



**FACULTAD DE INGENIERÍA CARRERA DE INGENIERÍA  
INFORMÁTICA FACULTAD COMUNITARIA DE CAACUPÉ**

**TEMA**

**APLICACIÓN EN FLET Y MONGODB**

**TÍTULO**

**CREACIÓN DE CRUD USANDO PYTHON COMO LENGUAJE DE  
PROGRAMACIÓN CON EL FRAMEWORK FLET Y MONGODB  
COMO BASE DE DATOS**

**EXAMEN FINAL**

**AUTORES**

**CARLOS SAUL GIRETT HERMOSILLA**

**TUTOR**

**PROF. ING. RICARDO MAIDANA**

**Caacupé – Paraguay**

**2024**

# 1. Base de datos MongoDB.

## Colección Usuarios.

crud-mongo.usuarios

121DOCUMENTSINDEXES

DocumentsAggregationsSchemaIndexesValidation

FilterType a query: { field: 'value' } or [Generate query](#) ExplainResetFindOptions

ADD DATAEXPORT DATAUPDATEDELETE1 - 12 of 12

#	usuarios	_id ObjectId	nombre String	clave String	email String	estado String	i
1	ObjectId('672398b2149d7b7...	"carlos"	"1234"	"carlossaulgirett29@gmail...	"True"		
2	ObjectId('6723c375149d7b7...	"Cristina"	"cristina"	"ccontreras@gmail.com"	"True"		
3	ObjectId('6723c3aa149d7b7...	"Gustavo"	"Gcardozo"	"gustavocardozo23@gmail.c...	"False"		
4	ObjectId('6723c3b8149d7b7...	"Jessica"	"JsC"	"jessicacoutlook.com"	"True"		
5	ObjectId('6723e469149d7b7...	"Benjamin"	"benja1234"	"bgaleanooutlook.es"	"True"		
6	ObjectId('6723e512149d7b7...	"Jose"	"jj2024"	"josevillan@icloud.com"	"False"		
7	ObjectId('6723e545149d7b7...	"Juan"	"JJ2323"	"juanr"	"True"		
8	ObjectId('6723e59b149d7b7...	"Nando"	"NNunez23"	"nandonunez23@hotmail.com"	"True"		
9	ObjectId('6723e5c2149d7b7...	"Amparo"	"ampv2312"	"amparooutlook.es"	"False"		
10	ObjectId('6723e5e6149d7b7...	"Jennifer"	"jjarami231"	"jenniferarami@outlook.es"	"True"		
11	ObjectId('672628d8affff85...	"Ramona"	"RR2024"	"ramonahermosilla"	"True"		
12	ObjectId('67268c33debab2f...	"erika"	"1234"	"erikab@gmail.com"	"true"		

## Colección Productos.

crud-mongo.productos

31DOCUMENTSINDEXES

DocumentsAggregationsSchemaIndexesValidation

FilterType a query: { field: 'value' } or Generate query ExplainResetFindOptions

ADD DATAEXPORT DATAUPDATEDELETE1 - 3 of 3

#	productos	_id ObjectId	nombre String	marca String	Precio_compra String	precio_venta String	i
1	ObjectId('67548489ea9af01...	"Aqua de Gio 100 ml"	"Armani"	"702.800"	"998.900"		
2	ObjectId('6754855bea9af01...	"protector solar 120ml"	"La Roche Posay"	"89.250"	"210.900"		
3	ObjectId('6754858aea9af01...	"Polo blue rahl Lauren ed...	"Ralph Lauren"	"889.750"	"1.336.000"		

## Colección Clientes

crud-mongo.clientes

91DOCUMENTSINDEXES

DocumentsAggregationsSchemaIndexesValidation

FilterType a query: { field: 'value' } or [Generate query](#) ExplainResetFindOptions

ADD DATAEXPORT DATAUPDATEDELETE

1 - 9 of 9

#	clientes	_id ObjectId	nombre String	apellido String	ruc String	direccion String	i
1	ObjectId('67535bdeea9af01...	"Juan"	"Pérez"	"2343564-0"	"Caacupe"		
2	ObjectId('67535c57ea9af01...	"Dionicio"	"Pérez"	"3453123-9"	"Asuncion"		
3	ObjectId('67535c8fea9af01...	"Marcela"	"Alcaraz"	"67652323"	"Eusebio Ayala"		
4	ObjectId('675373bca6b7d2b...	"Lile"	"Cartman"	"3987990"	"Isla Pucu"		
5	ObjectId('6756ee1a5873a8c...	"Robert"	"Baratheon"	"8997786"	"Caacupe"		
6	ObjectId('6756ee3f5873a8c...	"Carlos"	"Girett"	"8765456"	"San Jose Obrero"		
7	ObjectId('6756ee9b5873a8c...	"Melissa"	"Mendez"	"878670"	"Asuncion"		
8	ObjectId('6756ee55873a8c...	"Luis"	"Cabañas"	"2342178-9"	"Isla Pucu"		
9	ObjectId('6756eefa5873a8c...	"Ramon"	"Careaga"	"2457897-8"	"Mbokajaty"		

## 2. Conexión a MongoDB.

```
import pymongo

class MongoDBConnection:
    def __init__(self, uri="mongodb://localhost:27017/", db_name="crud-mongo"):
        self.cliente = pymongo.MongoClient(uri)
        self.bd = self.cliente[db_name]

    def get_collection(self, collection_name):
        """Devuelve una referencia a la colección especificada."""
        return self.bd[collection_name]

    def get_all_documents(self, collection_name):
        """Devuelve todos los documentos de una colección como lista de diccionarios."""
        coleccion = self.get_collection(collection_name)
        return list(coleccion.find())

    def delete_document(self, collection_name, query):
        """Elimina un documento de una colección en base a una consulta."""
        coleccion = self.get_collection(collection_name)
        result = coleccion.delete_one(query)
        return result.deleted_count # Devuelve el número de documentos eliminados (debería ser 1 o 0)

    def insert_document(self, collection_name, document):
        """Inserta un documento en la colección especificada"""
        coleccion = self.get_collection(collection_name)
        result = coleccion.insert_one(document)
        return result.inserted_id #devuelve el id del documento insertado

    #metodo para obtener el documento
    def get_document(self, collection_name, filter):
        return self.bd[collection_name].find_one(filter)

    def update_document(self, collection_name, filter, update):
        """Actualiza un documento en la colección especificada según el filtro y los datos de actualización."""
        coleccion = self.get_collection(collection_name)
        result = coleccion.update_one(filter, update)
        return result.modified_count # Devuelve el número de documentos modificados (debería ser 1 o 0)
```

