Pestañas

Como crearlas

`self.tab2 = self.tabview.add("Solicitud de Libros")`:

- **Descripción**: Esta línea agrega una nueva pestaña (tab) a un control de vista de pestañas (`tabview`). La pestaña se llama "Solicitud de Libros". El método `add` crea la pestaña y la asocia con `self.tab2`, permitiendo que puedas referenciarla más adelante en el código.

Como crear su configuración

`self.configurar_pestana2()`:

- **Descripción**: Esta línea llama a un método llamado `configurar_pestana2`, que contiene la configuración o inicialización de la pestaña que acabas de agregar (`self.tab2`). Esto puede incluir la adición de widgets, la configuración de su apariencia o cualquier otro tipo de personalización específica para esa pestaña.

Como agregamos imágenes a los Libros

```
def __init__(self, nombre, ingredientes, icono_menu=None,precio=0):
```

En su constructor agregaremos la propiedad icono_menu inicializado en None para luego pasar una imagen a este.

Importando `from PIL import Image`

```
def cargar_icono_libro(self, ruta_icono):
            imagen = Image.open(ruta_icono)
            icono_Libro = ctk.CTkImage(imagen, size=(64, 64))
            return icono_Libro
```

Crearemos las tarjetas usando esta misma función para dar una imagen a cada libro usando su atributo

```
def crear_tarjeta(self, libro):
        tarjeta = ctk.CTkFrame(tarjetas_frame)
        tarjeta.pack(padx=15, pady=15, fill="x")

        # Hacer que la tarjeta sea completamente clickeable
        tarjeta.bind("<Button-1>", lambda event: self.tarjeta_click(event, libro))

        # Cambiar el color del borde cuando el mouse pasa sobre la tarjeta
        tarjeta.bind("<Enter>", lambda event: tarjeta.configure(border_color="#FF0000"))  # Cambia a
rojo al pasar el mouse
        tarjeta.bind("<Leave>", lambda event: tarjeta.configure(border_color="#4CAF50"))  # Vuelve
al verde al salir

        # Añadir la imagen del libro (icono_Libro) en la tarjeta
        if libro.icono_Libro:
            icono_label = ctk.CTkLabel(tarjeta, image=libro.icono_Libro, text="",
bg_color="transparent")
            icono_label.pack(side="left", padx=10)
            icono_label.bind("<Button-1>", lambda event: self.tarjeta_click(event, libro))  #
Asegura que el clic en la imagen funcione

        # Añadir información del libro en la tarjeta
        info_label = ctk.CTkLabel(tarjeta, text=str(libro), text_color="black", font=("Helvetica",
12, "bold"), bg_color="transparent")
        info_label.pack(side="left", padx=10)
        info_label.bind("<Button-1>", lambda event: self.tarjeta_click(event, libro))  # Asegura que
el clic en el texto funcione
```

Hacemos que la tarjeta tenga un comportamiento al hacer un click.

```
def tarjeta_click(self, event, libro):
        if libro.cantidad > 0:
            libro.cantidad -= 1
            self.agregar_libro_pedido(libro)
            self.actualizar_treeview_pedido()
            total_libros = self.calcular_total_libros()
        else:
            CTkMessagebox(title="Stock Insuficiente", message=f"No hay suficientes
copias de '{libro.nombre}' disponibles.")
```

Actualizamos las tarjetas para mostrarlas.

```
def actualizar_tarjetas_libros(self):
        for widget in tarjetas_frame.winfo_children():
            widget.destroy()  # Limpiar tarjetas existentes

        libros = self.biblioteca.obtener_libros()
        for libro in libros:
            self.crear_tarjeta(libro)
```

