.campos_actualizar[columna].set(
        self.editorialesNombre[0])
      self.campos_actualizar[columna].pack(
        side="left", expand=True, fill="x", padx=15)
    else:
      self.campos_actualizar[columna] = tk.Entry(
        frame_fila, font=("Arial", 14, "bold"))
      self.campos_actualizar[columna].insert(0, value)
      self.campos_actualizar[columna].pack(
        side="left", expand=True, fill="x", padx=15)

    if "id" in columna or "autor" in columna:
      self.campos_actualizar[columna].config(state="readonly", fg="darkgrey")
```

Si es "Autor-Libro", llama a la función crear_cuerpo_panel_actualizar_autor_libro() del módulo ccp_actualizar_autorlibro.

ccp_actualizar_autorlibro

Url: modulos/paneles/ccp_actualizar_autorlibro.py

Este modulo hace lo mismo pero con Autor-Libro, lo que crea directamente dos options con los autores y los libros que estan asignados, cabe destacar que este modulo habria que rectificarlo a futuro ya que no tiene mucha logica puesto que trae de la tabla autor-libro el codigo aquí no se ah llegado a probar.

```
from tkinter import ttk
import tkinter as tk
import config.config as colores
from clases.libros import Libros
from clases.autores import Autores


def crear_cuerpo_panel_actualizar_autor_libro(self,dat_filas):
  self.campos_actualizar = {}
  for columna, value in dat_filas.items():
    frame_fila = tk.Frame(
      self.panel_acciones_cuerpo, bg=colores.COLOR_PANEL_INFO)
    frame_fila.pack(pady=10, fill="x")
    tk.Label(frame_fila, text=columna, width=15,
         anchor="w", font=("Arial", 14, "bold"),
bg=colores.COLOR_PANEL_INFO).pack(side="left", padx=5)
    if columna=="libro":
      libros = Libros()
      self.librosNombres = [
        libro["titulo"] for libro in libros.libros]
      # Lista de opciones para el autor o editorial
      opcionesLibros = self.librosNombres
      self.campos_actualizar[columna] = ttk.Combobox(
        frame_fila, values=opcionesLibros, font=("Arial", 14, "bold"))
      self.campos_actualizar[columna].set(
        self.librosNombres[0])
      self.campos_actualizar[columna].pack(
        side="left", expand=True, fill="x", padx=15)
    if columna == "autor":
      autores = Autores()
      self.autoresNombres = [
        f"{autor['nombre']} {autor['apellido']}" for autor in autores.autores]
      # Lista de opciones para el autor o editorial
      opcionesAutores = self.autoresNombres
      self.campos_actualizar[columna] = ttk.Combobox(
        frame_fila, values=opcionesAutores, font=("Arial", 14, "bold"))
```