## 3. Menu.py.

```
import flet as ft
from conexion.conexion import MongoDBConnection
from vistas.usuarios_view import mostrar_usuarios_view
from vistas.clientes_view import mostrar_clientes_view
from vistas.productos_view import mostrar_productos_view

def main(page: ft.Page, cerrar_sesion_callback=None):
    page.title = "Barra de Navegación"
    page.window.resizable = True
    page.window.width = 1200
    page.window.height = 800
    page.window_x = 1
    page.window_y = 1
    page.padding = 0
    page.spacing = 0

    # Conexión a MongoDB
    db = MongoDBConnection()
```

```

# Contenedor de contenido dinámico
content = ft.Column([], alignment=ft.MainAxisAlignment.START, expand=True,
scroll="auto")

# Definición de la usuarios table para mostrar información de usuarios
users_table = ft.DataTable(
    border=ft.border.all(1, "grey"),
    border_radius=10,
    vertical_lines=ft.BorderSide(1, "grey"),
    width=1000,
    columns=[
        ft.DataColumn(ft.Text("Nombre")),
        ft.DataColumn(ft.Text("Clave")),
        ft.DataColumn(ft.Text("E-mail")),
        ft.DataColumn(ft.Text("Rol")),
        ft.DataColumn(ft.Text("Estado")),
        ft.DataColumn(ft.Text("Acciones")),
    ],
    rows=[],
)

# Definición de la clientes table para mostrar información de clientes
clientes_table = ft.DataTable(
    border=ft.border.all(1, "grey"),
    border_radius=10,
    vertical_lines=ft.BorderSide(1, "grey"),
    width=1000,
    columns=[
        ft.DataColumn(ft.Text("Nombre")),
        ft.DataColumn(ft.Text("Apellido")),
        ft.DataColumn(ft.Text("RUC")),
        ft.DataColumn(ft.Text("Dirección")),
        ft.DataColumn(ft.Text("Fecha de Nacimiento")),
        ft.DataColumn(ft.Text("Acciones")),
    ],
    rows=[],
)

# Definición de la productos table para mostrar información de productos
productos_table = ft.DataTable(
    border=ft.border.all(1, "grey"),
    border_radius=10,
    vertical_lines=ft.BorderSide(1, "grey"),
    width=1500,
    columns=[
        ft.DataColumn(ft.Text("Nombre", size=14,width=57)),
        ft.DataColumn(ft.Text("Marca", size=14)),
        ft.DataColumn(ft.Text("Precio de \nCompra", size=14)),
        ft.DataColumn(ft.Text("Precio de \nVenta", size=14)),
        ft.DataColumn(ft.Text("Stock \nDisponible", size=14,width=57)),
        ft.DataColumn(ft.Text("Cantidad \nMínima", size=14)),
        ft.DataColumn(ft.Text("Proveedor", size=14)),
        ft.DataColumn(ft.Text("Estado", size=14)),
        ft.DataColumn(ft.Text("Acciones", size=14)),
    ],
    rows=[],
)

```

```

    # Función y configuración para gestionar la navegación entre diferentes
    vistas (Usuarios, Clientes, Productos)
    # mediante un Navigation Rail, actualizando el contenido de la página según
    la selección del usuario.
    def on_navigation_change(e=None):
        selected_index = rail.selected_index
        content.controls.clear()
        if selected_index == 0:
            mostrar_usuarios_view(page, content, db, users_table)
        elif selected_index == 1:
            mostrar_clientes_view(page, content, db, clientes_table)
        elif selected_index == 2:
            mostrar_productos_view(page, content, db, productos_table)
        page.update()

    rail = ft.NavigationRail(
        min_width=80,
        min_extended_width=300,
        group_alignment=-0.9,
        selected_index=0,
        on_change=on_navigation_change,
        destinations=[
            ft.NavigationRailDestination(icon=ft.Icon(ft.Icons.PERSON),
            label="Usuarios"),

            ft.NavigationRailDestination(icon=ft.Icon(ft.Icons.SUPERVISED_USER_CIRCLE),
            label="Clientes"),

            ft.NavigationRailDestination(icon=ft.Icon(ft.Icons.SHOPIFY),
            label="Productos"),
        ],
    )

    # Botón de cierre de sesión que muestra un ícono de logout y ejecuta la
    función de cierre de sesión
    # al hacer clic, si está definida.
    logout_button = ft.IconButton(
        icon=ft.icons.LOGOUT_ROUNDED,
        icon_color="red",
        tooltip="Cerrar Sesión",
        on_click=lambda e: cerrar_sesion_callback(page) if cerrar_sesion_callback
    else None
    )

    # Agrega un destino al Navigation Rail para cerrar sesión, con el botón de
    logout como ícono.
    rail.destinations.append(
        ft.NavigationRailDestination(
            icon=logout_button,
            label="Salir"
        )
    )

    # Usa un Container para envolver el contenido y agregar el padding
    page.add(
        ft.Row(
            [
                rail,

```

```

        ft.VerticalDivider(width=0),
        ft.Container(content, expand=True,
alignment=ft.alignment.top_left, padding=ft.padding.only(right=20, bottom=6))
    ],
    expand=True,
)
)

on_navigation_change()

if __name__ == "__main__":
    ft.app(target=main)

```

## 4. Usuarios\_view.py

Nombre	Clave	E-mail	Rol	Estado	Acciones
Cristina	cristina	ccontreras@gmail.com	Vendedora	True	
Gustavo	Gcardozo	gustavocardozo23@gmail.com	Vendedor	True	
Jessica	JsC	jessicac@outlook.com	Cajero	True	
Benjamin	benja1234	bgaleano@outlook.es	cajero	True	
Jose	jj2024	josevillam@icloud.com	Vendedor	False	
Juan	JJ2323	juanr	Vendedor	True	
Nando	NNunez23	nandonunez23@hotmail.com	Administrador	True	
Amparo	ampv2312	amparo@outlook.es	Administrador	False	

