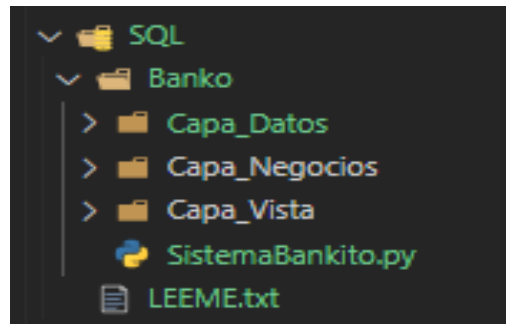


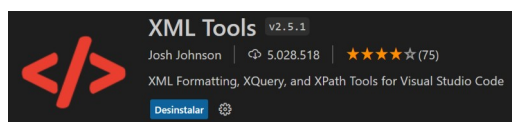
Python-SQL-QT

Arquitectura 3 capas

- **Primero la BD**
 - **EJECUTAR** 1-ScriptBancoMySQL
- **Prepara la estructura de carpetas** para el sistema de Banco “BANKO”
 - Crear Carpeta con 3 subcarpetas en una ubicaremos el los archivos de la capa vista y en la otra los archivos de la capa negocio y en la capa Datos la conexión con la Base de Datos



- En la Capa Datos agregaremos las clases necesaria que nos conectan a la base de datos que ya hemos aprendido junto con un archivo `__init__.py` para indicarle a python que consideramos esa carpeta como un paquete
- en la capa vista guardaremos todos los archivos que hacen a la **USER INTERFACE** del sistema
- por ultimo creamos el archivo que gestionara la aplicación
- Para crear las interfaces de usuario necesitamos alguna interface `tkinter` o `QT`
 - para esta practica elegi `QT` (a pedido del publico presente)
 - **para ayuda de uso** :<https://build-system.fman.io/pyqt5-tutorial>
- **CAPA DATOS**
 - **SCRIPT**
 - **extensiones (ver teoria)**
 - **crear conexión y chequear**
- hacemos **git init**
- **MOVER A TERMINAL POWERSHELL**
 - `pip list`



```
PS C:\workspace\python2024> pip list
Package Version
-----
pip      23.2.1

[notice] A new release of pip is available: 23.2.1 -> 24.2
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\workspace\python2024> python.exe -m pip install --upgrade pip
Requirement already satisfied: pip in c:\python312\lib\site-packages (23.2.1)
Collecting pip
  Obtaining dependency information for pip from https://files.pythonhosted.org/packages/d4/55/90db48d85f7689ec6f81c0db0622d704306c5284850383c090e6c1
  5a5c/pip-24.2-py3-none-any.whl.metadata
  Downloading pip-24.2-py3-none-any.whl.metadata (3.6 kB)
  Downloading pip-24.2-py3-none-any.whl (1.8 MB)
    1.8/1.8 MB 1.8 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 23.2.1
    Uninstalling pip-23.2.1:
      ERROR: Could not install packages due to an OSError: [WinError 5] Acceso denegado: 'c:\python312\lib\site-packages\pip-23.2.1.dist-info\AUTHORS.txt'
  Consider using the '--user' option or check the permissions.

[notice] A new release of pip is available: 23.2.1 -> 24.2
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\workspace\python2024> []
```

- python3.exe -m pip install --upgrade pip
- Instalamos el primer conector
 - pip install mysql-conector-python