```
        self.campos_actualizar[columna].set(
          self.autoresNombres[0])
        self.campos_actualizar[columna].pack(
          side="left", expand=True, fill="x", padx=15)
```

Si es otro título, crea manualmente los campos de entrada para cada columna del self.campo_selected_table.

Llama a la función crear_boton_sub_panel() del módulo btn_config para crear los botones del subpanel.

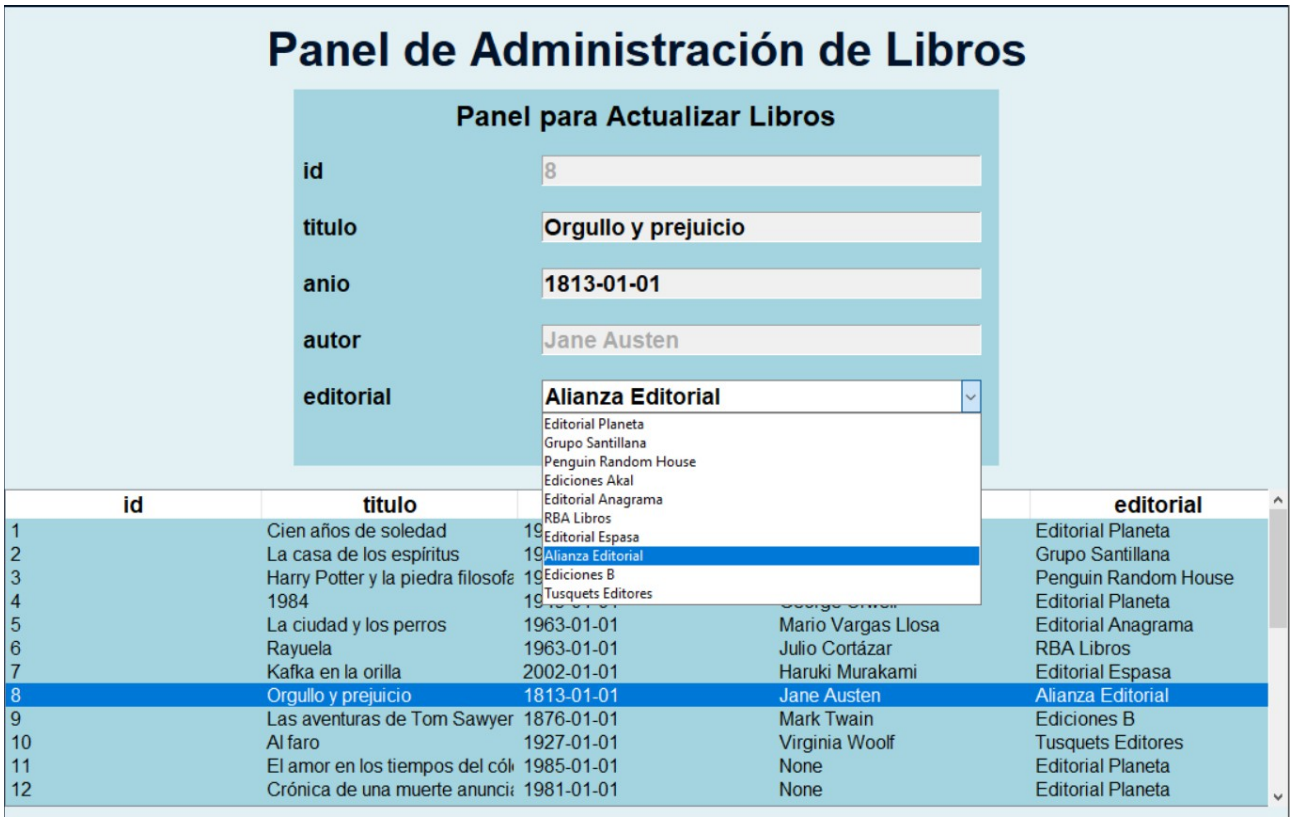
Fuerza el ajuste del panel después de cargar los datos para que no sea visible.
Función actualizar_campos_actualizar(self, campo_selected_table):

Esta función se encarga de actualizar los campos del panel de actualización.
Limpia los campos actuales, estableciendo el estado de cada campo a "normal" y borrando su contenido.

Introduce los datos de campo_selected_table en los campos correspondientes del self.campos_actualizar.
Establece el estado de los campos a "readonly" (de solo lectura) después de introducir los datos.

En resumen, este código define las funciones necesarias para cargar y actualizar los datos en el panel de actualización de la aplicación. La función cargarDatosParaActualizar() se encarga de crear el panel de actualización, mientras que actualizar_campos_actualizar() se encarga de actualizar los campos con los datos seleccionados. Esto permite a la aplicación mostrar y modificar los datos de los módulos "Libros" y "Autor-Libro".


eDe-Lib
¡Bienvenido!
Buscar
Insertar
Eliminar
Actualizar
Panel de Administración de Libros
Panel para Actualizar Libros
id
5
titulo
La ciudad y los perros
anio
1963-01-01
autor
Mario Vargas Llosa
editorial
Editorial Planeta
Actualizar
Inicio
Libros
Autores
Editoriales
Autor-Libro
Crear Tabla
id titulo anio autor editorial
1 Cien años de soledad 1967-01-01 Gabriel García Márquez Editorial Planeta
2 La casa de los espíritus 1982-01-01 Isabel Allende Grupo Santillana
3 Harry Potter y la piedra filosofa 1997-01-01 J.K. Rowling Penguin Random House
4 1984 1949-01-01 George Orwell Editorial Planeta
5 La ciudad y los perros 1963-01-01 Mario Vargas Llosa Editorial Anagrama
6 Rayuela 1963-01-01 Julio Cortázar RBA Libros
7 Kafka en la orilla 2002-01-01 Haruki Murakami Editorial Espasa
8 Orgullo y prejuicio 1813-01-01 Jane Austen Alianza Editorial
9 Las aventuras de Tom Sawyer 1876-01-01 Mark Twain Ediciones B
10 Al faro 1927-01-01 Virginia Woolf Tusquets Editores
11 El amor en los tiempos del cól 1985-01-01 None Editorial Planeta
12 Crónica de una muerte anunci 1981-01-01 None Editorial Planeta


Panel de Administración de Libros
Panel para Actualizar Libros
id
8
titulo
Orgullo y prejuicio
anio
1813-01-01
autor
Jane Austen
editorial
Alianza Editorial
Editorial Planeta
Grupo Santillana
Penguin Random House
Ediciones Akal
Editorial Anagrama
RBA Libros
Editorial Espasa
Alianza Editorial
Ediciones B
Tusquets Editores
id titulo editorial
1 Cien años de soledad Editorial Planeta
2 La casa de los espíritus Grupo Santillana
3 Harry Potter y la piedra filosofa Penguin Random House
4 1984 Editorial Planeta
5 La ciudad y los perros 1963-01-01 Mario Vargas Llosa Editorial Anagrama
6 Rayuela 1963-01-01 Julio Cortázar RBA Libros
7 Kafka en la orilla 2002-01-01 Haruki Murakami Editorial Espasa
8 Orgullo y prejuicio 1813-01-01 Jane Austen Alianza Editorial
9 Las aventuras de Tom Sawyer 1876-01-01 Mark Twain Ediciones B
10 Al faro 1927-01-01 Virginia Woolf Tusquets Editores
11 El amor en los tiempos del cól 1985-01-01 None Editorial Planeta
12 Crónica de una muerte anunci 1981-01-01 None Editorial Planeta

subpanel_insertar

Url: modulos/paneles/subpanel_insertar.py

Este modulo hace lo mismo que actualizar pero se encarga de la insercion, dependiendo del panel en el que estemos, tiene unos parametros predefinidos que ya hemos visto en el modulo datos. `dat_filas=insert_data.datos_llenar_insertar(self,self.titulo_panel_administracion)` del cual se alimenta.

Tambien es Automatizado, dependiendo del panel donde estemos

Importaciones:

tkinter: Importa el módulo principal de Tkinter.
modulos.botones.btn_config: Importa el módulo que contiene la configuración de los botones.
config.config: Importa el módulo de configuración que contiene variables de color.
modulos.datos.datos_para_insertar: Importa el módulo que contiene los datos para insertar.
modulos.paneles.ccp_insertar_libro: Importa el módulo que contiene la función para crear el panel de inserción de libros.
modulos.paneles.ccp_insertar_autorlibro: Importa el módulo que contiene la función para crear el panel de inserción de autor-libro.

Función cargarDatosParaInsertar(self, tipo_boton):

Esta función se encarga de cargar los datos en el panel de inserción.
Crea una etiqueta con el título del panel de inserción.
Inicializa un diccionario self.campos_insertar para almacenar los campos de entrada.
Llama a la función datos_llenar_insertar() del módulo datos_para_insertar para obtener los datos a insertar.

Verifica los datos obtenidos para determinar qué tipo de panel de inserción se debe crear.

Si los datos contienen "titulo", "año", "autor" y "editorial", llama a la función crear_cuerpo_panel_insertar_libros() del módulo ccp_insertar_libro.

ccp_insertar_libro

Url: modulos/paneles/ccp_insertar_libro.py

Este modulo se encarga de la insercion de los datos dentro del panel, se ah separado y modularizado

para que sea mas entendible puesto que hace cosas diferentes e inserta diferentes tipos de objetos y propiedades.

Importaciones:

tkinter: Importa el módulo principal de Tkinter.
tkinter.ttk: Importa el módulo de widgets extendidos de Tkinter.
config.config: Importa el módulo de configuración que contiene variables de color.
clases.autores: Importa la clase Autores que contiene la información de los autores.
clases.editoriales: Importa la clase Editoriales que contiene la información de las editoriales.

Función crear_cuerpo_panel_insertar_libros(self, dat_filas):

Esta función se encarga de crear el cuerpo del panel de inserción de libros.
Inicializa un diccionario self.campos_insertar para almacenar los campos de entrada.
Itera a través de cada columna de los datos dat_filas proporcionados.
Crea un marco frame_fila para contener cada campo de entrada.
Crea una etiqueta con el nombre de la columna y la agrega al marco.
Verifica el nombre de la columna y crea el campo de entrada correspondiente:

Si la columna es "titulo" o "año", crea un campo de entrada de tipo tk.Entry.
Si la columna es "autor" o "editorial", crea un campo de entrada de tipo ttk.Combobox.

Obtiene la lista de autores y editoriales de las clases Autores y Editoriales, respectivamente.
Establece las opciones del combobox con los nombres de los autores y editoriales.
Establece la opción por defecto como el primer elemento de la lista.