## Código fuente:

```

import flet as ft

def mostrar_usuarios_view(page: ft.Page, content: ft.Column, db, users_table):
    # Contenedor para mostrar contenido dinámico
    print("Vista de usuarios cargada")
    page.title = "Gestión de Usuarios"
    page.window.resizable = False

    # Variables para la paginación
    usuarios_por_pagina = 8
    pagina_actual = 0
    usuario_seleccionado = None

    # Variables de búsqueda
    tf_buscar = ft.TextField(label="Buscar usuario", width=200)
    usuarios_filtrados = []

    # Título principal

```

```

titulo = ft.Text("Gestión de usuarios", size=24, weight="bold")

# Campos de entrada de usuario
tf_user = ft.TextField(label="Nombre de Usuario")
tf_password = ft.TextField(label="Clave")
tf_email = ft.TextField(label="E-mail")
tf_rol = ft.TextField(label="Rol")
tf_estado = ft.TextField(label="Estado")

#funcion para limpiar campos
def limpiar_campos():
    tf_user.value = ""
    tf_password.value = ""
    tf_email.value = ""
    tf_rol.value = ""
    tf_estado.value = ""
    tf_buscar.value = ""
    page.update()

# Función para cargar los usuarios en la tabla según la página actual,
# gestionando la paginación y actualizando la lista de productos filtrados.
def cargar_usuarios(pagina, usuarios=None):
    nonlocal usuarios_filtrados
    users = usuarios if usuarios is not None else
db.get_all_documents("usuarios")
    if not usuarios: # Si no se especifican usuarios filtrados, utilizar los
completos
        usuarios_filtrados = users

    users_table.rows.clear()
    inicio = pagina * usuarios_por_pagina
    fin = inicio + usuarios_por_pagina
    usuarios_pagina = usuarios_filtrados[inicio:fin]

    for user in usuarios_pagina:
        users_table.rows.append(
            ft.DataRow(
                cells=[
                    ft.DataCell(ft.Text(user.get("nombre", ""))),
                    ft.DataCell(ft.Text(user.get("clave", ""))),
                    ft.DataCell(ft.Text(user.get("email", ""))),
                    ft.DataCell(ft.Text(user.get("rol", ""))),
                    ft.DataCell(ft.Text(user.get("estado", ""))),
                    ft.DataCell(ft.Row([
                        ft.IconButton(icon=ft.icons.EDIT,
on_click=seleccionar_usuario, data=user["_id"]),
                        ft.IconButton(icon=ft.icons.DELETE,
on_click=eliminar_fila, data=user["_id"])
                    ]))
                ])
        )

# Actualizar el estado de los botones de paginación
btn_anterior.disabled = pagina_actual <= 0
btn_siguiete.disabled = fin >= len(usuarios_filtrados)
page.update()

```

```

def cambiar_pagina(incremento):
    nonlocal pagina_actual
    pagina_actual += incremento
    if pagina_actual < 0:
        pagina_actual = 0
    cargar_usuarios(pagina_actual, usuarios_filtrados)

# Función para validar los datos ingresados, guardar un nuevo usuario en la
base de datos
# y manejar posibles errores o advertencias al usuario.
def guardar_usuario(e):
    if not (
        tf_user.value.strip() and tf_password.value.strip() and
        tf_email.value.strip() and tf_rol.value.strip() and tf_estado.value.strip()):
        dialog = ft.AlertDialog(
            title=ft.Text("Advertencia"),
            content=ft.Text("Por favor, complete todos los campos antes de
continuar."),
            actions=[ft.TextButton("Aceptar", on_click=lambda e:
cerrar_dialogo(dialog))]
        )
        page.dialog = dialog
        dialog.open = True
        page.update()
        return

    nuevo_usuario = {
        "nombre": tf_user.value,
        "clave": tf_password.value,
        "email": tf_email.value,
        "rol": tf_rol.value,
        "estado": tf_estado.value
    }

    if db:
        try:
            db.insert_document("usuarios", nuevo_usuario)
        except Exception as ex:
            dialog = ft.AlertDialog(
                title=ft.Text("Error"),
                content=ft.Text(f"No se pudo guardar el usuario: {ex}"),
                actions=[ft.TextButton("Aceptar", on_click=lambda e:
cerrar_dialogo(dialog))]
            )
            page.dialog = dialog
            dialog.open = True
            page.update()
            return
    else:
        dialog = ft.AlertDialog(
            title=ft.Text("Error"),
            content=ft.Text("No se pudo conectar con la base de datos."),
            actions=[ft.TextButton("Aceptar", on_click=lambda e:
cerrar_dialogo(dialog))]
        )
        page.dialog = dialog
        dialog.open = True
        page.update()

```



```

        return

    limpiar_campos()
    cargar_usuarios(pagina_actual)

    # Función para modificar un usuario existente en la base de datos
    # después de validar que un usuario ha sido seleccionado y los campos están
    # completos.
    def modificar_usuario(e):
        if not usuario_seleccionado or not (
            tf_user.value.strip() and tf_password.value.strip() and
            tf_email.value.strip() and tf_rol.value.strip() and tf_estado.value.strip()):
            dialog = ft.AlertDialog(
                title=ft.Text("Advertencia"),
                content=ft.Text("Por favor, seleccione una fila para
                modificar."),
                actions=[ft.TextButton("Aceptar", on_click=lambda e:
                cerrar_dialogo(dialog))]
            )
            page.dialog = dialog
            dialog.open = True
            page.update()
            return

        db.update_document("usuarios", {"_id": usuario_seleccionado}, {
            "$set": {
                "nombre": tf_user.value,
                "clave": tf_password.value,
                "email": tf_email.value,
                "rol": tf_rol.value,
                "estado": tf_estado.value
            }
        })
        limpiar_campos()
        cargar_usuarios(pagina_actual)

    #funcion que elimina un usuario seleccionado en la base de datos
    def eliminar_fila(e):
        fila_id = e.control.data
        db.delete_document("usuarios", {"_id": fila_id})
        cargar_usuarios(pagina_actual)

    # Función para seleccionar un usuario de la base de datos y cargar sus datos
    # en el formulario.
    def seleccionar_usuario(e):
        nonlocal usuario_seleccionado
        fila_id = e.control.data
        usuario = db.get_document("usuarios", {"_id": fila_id})
        if usuario:
            usuario_seleccionado = fila_id
            tf_user.value = usuario.get("nombre", "")
            tf_password.value = usuario.get("clave", "")
            tf_email.value = usuario.get("email", "")
            tf_rol.value = usuario.get("rol", "")
            tf_estado.value = usuario.get("estado", "")
            page.update()

    def cerrar_dialogo(dialog):