CODIGO

```
from ast import Str
import pymysql.cursors
# https://github.com/PyMySQL/PyMySQL

class Database:
    # Genero la apertura y cierre con la Base de Datos
    def __init__(self):
        self.__conectada = False
        self.__cnx=self.crearConexion()
        if self.bdConectada:
            self.__cursor= self.__cnx.cursor()
            print("Conexion establecida")

    def crearConexion(self):
        try:
            cnx= pymysql.connect(
                host='localhost',
                user='root',
                password='hepatalgina',
                database='banco',
                cursorclass=pymysql.cursors.DictCursor)
            self.__conectada=True
            return cnx

        except Exception as e:
            self.__conectada=False
            print("error de conexion")
            raise

    @property
    def bdConectada(self):
        return self.__conectada

    @property
    def connection(self):
        if self.bdConectada:
            return self.__cnx
        else:
            # identificar codigo repetido arreglar
            self.__cnx=self.crearConexion()
            if self.bdConectada:
                self.__cursor = self.__cnx.cursor()
                print("Conexion establecida")

    @property
    def cursor(self):
        return self.__cursor

    def close(self):
        self.connection.close()

#-----
class Cliente(Database):
    def __init__(self):
        super().__init__()

    def getCliente(self,id):
        sql="SELECT * FROM banco.cliente where idCliente='{}'".format(id)
        print(sql)
        try:
            self.cursor.execute(sql)
            unCliente=self.cursor.fetchone()
            return unCliente

        except Exception as e:
            raise

    def getTodos(self):
        sql= 'SELECT * FROM banco.cliente'
        try:
            self.cursor.execute(sql)
            clientes=self.cursor.fetchall()
            return clientes

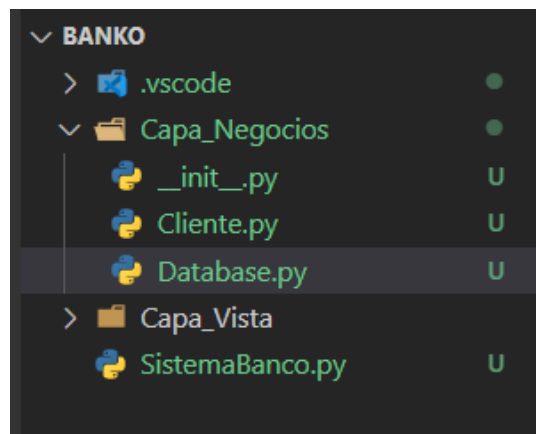
        except Exception as e:
            raise

    def insertCliente(self,nroCliente,nombre,direccion):
        sql="INSERT INTO banco.cliente VALUES ('{}','{}','{}')".format(nroCliente,nombre,direccion)
        try:
            self.cursor.execute(sql)
            self.connection.commit()
        except Exception as e:
            raise

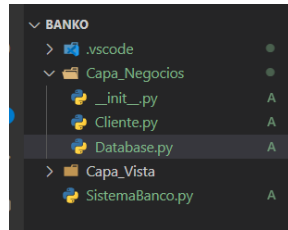
    def updateCliente(self,nroCliente,nombre,direccion):
        sql="UPDATE banco.cliente SET clienteNombre='{}', clienteDomicilio = '{} ' WHERE idCliente = '{} '".format(nombre,direccion,nroCliente)
        try:
            self.cursor.execute(sql)
            self.connection.commit()
        except Exception as e:
            raise

    def deleteCliente(self,nroCliente):
        sql="DELETE FROM banco.cliente WHERE idCliente = '{} '".format(nroCliente)
        try:
            self.cursor.execute(sql)
            self.connection.commit()
        except Exception as e:
            raise

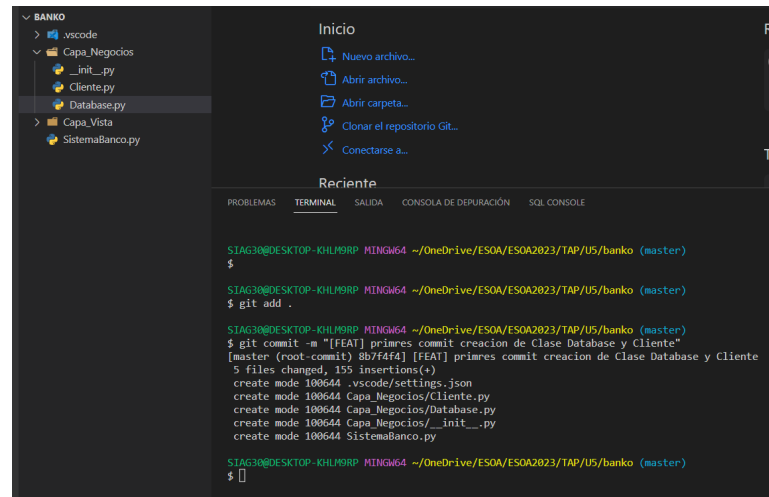
    def existeCliente(self,nroCliente):
        sql="SELECT * FROM banco.cliente where idCliente='{}'".format(nroCliente)
        cliente=self.cursor.fetchone()
        return cliente != None
```