```
 self.campos_insertar[columna].set(
          autoresNombres[0])
```

Agrega el campo de entrada creado al diccionario self.campos_insertar y lo empaqueta en el marco.
Empaqueta el marco frame_fila en el panel de acciones.

En resumen, este código define la función crear_cuerpo_panel_insertar_libros() que se encarga de crear los campos de entrada para el panel de inserción de libros. Utiliza los datos proporcionados (dat_filas) para determinar los campos necesarios y crea los widgets correspondientes, ya sea campos de entrada de texto o comboboxes para seleccionar autores y editoriales. Estos campos se almacenan en el diccionario self.campos_insertar para su posterior uso.

```
from tkinter import ttk
import tkinter as tk
import config.config as colores
from clases.autores import Autores
from clases.editoriales import Editoriales



def crear_cuerpo_panel_insertar_libros(self,dat_filas):
  self.campos_insertar = {}
```

```
  for columna in dat_filas:
    frame_fila = tk.Frame(
      self.panel_acciones_cuerpo, bg=colores.COLOR_PANEL_INFO)
    frame_fila.pack(pady=10, fill="x")
    tk.Label(frame_fila, text=columna, width=15,
        anchor="w", font=("Arial", 14, "bold"),
bg=colores.COLOR_PANEL_INFO).pack(side="left", padx=5)
    if columna == "titulo" or columna == "año":
      self.campos_insertar[columna] = tk.Entry(
        frame_fila, font=("Arial", 14, "bold"))
      self.campos_insertar[columna].pack(
        side="left", expand=True, fill="x", padx=15)
    if columna == "autor" or columna == "editorial":
      editoriales = Editoriales()
      autores = Autores()
      editorialesNombre = [
        editorial["nombre"] for editorial in editoriales.editoriales]
      autoresNombres = [
        f"{autor["nombre"]} {autor["apellido"]}" for autor in autores.autores]
      # Lista de opciones para el autor o editorial
      opcionesEditoriales = editorialesNombre
      opcionesAutores = autoresNombres
      # Establece la opción por defecto
      if columna == "autor":
        self.campos_insertar[columna] = ttk.Combobox(
          frame_fila, values=opcionesAutores, font=("Arial", 14, "bold"))
        self.campos_insertar[columna].set(
          autoresNombres[0])
      elif columna == "editorial":
        self.campos_insertar[columna] = ttk.Combobox(
          frame_fila, values=opcionesEditoriales, font=("Arial", 14, "bold"))
        self.campos_insertar[columna].set(
          editorialesNombre[0])
      self.campos_insertar[columna].pack(
        side="left", expand=True, fill="x", padx=15)
```

Si los datos contienen "libro" y "autor", llama a la función crear_cuerpo_panel_insertar_autor_libro() del módulo ccp_insertar_autorlibro.

ccp_insertar_autorlibro

Url: modulos/paneles/ccp_insertar_autorlibro.py

Importaciones:

tkinter: Importa el módulo principal de Tkinter.
tkinter.ttk: Importa el módulo de widgets extendidos de Tkinter.
config.config: Importa el módulo de configuración que contiene variables de color.
clases.autores: Importa la clase Autores que contiene la información de los autores.
clases.libros: Importa la clase Libros que contiene la información de los libros.
Función crear_cuerpo_panel_insertar_autor_libro(self, dat_filas):

Esta función se encarga de crear el cuerpo del panel de inserción de autor-libro.
Inicializa un diccionario self.campos_insertar para almacenar los campos de entrada.

Itera a través de cada columna de los datos dat_filas proporcionados.
Crea un marco frame_fila para contener cada campo de entrada.
Crea una etiqueta con el nombre de la columna y la agrega al marco.
Verifica el nombre de la columna y crea el campo de entrada correspondiente:

Si la columna es "autor", crea un campo de entrada de tipo ttk.Combobox.
Obtiene la lista de autores de la clase Autores.

Establece las opciones del combobox con los nombres completos de los autores.
Establece la opción por defecto como el primer elemento de la lista.

Si la columna es "libro", crea un campo de entrada de tipo ttk.Combobox.
Obtiene la lista de libros de la clase Libros.
Establece las opciones del combobox con los títulos de los libros.
Establece la opción por defecto como el primer elemento de la lista.

Agrega el campo de entrada creado al diccionario self.campos_insertar y lo empaqueta en el marco.
Empaqueta el marco frame_fila en el panel de acciones.

En resumen, este código define la función crear_cuerpo_panel_insertar_autor_libro() que se encarga de crear los campos de entrada para el panel de inserción de autor-libro. Utiliza los datos proporcionados (dat_filas) para determinar los campos necesarios y crea los comboboxes para seleccionar el autor y el libro. Estos campos se almacenan en el diccionario self.campos_insertar para su posterior uso.