```

```

        dialog.open = False
        page.update()

# Función para buscar usuarios en la base de datos según el texto ingresado
# y actualizar la lista de productos mostrados en la interfaz.
def buscar_usuario(e):
    nonlocal usuarios_filtrados
    query = tf_buscar.value.strip().lower()
    if query:
        usuarios_filtrados = [user for user in
db.get_all_documents("usuarios") if query in user.get("nombre", "").lower()]
    else:
        usuarios_filtrados = db.get_all_documents("usuarios")
        cargar_usuarios(pagina_actual, usuarios_filtrados)

    btn_anterior = ft.IconButton(icon=ft.icons.ARROW_LEFT, on_click=lambda e:
cambiar_pagina(-1))
    btn_siguiente = ft.IconButton(icon=ft.icons.ARROW_RIGHT, on_click=lambda e:
cambiar_pagina(1))
    btn_buscar = ft.OutlinedButton(text="Buscar", on_click=buscar_usuario)

    navegacion_row = ft.Row(
        [
            tf_buscar,
            btn_buscar,
            btn_anterior,
            btn_siguiente,
        ],
        alignment=ft.MainAxisAlignment.START,
        spacing=10,
    )

    submit = ft.OutlinedButton(text="Guardar", on_click=guardar_usuario)
    modificar_button = ft.OutlinedButton(text="Modificar",
on_click=modificar_usuario)
    limpiar_button = ft.OutlinedButton(text="Limpiar Campos", on_click=lambda e:
limpiar_campos())
    botones_row = ft.Row([submit, modificar_button, limpiar_button], spacing=10)

    espacio = ft.Container(height=20)

    content.controls.clear()
    content.controls.append(
        ft.Column(
            [
                titulo,
                ft.Row([tf_user, tf_password, tf_email], spacing=10),
                ft.Row([tf_rol, tf_estado], spacing=10),
                botones_row,
                espacio,
                navegacion_row,
                users_table,
            ],
            alignment=ft.MainAxisAlignment.START,
            spacing=10,
        )
    )
)

```

```
cargar_usuarios(pagina_actual)
page.update()
```

## 5. Clientes.view.py

















**Gestión de Clientes**

Usuarios  
Clientes  
Productos  
Salir

Nombre:  Apellido:  Ruc:

Dirección:  Fecha de Nacimiento aa-mm-dd:

Buscar Cliente:   < >

Nombre	Apellido	RUC	Dirección	Fecha de Nacimiento	Acciones
Juan	Pérez	2343564-0	Caacupe	1990-05-15	 
Dionicio	Pérez	3453123-9	Asuncion	2003-05-15	 
Marcela	Alcaraz	67652323	Eusebio Ayala	2001-06-07	 
Lile	Cartman	3987990	Isla Pucu	2003-09-22	 
Robert	Baratheon	8997786	Caacupe	2001-02-14	 
Carlos	Girett	8765456	San Jose Obrero	2003-03-30	 
Melissa	Mendez	878670	Asuncion	2003-12-20	 
Luis	Cabañas	2342178-9	Isla Pucu	1998-12-13	 

### Código fuente:

```
import flet as ft

def mostrar_clientes_view(page: ft.Page, content: ft.Column, db, clientes_table):
    print("Vista de Clientes cargada")
    page.title = "Gestión de Clientes"

    #variables de paginacion
    clientes_por_pagina = 10
    pagina_actual = 0
    cliente_seleccionado = None

    #variables para la busqueda
    tf_buscar = ft.TextField(label="Buscar Cliente", width=200)
    clientes_filtrados = []

    #titulo principal
    titulo = ft.Text("Gestión de Clientes", size=24, weight="bold")

    #campos de entrada de clientes
    tf_nombre = ft.TextField(label="Nombre")
    tf_apellido = ft.TextField(label="Apellido")
    tf_ruc = ft.TextField(label="Ruc")
    tf_direccion = ft.TextField(label="Direccion")
    tf_fecha_nacimiento = ft.TextField(label="Fecha de Nacimiento aa-mm-dd")

    #limpia los campos de texto
    def limpiar_campos():
        tf_nombre.value = ""
        tf_apellido.value = ""
        tf_ruc.value = ""
        tf_direccion.value = ""
```

```

        tf_fecha_nacimiento.value = ""
        tf_buscar.value = ""
        page.update()

# Función para cargar los registros de clientes en la tabla según la página
actual,
# gestionando la paginación y actualizando la lista de clientes filtrados.
def cargar_clientes(pagina, clientes=None, clientes_por_pagina=8):
    nonlocal clientes_filtrados
    customers = clientes if clientes is not None else
db.get_all_documents("clientes")
    if not clientes:
        clientes_filtrados = customers

    clientes_table.rows.clear()
    inicio = pagina * clientes_por_pagina
    fin = inicio + clientes_por_pagina
    clientes_por_pagina = clientes_filtrados[inicio:fin]

    for customers in clientes_por_pagina:
        clientes_table.rows.append(
            ft.DataRow(
                cells=[
                    ft.DataCell(ft.Text(customers.get("nombre", ""))),
                    ft.DataCell(ft.Text(customers.get("apellido", ""))),
                    ft.DataCell(ft.Text(customers.get("ruc", ""))),
                    ft.DataCell(ft.Text(customers.get("direccion", ""))),
                    ft.DataCell(ft.Text(customers.get("fecha_nacimiento",
""))),
                    ft.DataCell(ft.Row([
                        ft.IconButton(icon=ft.icons.EDIT,
on_click=seleccionar_cliente, data=customers["_id"]),
                        ft.IconButton(icon=ft.icons.DELETE,
on_click=eliminar_fila, data=customers["_id"])
                    ]))
                ])
            )
        )

#Actualiza el estado de los botones de paginacion
btn_anterior.disabled = pagina_actual <=0
btn_siguiente.disabled = fin >= len(clientes_filtrados)
page.update()

def cambiar_pagina(incremento):
    nonlocal pagina_actual
    pagina_actual += incremento
    if pagina_actual < 0:
        pagina_actual = 0
    cargar_clientes(pagina_actual, clientes_filtrados)

# Función para validar los datos ingresados, guardar un nuevo cliente en la
base de datos
# y manejar posibles errores o advertencias al usuario.
def guardar_cliente(e):
    if not(
        tf_nombre.value.strip() and tf_apellido.value.strip() and
        tf_ruc.value.strip() and tf_direccion.value.strip() and