- Es tiempo de mandar código al STAGE
 - `git add .`

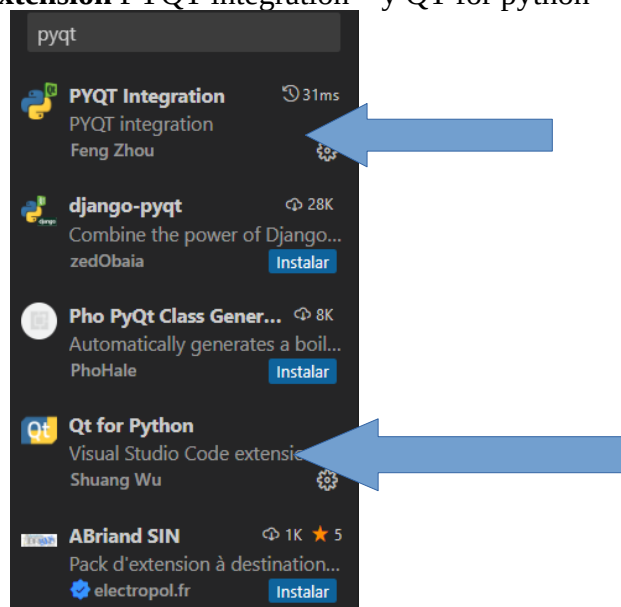


- De hecho podemos crear nuestro primer commit -m "primercommit"



INTERFAZ GRAFICA

- 2022 → Para crear las interfaces de usuario necesitamos alguna interface tkinter o QT
 - para esta practica elegi QT (a pedido del publico presente)
 - documentacion oficial en :
 - <https://doc.qt.io/qtforpython-6/tools/pyside-designer.html#pyside6-designer>
- **AHORA** entramos a nuestro IDE :VSCODE y lo instalamos
- **PASOS** de INSTALACION:
 - QT
 - PRIMERO instalamos **extension PYQT integration** y QT for python

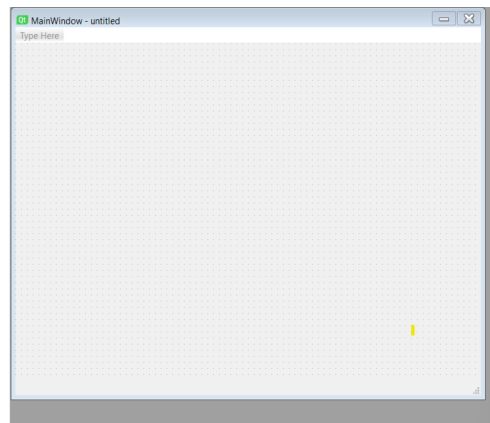


- con pyside6 se instala todo
 - Installing collected packages: shiboken6, PySide6-Essentials, PySide6-Addons, pyside6
 - Successfully installed PySide6-Addons-6.7.2 PySide6-Essentials-6.7.2 pyside6-6.7.2 shiboken6-6.7.2