```
from tkinter import ttk
import tkinter as tk
import config.config as colores
from clases.autores import Autores
from clases.libros import Libros
```

```
def crear_cuerpo_panel_insertar_autor_libro(self,dat_filas):
  self.campos_insertar = {}
  for columna in dat_filas:
    frame_fila = tk.Frame(
      self.panel_acciones_cuerpo, bg=colores.COLOR_PANEL_INFO)
    frame_fila.pack(pady=10, fill="x")
    tk.Label(frame_fila, text=columna, width=15,
        anchor="w", font=("Arial", 14, "bold"),
bg=colores.COLOR_PANEL_INFO).pack(side="left", padx=5)
    if columna == "autor":
      autores = Autores()
      self.autoresNombres = [
        f"{autor["nombre"]} {autor["apellido"]}" for autor in autores.autores]
      # Lista de opciones para el autor o editorial
      opcionesAutores = self.autoresNombres
      self.campos_insertar[columna] = ttk.Combobox(
        frame_fila, values=opcionesAutores, font=("Arial", 14, "bold"))
      self.campos_insertar[columna].set(
        self.autoresNombres[0])
    if columna == "libro":
      libros = Libros()
      self.librosNombres = [
        libro["titulo"] for libro in libros.libros]
      # Lista de opciones para el autor o editorial
      opcionesLibros = self.librosNombres
      self.campos_insertar[columna] = ttk.Combobox(
        frame_fila, values=opcionesLibros, font=("Arial", 14, "bold"))
      self.campos_insertar[columna].set(
        self.librosNombres[0])
    self.campos_insertar[columna].pack(
      side="left", expand=True, fill="x", padx=15)
```

Si los datos no encajan en los casos anteriores, crea manualmente los campos de entrada para cada columna de los datos.
Llama a la función crear_boton_sub_panel() del módulo btn_config para crear los botones del subpanel.
Fuerza el ajuste del panel después de cargar los datos para que no sea visible.

En resumen, este código define la función cargarDatosParaInsertar() que se encarga de crear el panel de inserción de la aplicación. Dependiendo de los datos obtenidos, llama a las funciones correspondientes de los módulos ccp_insertar_libro y ccp_insertar_autorlibro para crear el panel de inserción adecuado. Además, crea los botones del subpanel y ajusta la visibilidad del panel después de cargar los datos.

```
import tkinter as tk
import modulos.botones.btn_config as btn_config
import config.config as colores
import modulos.datos.datos_para_insertar as insert_data
import modulos.paneles.ccp_insertar_libro as insertar_libro
import modulos.paneles.ccp_insertar_autorlibro as insertar_autorlibro


def cargarDatosParaInsertar(self, tipo_boton):
  #titulo del panel
  tk.Label(self.panel_acciones_cuerpo, text=f"Panel para Insertar
{self.titulo_panel_administracion}", bg=colores.COLOR_PANEL_INFO, font=("Arial",
18, "bold")).pack(pady=5)
  self.campos_insertar = {}

  #buscar los datos a insertar en el panel y aplicar la condicion dependiendo de
los datos
  #para crear la estructura del tipo de panel
  dat_filas=insert_data.datos_llenar_insertar(self,
self.titulo_panel_administracion)

  if "titulo" and "año" and "autor" and "editorial" in dat_filas:
    insertar_libro.crear_cuerpo_panel_insertar_libros(self,dat_filas)

  elif "libro" and "autor" in dat_filas:
    insertar_autorlibro.crear_cuerpo_panel_insertar_autor_libro(self,dat_filas)

  else:
    for columna in dat_filas:

      frame_fila = tk.Frame(
        self.panel_acciones_cuerpo, bg=colores.COLOR_PANEL_INFO)
      frame_fila.pack(pady=10, fill="x")

      tk.Label(frame_fila, text=columna, width=15,
          anchor="w", font=("Arial", 14, "bold"),
bg=colores.COLOR_PANEL_INFO).pack(side="left", padx=5)

      self.campos_insertar[columna] = tk.Entry(
          frame_fila, font=("Arial", 14, "bold"))
      self.campos_insertar[columna].pack(
          side="left", expand=True, fill="x", padx=15)