```

```

tf_fecha_nacimiento.value.strip()):
    dialog = ft.AlertDialog(
        title=ft.Text("Advertencia"),
        content=ft.Text("Por favor, complete todos los campos antes de
continuar."),
        actions=[ft.TextButton("Aceptar", on_click=lambda e:
cerrar_dialogo(dialog))]
    )
    page.dialog = dialog
    dialog.open = True
    page.update()
    return

nuevo_cliente = {
    "nombre": tf_nombre.value,
    "apellido": tf_apellido.value,
    "ruc": tf_ruc.value,
    "direccion": tf_direccion.value,
    "fecha_nacimiento": tf_fecha_nacimiento.value
}

if db:
    try:
        db.insert_document("clientes", nuevo_cliente)
    except Exception as ex:
        dialog = ft.AlertDialog(
            title=ft.Text("Error"),
            content=ft.Text(f"No se pudo guardar el cliente: {ex}"),
            actions=[ft.TextButton("Aceptar", on_click=lambda e:
cerrar_dialogo(dialog))]
        )
        page.dialog = dialog
        dialog.open = True
        page.update()
        return
    else:
        dialog = ft.AlertDialog(
            title=ft.Text("Error"),
            content=ft.Text("No se pudo conectar con la base de datos."),
            actions=[ft.TextButton("Aceptar", on_click=lambda e:
cerrar_dialogo(dialog))]
        )
        page.dialog = dialog
        dialog.open = True
        page.update()
        return

limpiar_campos()
cargar_clientes(pagina_actual)

# Función para modificar un registro de cliente existente en la base de datos
# después de validar que un cliente ha sido seleccionado y los campos están
completos.
def modificar_cliente(e):
    if not cliente_seleccionado or not(
        tf_nombre.value.strip() and tf_apellido.value.strip() and
        tf_ruc.value.strip() and tf_direccion.value.strip() and
        tf_fecha_nacimiento.value.strip()):

```

```

        dialog = ft.AlertDialog(
            title=ft.Text("Advertencia"),
            content=ft.Text("Por favor, seleccione una fila para
modificar."),
            actions=[ft.TextButton("Aceptar", on_click=lambda e:
cerrar_dialogo(dialog))]
        )
        page.dialog = dialog
        dialog.open = True
        page.update()
        return

    db.update_document("clientes", {"_id": cliente_seleccionado},{
        "$set":{
            "nombre":tf_nombre.value,
            "apellido":tf_apellido.value,
            "ruc":tf_ruc.value,
            "direccion": tf_direccion.value,
            "fecha_nacimiento":tf_fecha_nacimiento.value
        }
    })
    limpiar_campos()
    cargar_clientes(pagina_actual)

#funcion para eliminar cliente seleccionado de la base de datos
def eliminar_fila(e):
    fila_id = e.control.data
    db.delete_document("clientes", {"_id": fila_id})
    cargar_clientes(pagina_actual)

# Función para seleccionar un cliente de la base de datos y cargar sus datos
en el formulario.
def seleccionar_cliente(e):
    nonlocal cliente_seleccionado
    fila_id = e.control.data
    cliente = db.get_document("clientes", {"_id": fila_id})
    if cliente:
        cliente_seleccionado = fila_id
        tf_nombre.value = cliente.get("nombre", "")
        tf_apellido.value = cliente.get("apellido", "")
        tf_ruc.value = cliente.get("ruc", "")
        tf_direccion.value = cliente.get("direccion", "")
        tf_fecha_nacimiento.value = cliente.get("fecha_nacimiento", "")
        page.update()

def cerrar_dialogo(dialog):
    dialog.open = False
    page.update()

# Función para buscar clientes en la base de datos según el texto ingresado
# y actualizar la lista de clientes mostrados en la interfaz.
def buscar_cliente(e):
    nonlocal clientes_filtrados
    query = tf_buscar.value.strip().lower()
    if query:
        clientes_filtrados = [customers for customers in
db.get_all_documents("clientes") if
                                query in customers.get("nombre", "").lower()]

```

```

        else:
            clientes_filtrados = db.get_all_documents("clientes")
            cargar_clientes(pagina_actual, clientes_filtrados)

            btn_anterior = ft.IconButton(icon=ft.icons.ARROW_LEFT, on_click=lambda e:
cambiar_pagina(-1))
            btn_siguiente = ft.IconButton(icon=ft.icons.ARROW_RIGHT, on_click=lambda e:
cambiar_pagina(1))
            btn_buscar = ft.OutlinedButton(text="Buscar", on_click=buscar_cliente)

            navegacion_row = ft.Row(
                [
                    tf_buscar,
                    btn_buscar,
                    btn_anterior,
                    btn_siguiente,
                ],
                alignment=ft.MainAxisAlignment.START,
                spacing=10,
            )

            submit = ft.OutlinedButton(text="Guardar", on_click=guardar_cliente)
            modificar_button = ft.OutlinedButton(text="Modificar",
on_click=modificar_cliente)
            limpiar_button = ft.OutlinedButton(text="Limpiar Campos", on_click=lambda e:
limpiar_campos())
            botones_row = ft.Row([submit, modificar_button, limpiar_button], spacing=10)

            espacio = ft.Container(height=20)

            content.controls.clear()
            content.controls.append(
                ft.Column(
                    [
                        titulo,
                        ft.Row([tf_nombre, tf_apellido, tf_ruc], spacing=10),
                        ft.Row([tf_direccion, tf_fecha_nacimiento], spacing=10),
                        botones_row,
                        espacio,
                        navegacion_row,
                        clientes_table,
                    ],
                    alignment=ft.MainAxisAlignment.START,
                    spacing=10,
                )
            )

            cargar_clientes(pagina_actual)
            page.update()