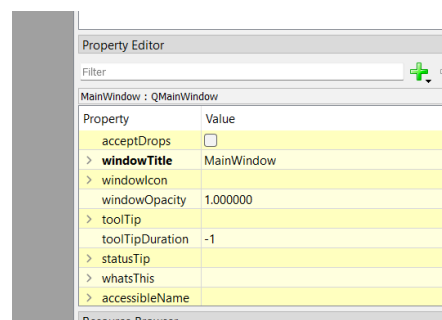
```
PS C:\workspace\python2024> pip install pyside6
Collecting pyside6
  Using cached PySide6-6.7.2-cp39-abi3-win_amd64.whl.metadata (5.5 kB)
Collecting shiboken6==6.7.2 (from pyside6)
  Using cached shiboken6-6.7.2-cp39-abi3-win_amd64.whl.metadata (2.6 kB)
Collecting PySide6-Essentials==6.7.2 (from pyside6)
  Using cached PySide6-Essentials-6.7.2-cp39-abi3-win_amd64.whl.metadata (3.8 kB)
Using cached PySide6-6.7.2-cp39-abi3-win_amd64.whl (537 kB)
Using cached PySide6-Addons-6.7.2-cp39-abi3-win_amd64.whl (123.0 MB)
Using cached PySide6-Essentials-6.7.2-cp39-abi3-win_amd64.whl (78.9 MB)
Using cached shiboken6-6.7.2-cp39-abi3-win_amd64.whl (1.1 MB)
Installing collected packages: shiboken6, PySide6-Essentials, PySide6-Addons, pyside6
Successfully installed PySide6-Addons-6.7.2 PySide6-Essentials-6.7.2 pyside6-6.7.2 shiboken6-6.7.2
PS C:\workspace\python2024> designer
PS C:\workspace\python2024> pyside6-designer
```

- si hay errores : instalamos extension PYQT integration y QT for python desde la pagina de QT en CMD como ADMINISTRADOR

- y abrimos el qt desde terminal con
 - pyside6-designer



- DENTRO de QT
- Ya podemos empezar a creat nuestras ventanas en el designer :
 - newForm
 - Main Window
 - CREATE
 - Cambio el nombre de la Ventana en propiedades
 - window title :.....nombre_Ventana....
 - Windowicon
 - fontelijo letra

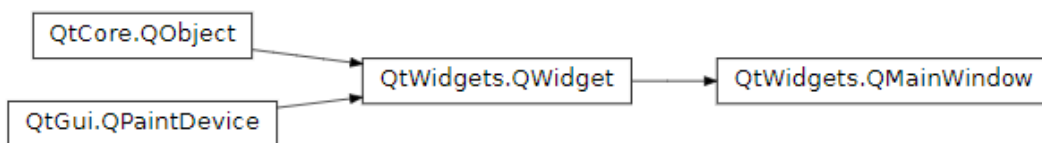


VENTANA PRINCIPAL

La Main window es la Pantalla Principal, como su nombre lo indica , me permite poner los menu de acceso y algun grafico