  #este metodo crea los botones automaticos dependiendo del panel
  btn_config.crear_boton_sub_panel(self, tipo_boton)

  # Forzar el ajuste del panel después de cargar los datos para que no sea visible
  self.panel_cuerpo.after(10, lambda: self.panel_acciones_cuerpo.place(y=-400))
```

subpanel_buscar

Url: modulos/paneles/subpanel_buscar.py

Este panel se encaraga de buscar un elemento directamente en la base de datos .

Importaciones

```
import tkinter as ttk
import tkinter as tk
import config.config as colores
import modulos.paneles.tabla as tabla
import modulos.botones.btn_hover as btn_hover
from clases.libros import Libros
from clases.autores import Autores
from clases.editoriales import Editoriales
from clases.autorlibro import AutorLibro
```

tkinter: Se usa para crear la interfaz gráfica.
config.config: Contiene configuraciones de colores.
modulos: Importa módulos para manejar tablas y botones.
clases: Importa clases que representan libros, autores, editoriales y la relación entre autores y libros.

Función creacion_penel_buscar
Esta función crea un panel de búsqueda en la interfaz.

Componentes:
Etiqueta de Título:

```
tk.Label(self.panel_acciones_cuerpo, text=f"Buscador de {self.titulo_panel_administracion}",
        bg=colores.COLOR_PANEL_INFO, font=("Arial", 20, "bold"),
fg="darkblue").pack(pady=15)
```

Muestra el título del panel de administración.
Icono de Lupa:

```
self.label_lupa = tk.Label(self.panel_acciones_cuerpo, image=self.lupa,
                           bg=colores.COLOR_PANEL_INFO, padx=10, pady=10)
```

Muestra una imagen de lupa.
Checkboxes:

```
crear_checkboxes(self, self.titulo_panel_administracion)
```

Llama a la función crear_checkboxes para generar opciones dinámicamente según el tipo de panel.
Campo de Búsqueda:

```
self.campo_busqueda = tk.Entry(frame_fila, font=("Arial", 14, "bold"))
```

Campo donde el usuario puede ingresar el texto a buscar.
Botón de Buscar:

```
boton = tk.Button(
    self.panel_acciones_cuerpo,
    text="Buscar",
    padx=20,
    bg=colores.COLOR_BTN,
    font=("Arial", 12, "bold"),
    fg="white",
    command= lambda: recolectar_datos_busqueda(self)
)
```

Al hacer clic, se llama a la función recolectar_datos_busqueda.
Ajuste del Panel:

self.panel_cuerpo.after(10, lambda: self.panel_acciones_cuerpo.place(y=-400))
Ajusta la posición del panel después de cargar los datos.
Función crear_checkboxes
Genera checkboxes basados en el tipo de panel administrado.

Componentes:
Opciones de Checkbox: Un diccionario que define qué checkboxes se deben crear según el tipo de panel.
Variables de Checkbox: Se utilizan IntVar para almacenar el estado (marcado/desmarcado) de cada checkbox.
Función obtener_valores
Recoge y devuelve los valores actuales de los checkboxes.

Función recolectar_datos_busqueda
Recoge los valores de los checkboxes y determina qué campos se deben buscar. Si no hay campos seleccionados, se establece un valor predeterminado según el tipo de panel.

Función busqueda_baseDatos
Realiza la búsqueda en la base de datos según los campos seleccionados y el texto ingresado en el campo de búsqueda.

Función busqueda_libros
Filtra los resultados de libros según el campo seleccionado y el texto de búsqueda.

Resumen
Este código implementa una interfaz gráfica para buscar libros, autores y editoriales, utilizando checkboxes para seleccionar qué campos buscar y un campo de entrada para el texto de búsqueda.

```
import tkinter as ttk
import tkinter as tk
import config.config as colores
import modulos.paneles.tabla as tabla
import modulos.botones.btn_hover as btn_hover
from clases.libros import Libros
from clases.autores import Autores
from clases.editoriales import Editoriales
from clases.autorlibro import AutorLibro
```

```
def creacion_penel_buscar(self, tipo_boton):
  tk.Label(self.panel_acciones_cuerpo, text=f"Buscador de
{self.titulo_panel_administracion}",
       bg=colores.COLOR_PANEL_INFO, font=("Arial", 20, "bold"),
fg="darkblue").pack(pady=15)

  self.label_lupa = tk.Label(self.panel_acciones_cuerpo, image=self.lupa,
                bg=colores.COLOR_PANEL_INFO, padx=10, pady=10)
  self.label_lupa.place(x=500, y=80, anchor="center",
            width=self.lupa.width(), height=self.lupa.height())
  self.label_lupa.lift()

  self.checkbox_vars=[]
  # Este método crea los checkboxes dependiendo del panel
  crear_checkboxes(self, self.titulo_panel_administracion)