Mostramos las tarjetas en el treview.

```
def actualizar_treeview_pedido(self):
        self.treeview_pedidos.delete(*self.treeview_pedidos.get_children())  # Limpiar el Treeview
antes de actualizar

        for libro in self.pedido:
            # Encontrar el libro existente en el pedido
            for item in self.treeview_pedidos.get_children():
                if self.treeview_pedidos.item(item)['values'][0] == libro.nombre:
                    # Actualizar la cantidad en el pedido existente
                    cantidad_actual = int(self.treeview_pedidos.item(item)['values'][3])
                    nueva_cantidad = cantidad_actual + 1
                    self.treeview_pedidos.item(item, values=(libro.nombre, libro.autor,
libro.genero, nueva_cantidad))
                    break
            else:
                # Agregar el libro al Treeview si no existe en el pedido
                self.treeview_pedidos.insert("", "end", values=(libro.nombre, libro.autor,
libro.genero, 1))
```

Como crear un pdf con estos datos

```
from fpdf import FPDF
from datetime import datetime

class ListaFacade:
    def __init__(self, pedido):
        self.pedido = pedido
        self.detalle = ""

    def generar_detalle_libros(self):
        """Genera el detalle del listado de libros como una cadena de texto y
lo almacena en self.detalle."""
        self.detalle = ""
        for libro in self.pedido:
            self.detalle += f"{libro.nombre:<30} {libro.autor:<20}
{libro.genero:<15} {libro.cantidad:<10}\n"
```

```
    def crear_pdf(self):
        """Utiliza el detalle generado para crear un archivo PDF."""
        pdf = FPDF()
        pdf.add_page()
        pdf.set_font("Arial", size=12)

        # Encabezado
        pdf.set_font("Arial", 'B', 16)
        pdf.cell(0, 10, "Listado de Libros Pedidos", ln=True, align='C')
        pdf.set_font("Arial", size=12)
        pdf.cell(0, 10, "Biblioteca X", ln=True, align='L')
        pdf.cell(0, 10, f"Fecha: {datetime.now().strftime('%d/%m/%Y
%H:%M:%S')}", ln=True, align='R')
        pdf.ln(10)

        # Encabezado de tabla
        pdf.set_font("Arial", 'B', 12)
        pdf.cell(70, 10, "Nombre", border=1)
        pdf.cell(50, 10, "Autor", border=1)
        pdf.cell(30, 10, "Categoría", border=1)
        pdf.cell(20, 10, "Cantidad", border=1)
        pdf.ln()

        # Detalle de la tabla
        pdf.set_font("Arial", size=12)
        for libro in self.pedido:
            pdf.cell(70, 10, libro.nombre, border=1)
            pdf.cell(50, 10, libro.autor, border=1)
            pdf.cell(30, 10, libro.genero, border=1)
            pdf.cell(20, 10, str(libro.cantidad), border=1)
            pdf.ln()

        # Pie de página
        pdf.set_font("Arial", 'I', 10)
        pdf.cell(0, 10, "Gracias por utilizar nuestros servicios.", 0, 1, 'C')
```

```
        # Guardar el archivo PDF
        pdf_filename = "listado_libros_pedidos.pdf"
        pdf.output(pdf_filename)
        return pdf_filename

    def generar_listado_libros(self):
        """Coordina la generación del listado de libros y la creación del
PDF."""
        self.generar_detalle_libros()
        pdf_path = self.crear_pdf()
        return f"Listado de libros generado y guardado en: {pdf_path}"
```

Luego importamos esta clase en la interfaz

```
from Lista_ejemplo import ListaFacade
```

y la llamamos en generar_pedido

```
def generar_pedido(self):
        rutas_iconos = ["IMG/icono_papas_fritas_64x64.png"]
        for widget in tarjetas_frame.winfo_children():
            widget.destroy()

        libros = self.biblioteca.obtener_libros()
        for libro in libros:
            if libro.cantidad > 0:
                libro.icono_Libro =
self.cargar_icono_libro(ra.choice(rutas_iconos))
                self.crear_tarjeta(libro)

        lista_facade = ListaFacade(self.pedido)

        # Generar el PDF
```

```
mensaje = lista_facade.generar_listado_libros()
```