```

## 6. Productos.view.py

Usuarios

Cientes

Productos

Salir

### Gestión de Productos

Nombre

Marca

Precio de compra

Precio de venta

Stock

Cantidad Mínima

Proveedor

Estado

Guardar

Modificar







Limpiar Campos

Buscar Producto

Buscar

<

>

Nombre	Marca	Precio de Compra	Precio de Venta	Stock Disponible	Cantidad Mínima	Proveedor	Estado	Acciones
Aqua de Gio 100 ml	Armani	702.800	998.900	15	10	Armani	Disponible	 
protector solar 120ml	La Roche Posay	89.250	210.900	20	10	La Roche Posay	Disponible	 
Polo blue rahl lauren edt 200ml	Ralph Lauren	889.750	1.336.000	8	8	Ralph Lauren	Disponible	 

```

import flet as ft
def mostrar_productos_view(page: ft.Page, content: ft.Column, db,
productos_table):
    print("Vista de productos cargada")
    page.title = "Gestión de Productos"

    # Variables de paginación
    productos_por_pagina = 10
    pagina_actual = 0
    producto_seleccionado = None

    # Variables para la búsqueda
    tf_buscar = ft.TextField(label="Buscar Producto", width=200)
    productos_filtrados = []

    # Título principal
    titulo = ft.Text("Gestión de Productos", size=24, weight="bold")

    # Campos de entrada de productos
    tf_nombre = ft.TextField(label="Nombre")
    tf_marca = ft.TextField(label="Marca")
    tf_precio_compra = ft.TextField(label="Precio de compra")
    tf_precio_venta = ft.TextField(label="Precio de venta")
    tf_stock = ft.TextField(label="Stock")
    tf_cantidad_minima = ft.TextField(label="Cantidad Minima")
    tf_proveedor = ft.TextField(label="Proveedor")
    tf_estado = ft.TextField(label="Estado")

    # Limpiar los campos de texto
    def limpiar_campos():
        tf_nombre.value = ""
        tf_marca.value = ""
        tf_precio_compra.value = ""
        tf_precio_venta.value = ""
        tf_stock.value = ""
        tf_cantidad_minima.value = ""
        tf_proveedor.value = ""
        tf_estado.value = ""

```



```

tf_buscar.value = ""
page.update()

# Función para cargar los productos en la tabla según la página actual,
# gestionando la paginación y actualizando la lista de productos filtrados.
def cargar_productos(pagina, productos=None, productos_por_pagina=8):
    nonlocal productos_filtrados
    products = productos if productos is not None else
db.get_all_documents("productos")
    if not productos:
        productos_filtrados = products

    productos_table.rows.clear()
    inicio = pagina * productos_por_pagina
    fin = inicio + productos_por_pagina
    productos_por_pagina = productos_filtrados[inicio:fin]

    # Código para generar dinámicamente las filas de una tabla con los datos
    de los productos
    # gestiona la paginación y las acciones como editar o eliminar.
    for product in productos_por_pagina:
        productos_table.rows.append(
            ft.DataRow(
                cells=[
                    ft.DataCell(
                        ft.Text(product.get("nombre", ""),
max_lines=3,style=ft.TextStyle(size=12)) # Ajustar el texto a máximo 2 líneas
                    ),
                    ft.DataCell(
                        ft.Text(product.get("marca", ""),
max_lines=3,style=ft.TextStyle(size=12))
                    ),
                    ft.DataCell(
                        ft.Text(product.get("Precio_compra", ""),
max_lines=3, style=ft.TextStyle(size=12))
                    ),
                    ft.DataCell(
                        ft.Text(product.get("precio_venta", ""), max_lines=3,
style=ft.TextStyle(size=12))
                    ),
                    ft.DataCell(
                        ft.Text(product.get("stock_disponible", ""),
max_lines=3, style=ft.TextStyle(size=12))
                    ),
                    ft.DataCell(
                        ft.Text(product.get("cantidad_minima", ""),
max_lines=3,style=ft.TextStyle(size=12))
                    ),
                    ft.DataCell(
                        ft.Text(product.get("proveedor", ""),
max_lines=3,style=ft.TextStyle(size=11))
                    ),
                    ft.DataCell(
                        ft.Text(product.get("estado", ""),
max_lines=3,style=ft.TextStyle(size=11))
                    ),
                    ft.DataCell(
                        ft.Row([

```

```

        ft.IconButton(icon=ft.icons.EDIT,
on_click=seleccionar_producto, data=product["_id"]),
        ft.IconButton(icon=ft.icons.DELETE,
on_click=eliminar_fila, data=product["_id"])
    ])
    )
    ]
    )
    )
    # Actualiza el estado de los botones de paginación
    btn_anterior.disabled = pagina_actual <= 0
    btn_siguiete.disabled = fin >= len(productos_filtrados)
    page.update()

#funcion para cambiar de pagina en la tabla
def cambiar_pagina(incremento):
    nonlocal pagina_actual
    pagina_actual += incremento
    if pagina_actual < 0:
        pagina_actual = 0
    cargar_productos(pagina_actual, productos_filtrados)

# Función para validar los datos ingresados, guardar un nuevo producto en la
base de datos
# y manejar posibles errores o advertencias al usuario.
def guardar_producto(e):
    if not(
        tf_nombre.value.strip() and tf_marca.value.strip() and
tf_precio_compra.value.strip() and tf_precio_venta.value.strip() and
tf_stock.value.strip() and tf_cantidad_minima.value.strip() and
tf_proveedor.value.strip() and tf_estado.value.strip()):
        dialog = ft.AlertDialog(
            title=ft.Text("Advertencia"),
            content=ft.Text("Por favor, complete todos los campos antes de
continuar."),
            actions=[ft.TextButton("Aceptar", on_click=lambda e:
cerrar_dialogo(dialog))]
        )
        page.dialog = dialog
        dialog.open = True
        page.update()
        return

    nuevo_producto = {
        "nombre": tf_nombre.value,
        "marca": tf_marca.value,
        "Precio_compra": tf_precio_compra.value,
        "precio_venta": tf_precio_venta.value,
        "stock_disponible": tf_stock.value,
        "cantidad_minima": tf_cantidad_minima.value,
        "proveedor": tf_proveedor.value,
        "estado": tf_estado.value
    }

    if db:
        try:
            db.insert_document("productos", nuevo_producto)
        except Exception as ex:

```

```

        dialog = ft.AlertDialog(
            title=ft.Text("Error"),
            content=ft.Text(f"No se pudo guardar el producto: {ex}"),
            actions=[ft.TextButton("Aceptar", on_click=lambda e:
cerrar_dialogo(dialog))]
        )
        page.dialog = dialog
        dialog.open = True
        page.update()
        return
    else:
        dialog = ft.AlertDialog(
            title=ft.Text("Error"),
            content=ft.Text("No se pudo conectar con la base de datos."),
            actions=[ft.TextButton("Aceptar", on_click=lambda e:
cerrar_dialogo(dialog))]
        )
        page.dialog = dialog
        dialog.open = True
        page.update()
        return
    limpiar_campos()
    cargar_productos(pagina_actual)

# Función para modificar un producto existente en la base de datos
# después de validar que un producto ha sido seleccionado y los campos están
completos.
def modificar_producto(e):
    if not producto_seleccionado or not(
        tf_nombre.value.strip() and tf_marca.value.strip() and
tf_precio_compra.value.strip() and tf_precio_venta.value.strip() and
tf_stock.value.strip() and tf_cantidad_minima.value.strip() and
tf_proveedor.value.strip() and tf_estado.value.strip()):
        dialog = ft.AlertDialog(
            title=ft.Text("Advertencia"),
            content=ft.Text("Por favor, seleccione una fila para
modificar."),
            actions=[ft.TextButton("Aceptar", on_click=lambda e:
cerrar_dialogo(dialog))]
        )
        page.dialog = dialog
        dialog.open = True
        page.update()
        return

    db.update_document("productos", {"_id": producto_seleccionado}, {
        "$set": {
            "nombre": tf_nombre.value,
            "marca": tf_marca.value,
            "Precio_compra": tf_precio_compra.value,
            "precio_venta": tf_precio_venta.value,
            "stock_disponible": tf_stock.value,
            "cantidad_minima": tf_cantidad_minima.value,
            "proveedor": tf_proveedor.value,
            "estado": tf_estado.value
        }
    })
    limpiar_campos()

```

```

        cargar_productos(pagina_actual)

#funcion que elimina el producto seleccionado
def eliminar_fila(e):
    fila_id = e.control.data
    db.delete_document("productos", {"_id": fila_id})
    cargar_productos(pagina_actual)

# Función para seleccionar un producto de la base de datos y cargar sus datos
en el formulario.
def seleccionar_producto(e):
    nonlocal producto_seleccionado
    fila_id = e.control.data
    producto = db.get_document("productos", {"_id": fila_id})
    if producto:
        producto_seleccionado = fila_id
        tf_nombre.value = producto.get("nombre", "")
        tf_marca.value = producto.get("marca", "")
        tf_precio_compra.value = producto.get("Precio_compra", "")
        tf_precio_venta.value = producto.get("precio_venta", "")
        tf_stock.value = producto.get("stock_disponible", "")
        tf_cantidad_minima.value = producto.get("cantidad_minima", "")
        tf_proveedor.value = producto.get("proveedor", "")
        tf_estado.value = producto.get("estado", "")
        page.update()

def cerrar_dialogo(dialog):
    dialog.open = False
    page.update()

# Función para buscar productos en la base de datos según el texto ingresado
# y actualizar la lista de productos mostrados en la interfaz.
def buscar_producto(e):
    nonlocal productos_filtrados
    query = tf_buscar.value.strip().lower()
    if query:
        productos_filtrados = [product for product in
db.get_all_documents("productos") if
                                query in product.get("nombre", "").lower()]
    else:
        productos_filtrados = db.get_all_documents("productos")
        cargar_productos(pagina_actual, productos_filtrados)

    btn_anterior = ft.IconButton(icon=ft.icons.ARROW_LEFT, on_click=lambda e:
cambiar_pagina(-1))
    btn_siguiente = ft.IconButton(icon=ft.icons.ARROW_RIGHT, on_click=lambda e:
cambiar_pagina(1))
    btn_buscar = ft.OutlinedButton(text="Buscar", on_click=buscar_producto)

    navegacion_row = ft.Row([
        tf_buscar,
        btn_buscar,
        btn_anterior,
        btn_siguiente,
    ],
        alignment=ft.MainAxisAlignment.START,
        spacing=10
    )

```

```

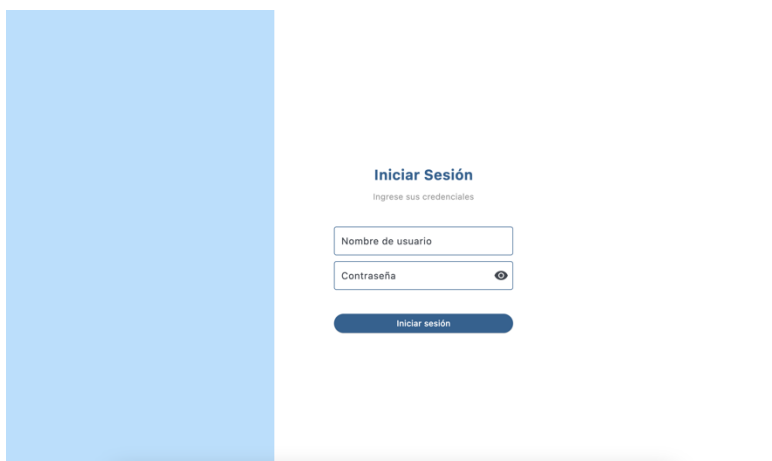
submit = ft.OutlinedButton(text="Guardar", on_click=guardar_producto)
modificar_button = ft.OutlinedButton(text="Modificar",
on_click=modificar_producto)
limpiar_button = ft.OutlinedButton(text="Limpiar Campos", on_click=lambda e:
limpiar_campos())
botones_row = ft.Row([submit, modificar_button, limpiar_button], spacing=10)

espacio = ft.Container(height=20)

content.controls.clear()
content.controls.append(
    ft.Column(
        [
            titulo,
            ft.Row([tf_nombre, tf_marca, tf_precio_compra], spacing=10),
            ft.Row([tf_precio_venta, tf_stock, tf_cantidad_minima],
spacing=10),
            ft.Row([tf_proveedor, tf_estado], spacing=10),
            botones_row,
            espacio,
            navegacion_row,
            productos_table
        ],
        alignment=ft.MainAxisAlignment.START,
        spacing=10,
    )
)
cargar_productos(pagina_actual)
page.update()