  # Frame para el campo de búsqueda
  frame_fila = tk.Frame(self.panel_acciones_cuerpo, bg=colores.COLOR_PANEL_INFO)
  frame_fila.pack(pady=30, fill="x")

  tk.Label(frame_fila, text="Texto a Buscar", width=12, anchor="w",
       font=("Arial", 14, "bold"), bg=colores.COLOR_PANEL_INFO).pack(side="left",
padx=5)

  self.campo_busqueda = tk.Entry(frame_fila, font=("Arial", 14, "bold"))
  self.campo_busqueda.pack(side="left", expand=True, fill="x", padx=20)

  # Este método crea los botones automáticos dependiendo del panel
  boton = tk.Button(
    self.panel_acciones_cuerpo,
    text="Buscar",
    padx=20,
    bg=colores.COLOR_BTN,
    font=("Arial", 12, "bold"),
    fg="white",
    command= lambda: recolectar_datos_busqueda(self) # Aquí está la corrección
  )

  # Empaquetar el botón
  boton.pack(side="right", padx=20)
  btn_hover.hover_event(self, boton)

  # Forzar el ajuste del panel después de cargar los datos para que no sea visible
  self.panel_cuerpo.after(10, lambda: self.panel_acciones_cuerpo.place(y=-400))



# Este método crea los checkboxes dependiendo del panel
def crear_checkboxes(self, tipo):
  opciones = {
```

```
    "Libros": ["id", "titulo", "año", "autor", "editorial"],
    "Autores": ["id", "nombre", "apellido", "nacionalidad"],
    "Editoriales": ["id", "nombre", "direccion", "telefono"],
    "Autor-Libro": ["nombre_del_autor", "titulo_del_libro"]
  }

  if tipo in opciones:
    # Frame para los primeros dos checkboxes
    frame_check1 = tk.Frame(self.panel_acciones_cuerpo,
bg=colores.COLOR_PANEL_INFO)
    frame_check1.pack(pady=10)

    # Frame para los últimos checkboxes
    frame_check2 = tk.Frame(self.panel_acciones_cuerpo,
bg=colores.COLOR_PANEL_INFO)
    frame_check2.pack(pady=15, fill="x")

    # Diccionario para almacenar variables de checkboxes
    self.checkbox_vars = {}

    for i, opcion in enumerate(opciones[tipo]):
      var = tk.IntVar()  # Crear una variable para el checkbox
      checkbox = tk.Checkbutton(frame_check1 if i < 2 else frame_check2,
text=opcion, variable=var,
                    bg=colores.COLOR_PANEL_INFO, font=("Arial", 12,
"bold"),highlightbackground=colores.COLOR_PANEL_INFO,
                    activebackground=colores.COLOR_PANEL_INFO)
      checkbox.pack(side="left", padx=10)  # Empaquetar a la izquierda

      # Guardar la variable en el diccionario con el nombre de la opción
      self.checkbox_vars[opcion] = var

def obtener_valores(self):
  # Recoger y mostrar los valores de los checkboxes
  valores = {nombre: var.get() for nombre, var in self.checkbox_vars.items()}
  return valores


def recolectar_datos_busqueda(self):
  # Obtiene un diccionario con los valores de los checkboxes
  valores= obtener_valores(self)
  campos = []

  for campo, valor in valores.items():
    if valor == 1:
      campos.append(campo)

  # Cambiar "anio" a "anios" si está presente en los campos
  if "año" in campos:
    campos[campos.index("año")] = "anio"
```

```
  # Si no hay campos seleccionados, realizar la búsqueda por esto
  if not campos and self.titulo_panel_administracion == "Libros":
    campos = ["id", "titulo", "anio"]
  elif not campos and self.titulo_panel_administracion == "Autores":
    campos = ["id", "nombre", "apellido", "nacionalidad"]
  elif not campos and self.titulo_panel_administracion == "Editoriales":
    campos = ["id", "nombre","direccion","telefono"]
  elif not campos and self.titulo_panel_administracion == "AutorLibro":
    campos = ["id_libro", "id_autor"]

  # Actualizar la tabla con los resultados de la búsqueda
  busqueda_baseDatos(self, campos)


def busqueda_baseDatos(self , campos):
  busqueda = self.campo_busqueda.get()
  self.reg_busqueda=[]
  if self.titulo_panel_administracion == "Libros":
    self.reg_busqueda=busqueda_libros(self, campos, busqueda)
  elif self.titulo_panel_administracion == "Autores":
    autores = Autores()
    self.reg_busqueda=autores.filtrar(campos,busqueda)
  elif self.titulo_panel_administracion == "Editoriales":
    editoriales = Editoriales()
    self.reg_busqueda= editoriales.filtrar(campos,busqueda)
  # elif self.titulo_panel_administracion == "AutorLibro":
  #     autorlibro= AutorLibro()
  #     resultados = autorlibro.filtrar(campos,busqueda)
  #     self.busqueda=resultados

  tabla.actualizar_tabla(self, self.reg_busqueda)
  self.campo_selected_table = {}

def busqueda_libros(self, campos, busqueda):
  libros = Libros()
  if campos == ["editorial"]:
    resultados = libros.filtrar_libros_por_editorial(busqueda)
    return resultados
  elif campos == ["autor"]:
    resultados = libros.filtrar_libros_por_autor(busqueda)
    return resultados
  else:
    resultados = libros.filtrar_libros(campos, busqueda)
    if not resultados:
      resultados = libros.filtrar_libros_por_autor(busqueda)
      if not resultados:
        resultados = libros.filtrar_libros_por_editorial(busqueda)

    return resultados
```

crear_tabla_bd

Url: modulos/paneles/crear_tabla_bd.py

Este es un modulo el cual se encargaria de crear tablas en la base de datos, está diseñado para crear una interfaz gráfica de usuario (GUI) utilizando tkinter, que permite al usuario ingresar información para crear una nueva tabla en una base de datos.

1. Importaciones y Configuración Inicial

```
import tkinter as tk
from tkinter import ttk
from clases.crear_tabla import Crear
import config.config as colores
import util.util_ventana as util_ventana
import modulos.botones.btn_hover as btn_hover
```

tkinter: Biblioteca principal para crear interfaces gráficas.
ttk: Módulo de tkinter para widgets temáticos.

Importaciones adicionales: Se importan clases y módulos personalizados para la creación de tablas y configuración de colores.

2. Función nueva_tabla_Base_Datos
Esta función es el núcleo de la interfaz. Se encarga de configurar el panel donde se ingresarán los datos de la nueva tabla.

Panel de Datos: Se oculta el panel anterior y se crea un nuevo panel para la entrada de datos.
Ajuste del Panel: La función ajustar_panel() se utiliza para centrar el panel en la ventana.
Etiquetas y Entradas: Se crean etiquetas y campos de entrada para que el usuario ingrese el nombre de la tabla.
3. Agregar Campos Dinámicamente

```
self.boton_agregar_campo = tk.Button(self.primeraFila,
                                     text="Agregar Campo",
                                     command=lambda: agregar_campo(self),
                                     bg=colores.COLOR_BARRA_SUPERIOR, fg="white", font=("Arial", 12,
"bold"))
```

Botón "Agregar Campo": Permite al usuario añadir campos adicionales a la tabla. Cada vez que se presiona, se llama a la función agregar_campo.
4. Función agregar_campo

```
def agregar_campo(self):
    crear_campo(self, "Nombre:", tk.Entry)
    crear_campo(self, "Tipo:", ttk.Combobox, ["INT", "VARCHAR", "TEXT", "DATE"])
    crear_campo(self, "Predeterminado:", ttk.Combobox, ["NULL", "NOT NULL"])
    crear_campo(self, "A/I:", tk.Checkbutton, None, variable=self.chekked)
    crear_campo(self, "Indice:", ttk.Combobox, ["NONE", "PRIMARY KEY", "FOREIGN KEY"])
```

Esta función crea varios campos de entrada, permitiendo al usuario especificar diferentes atributos para la tabla.

5. Función crear_campo
Esta función se encarga de crear cada campo individualmente, incluyendo etiquetas y widgets:

```
def crear_campo(self, label_text, widget_type, values=None, **kwargs):
    ...
```

Widgets: Dependiendo del tipo de widget (Entrada, Combobox, Checkbutton), se crea y se coloca en el panel.
6. Botón para Imprimir Datos

```
self.boton_imprimir = tk.Button(self.panel_cuerpo_tabla,
                                text="Imprimir Datos",
                                command=lambda: imprimir_datos(self),
                                bg=colores.COLOR_BARRA_SUPERIOR, fg="white", font=("Arial", 12,
"bold"))
```

Función imprimir_datos: Recoge todos los datos introducidos por el usuario y los imprime en la consola.
7. Función imprimir_datos

```
def imprimir_datos(self):
    datos = []
    # Recoger el nombre de la tabla
    nombre_tabla = self.input_nombre_tabla.get()
    datos.append(f"Nombre de la Tabla: {nombre_tabla}")

    # Recoger los datos de los campos
    for widget in self.sub_cuerpo_campos.winfo_children():
        ...
    print("\n".join(datos))
```

Esta función recopila los datos de todos los campos y los imprime en la consola para que el usuario pueda ver lo que ha ingresado.

Resumen
El código proporciona una interfaz gráfica para que los usuarios ingresen datos necesarios para crear una nueva tabla en una base de datos. Utiliza tkinter para construir la GUI y permite agregar campos dinámicamente. Además, incluye un botón para imprimir todos los datos ingresados, facilitando la revisión de la información antes de proceder con la creación de la tabla.

Características Clave:
Interfaz Gráfica: Utiliza tkinter para crear una GUI interactiva.
Campos Dinámicos: Permite agregar múltiples campos de entrada según sea necesario.
Impresión de Datos: Facilita la impresión de todos los datos ingresados en la consola.
Si necesitas más detalles sobre alguna sección específica o cualquier otra consulta, ¡házmelo saber!

```
import tkinter as tk
from tkinter import ttk
from clases.crear_tabla import Crear
import config.config as colores
import util.util_ventana as util_ventana
import modulos.botones.btn_hover as btn_hover

def nueva_tabla_Base_Datos(self):
  if self.titulo_panel_administracion == "Bienvenido a eDe-Lib":
    self.panel_inicio.pack_forget()
  else:
    self.panel_datos.pack_forget()

  self.panel_nueva_tabla = tk.Frame(self.cuerpo_principal,
bg=colores.COLOR_CUERPO_PRINCIPAL)
  self.panel_nueva_tabla.place(relwidth=1, relheight=1)

  self.titulo_panel_administracion = "Nueva Tabla en la Base de Datos"

  ancho = 1024
  alto = 600
  self.panel_cuerpo_tabla = tk.Frame(self.panel_nueva_tabla,
bg=colores.COLOR_PANEL_INFO)

  def ajustar_panel():
    x, y = util_ventana.centrar_panel(self.panel_nueva_tabla, ancho, alto)
    self.panel_cuerpo_tabla.place(width=ancho, height=alto, x=x, y=y)
    self.coordenadas = [x, y, ancho, alto]