```

## 7. Login.py



## Código fuente:

```
# Importar las bibliotecas necesarias
import flet as ft
from conexion.conexion import MongoDBConnection
from menu import main as menu_main
import logging

# Configurar logging para depuración
logging.basicConfig(level=logging.INFO)

# Variables globales para gestionar el estado de autenticación
usuario_autenticado = False
usuario_logueado = None

def main(page: ft.Page):
    """
    Función principal que configura la ventana de la aplicación
    y gestiona el flujo de inicio de sesión.
    """
    global usuario_autenticado, usuario_logueado

    # Configuración inicial de la ventana de la aplicación
    page.padding = 0
    page.spacing = 0
    page.title = "Sistema de Gestión"
    page.window.width = 1300
    page.window.height = 800
    page.window.resizable = False
    page.window_x = 0
    page.window_y = 0
    page.bgcolor = ft.colors.WHITE

    # Conexión con la base de datos
    db = MongoDBConnection()

    def cerrar_sesion(page: ft.Page):
        """
        Restablece el estado de autenticación y muestra la pantalla de login.
        """
        global usuario_autenticado, usuario_logueado
        usuario_autenticado = False
        usuario_logueado = None
        page.controls.clear()
        page.update()
        mostrar_login()

    def mostrar_login():
        """
        Muestra la interfaz de inicio de sesión.
        """

        def validar_credenciales(e):
            """
            Valida las credenciales ingresadas contra la base de datos.
            """
            global usuario_autenticado, usuario_logueado
```

```

try:
    # Obtener valores ingresados
    usuario = username_input.value.strip()
    clave = password_input.value.strip()

    if not usuario or not clave:
        mostrar_advertencia("Por favor, completa todos los campos.")
        return

    # Buscar usuario en la base de datos (sin distinguir
mayúsculas/minúsculas)
    user_data = db.get_document(
        "usuarios",
        {"nombre": {"$regex": f"^{usuario}$", "$options": "i"}}
    )

    if user_data:
        # Verificar contraseña
        stored_password = user_data.get("clave")
        if clave == stored_password:
            # Verificar si la cuenta está activa
            if str(user_data.get("estado", "")).lower() == "true":
                usuario_autenticado = True
                usuario_logueado = user_data
                mostrar_menu()
            else:
                mostrar_advertencia("Cuenta desactivada. Contacte al
administrador.")
        else:
            mostrar_advertencia("Contraseña incorrecta.")
    else:
        mostrar_advertencia("Usuario no encontrado.")
except Exception as ex:
    # Manejo de errores de validación
    logging.error(f"Error al validar credenciales: {str(ex)}")
    mostrar_advertencia(f"Error de sistema: {str(ex)}")

def mostrar_advertencia(mensaje):
    """
    Muestra un cuadro de diálogo con un mensaje de advertencia.
    """
    def cerrar_dialogo(e):
        page.dialog.open = False
        page.update()

    dialog = ft.AlertDialog(
        title=ft.Text("Error"),
        content=ft.Text(mensaje),
        actions=[ft.TextButton("Cerrar", on_click=cerrar_dialogo)]
    )
    page.dialog = dialog
    dialog.open = True
    page.update()

def mostrar_menu():
    """
    Muestra el menú principal si la autenticación es exitosa.

```

```

    """

    global usuario_autenticado
    page.controls.clear()
    page.update()

    if usuario_autenticado:
        menu_main(page, cerrar_sesion)
        page.update()

# Componentes de la interfaz de login
username_input = ft.TextField(
    label="Nombre de usuario",
    width=300,
    border_color=ft.colors.PRIMARY,
    focused_border_color=ft.colors.PRIMARY_CONTAINER
)

password_input = ft.TextField(
    label="Contraseña",
    width=300,
    password=True,
    can_reveal_password=True,
    border_color=ft.colors.PRIMARY,
    focused_border_color=ft.colors.PRIMARY_CONTAINER
)

login_button = ft.ElevatedButton(
    "Iniciar sesión",
    width=300,
    on_click=validar_credenciales,
    style=ft.ButtonStyle(
        bgcolor={
            ft.MaterialState.DEFAULT: ft.colors.PRIMARY,
            ft.MaterialState.HOVERED: ft.colors.PRIMARY_CONTAINER
        },
        color={
            ft.MaterialState.DEFAULT: ft.colors.WHITE
        }
    )
)

# Diseño de la pantalla de login
login_content = ft.Row(
    [
        # Panel izquierdo con imagen
        ft.Container(
            width=450,
            height=800,
            bgcolor=ft.colors.BLUE_100,
            content=ft.Image(
                src="/api/placeholder/500/600",
                width=500,
                height=600,
                fit=ft.ImageFit.COVER
            )
        ),
        # Panel derecho con formulario de login
        ft.Container(

```



```

        width=500,
        height=600,
        padding=50,
        content=ft.Column(
            [
                ft.Text(
                    "Iniciar Sesión",
                    size=24,
                    weight=ft.FontWeight.BOLD,
                    color=ft.colors.PRIMARY
                ),
                ft.Text(
                    "Ingrese sus credenciales",
                    size=14,
                    color=ft.colors.GREY
                ),
                ft.Container(height=20),
                username_input,
                password_input,
                ft.Container(height=20),
                login_button
            ],
            alignment=ft.MainAxisAlignment.CENTER,
            horizontal_alignment=ft.CrossAxisAlignment.CENTER,
            spacing=10
        )
    ),
    ],
    spacing=0,
    alignment=ft.MainAxisAlignment.START
)

# Mostrar la pantalla de login
page.controls.clear()
page.add(login_content)
page.update()

# Mostrar la pantalla de login al iniciar
mostrar_login()

# Iniciar la aplicación
ft.app(target=main)

```

## 8. Repositorio de GitHub del proyecto.

<https://github.com/carlosgirett/crud-mongodb/tree/main>