  tk.Label(self.panel_cuerpo_tabla, text="Para Crear una nueva Tabla en la Base de
Datos, por favor ingrese los datos solicitados",
       bg=colores.COLOR_PANEL_INFO, fg=colores.COLOR_BARRA_SUPERIOR, font=("Arial",
16, "bold")).place(relx=0.5, y=10, anchor="n")

  self.primeraFila = tk.Frame(self.panel_cuerpo_tabla, bg=colores.COLOR_PANEL_INFO)
  self.primeraFila.place(relx=0.5, y=70, anchor="n")

  tk.Label(self.primeraFila, text="Ingrese el Nombre de la Tabla",
       bg=colores.COLOR_PANEL_INFO,
       fg=colores.COLOR_BARRA_SUPERIOR,
       font=("Arial", 12, "bold")).pack(side="left", padx=10)

  self.input_nombre_tabla = tk.Entry(self.primeraFila, bg=colores.COLOR_PANEL_INFO,
fg=colores.COLOR_BARRA_SUPERIOR, font=("Arial", 12, "bold"))
  self.input_nombre_tabla.pack(side="left", padx=5)

  self.boton_agregar_campo = tk.Button(self.primeraFila,
                    text="Agregar Campo",
                    command=lambda: agregar_campo(self),
                    bg=colores.COLOR_BARRA_SUPERIOR, fg="white", font=("Arial",
12, "bold"))
```

```
  self.boton_agregar_campo.pack(side="left", padx=30)

  self.cuerpo_campos = tk.Frame(self.panel_cuerpo_tabla,
bg=colores.COLOR_PANEL_INFO)
  self.cuerpo_campos.place(relx=0.5, y=120, anchor="n")
  self.sub_cuerpo_campos = tk.Frame(self.cuerpo_campos,
bg=colores.COLOR_PANEL_INFO)
  self.sub_cuerpo_campos.grid(row=0, column=0, columnspan=5, padx=10, pady=5)

  # Botón para imprimir datos
  self.boton_imprimir = tk.Button(self.panel_cuerpo_tabla,
                  text="Imprimir Datos",
                  command=lambda: imprimir_datos(self),
                  bg=colores.COLOR_BARRA_SUPERIOR, fg="white", font=("Arial", 12,
"bold"))
  self.boton_imprimir.place(relx=0.5, y=400, anchor="center")

  self.chekked = tk.IntVar()
  self.contador_filas = 0
  self.contador_columnas = 0

  self.panel_nueva_tabla.bind("<Configure>", lambda event: ajustar_panel())
  ajustar_panel()
  btn_hover.hover_event(self, self.boton_agregar_campo)

def agregar_campo(self):
  crear_campo(self, "Nombre:", tk.Entry)
  crear_campo(self, "Tipo:", ttk.Combobox, ["INT", "VARCHAR", "TEXT", "DATE"])
  crear_campo(self, "Predeterminado:", ttk.Combobox, ["NULL", "NOT NULL"])
  crear_campo(self, "A/I:", tk.Checkbutton, None, variable=self.chekked)
  crear_campo(self, "Indice:", ttk.Combobox, ["NONE", "PRIMARY KEY", "FOREIGN
KEY"])

def crear_campo(self, label_text, widget_type, values=None, **kwargs):
  frame_campo = tk.Frame(self.sub_cuerpo_campos, bg=colores.COLOR_PANEL_INFO)
  frame_campo.grid(row=self.contador_filas, column=self.contador_columnas, padx=10,
pady=5, sticky="w")

  self.contador_columnas += 1
  if self.contador_columnas >= 5:
    self.contador_columnas = 0
    self.contador_filas += 1

  tk.Label(frame_campo, text=label_text, bg=colores.COLOR_PANEL_INFO,
fg=colores.COLOR_BARRA_SUPERIOR, font=("Arial", 12, "bold")).pack(side=tk.TOP)

  if widget_type == tk.Entry:
    widget = widget_type(frame_campo, **kwargs, fg=colores.COLOR_BARRA_SUPERIOR,
font=("Arial", 12, "bold"))
  elif widget_type == ttk.Combobox:
    widget = widget_type(frame_campo, values=values, **kwargs,
```

```
background=colores.COLOR_PANEL_INFO, foreground=colores.COLOR_BARRA_SUPERIOR,
font=("Arial", 12, "bold"))
  elif widget_type == tk.Checkbutton:
    widget = widget_type(frame_campo, **kwargs, bg=colores.COLOR_PANEL_INFO,
fg=colores.COLOR_BARRA_SUPERIOR, font=("Arial", 12, "bold"))

  widget.pack(side=tk.BOTTOM)

def imprimir_datos(self):
  datos = []
  # Recoger el nombre de la tabla
  nombre_tabla = self.input_nombre_tabla.get()
  datos.append(f"Nombre de la Tabla: {nombre_tabla}")

  # Recoger los datos de los campos
  for widget in self.sub_cuerpo_campos.winfo_children():
    if isinstance(widget, tk.Entry):
      datos.append(f"Campo: {widget.get()}")
    elif isinstance(widget, ttk.Combobox):
      datos.append(f"Campo: {widget.get()}")
    elif isinstance(widget, tk.Checkbutton):
      estado = "Activado" if widget.var.get() else "Desactivado"
      datos.append(f"Campo A/I: {estado}")

  # Imprimir los datos en la consola
  print("\n".join(datos))
```