class QMainWindow

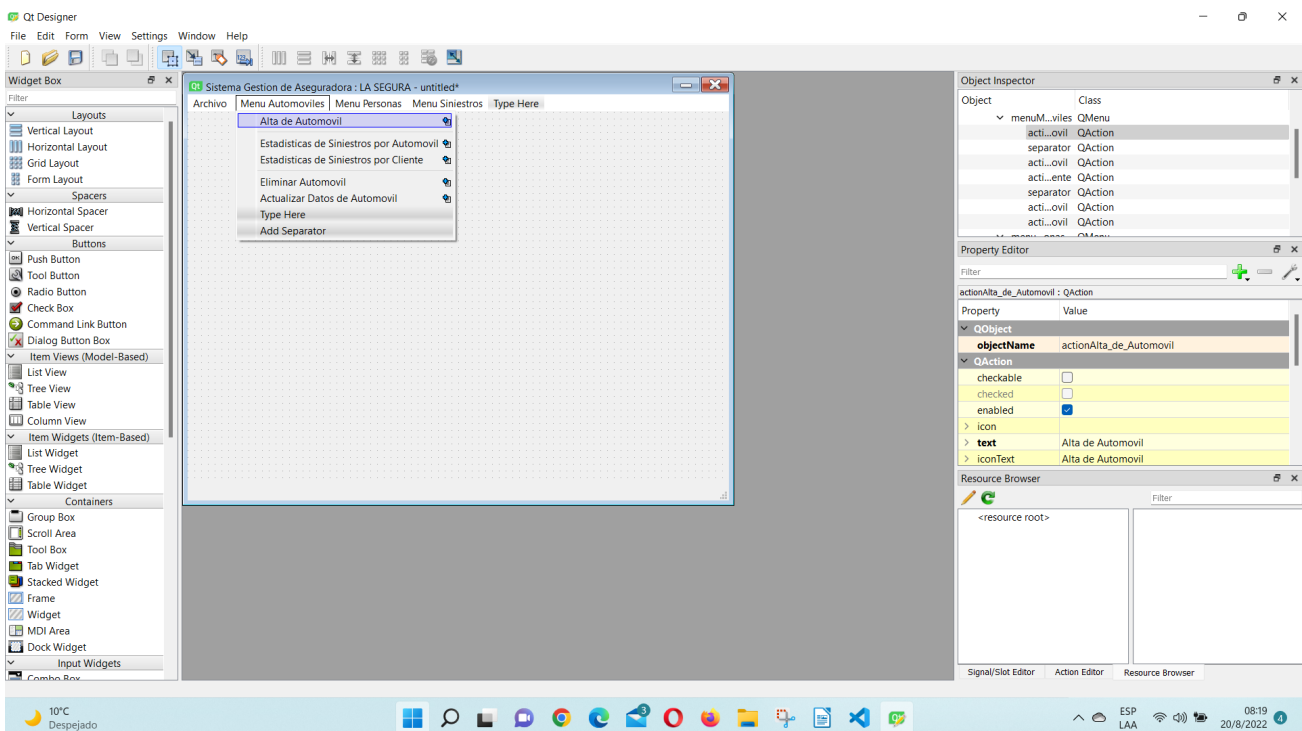
The `QMainWindow` class provides a main application window. [More...](#)



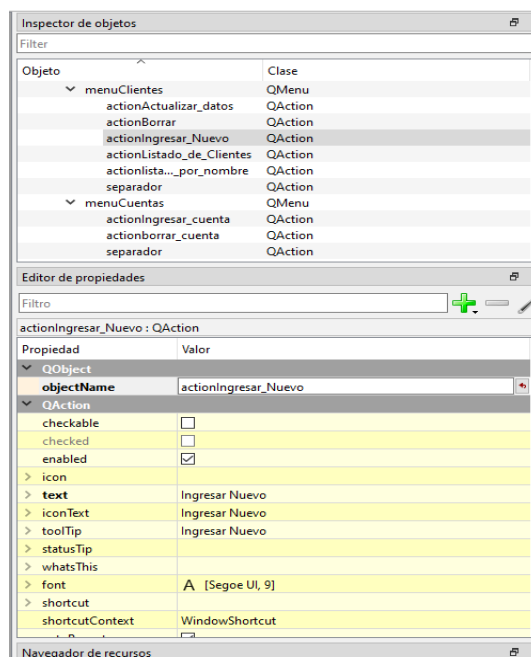
<https://doc.qt.io/qtforpython-6/PySide6/QtWidgets/QMainWindow.html>

MENUS

- Luego donde dice **Type Here** comienzo a Indicar mis menus.....



- Los menus tienen una clase y propiedades
- Respecto de la clase
- cada Menu tiene clase QMenu
- y
- cada submenu se transforma en un objeto de clase QAction
- asi mismo cada objeto tiene propiedades que se pueden modificar para el usuario



- Incluso por ejemplo al comando salir le puedo asignar un **SHORTCUT** como <ctrl><q> a travez de la propiedad **SHORTCUT**

actionSalir : QAction	
Property	Value
> statusTip	
> whatsThis	
> font	A [MS Shell Dlg 2, 8]
> shortcut	Ctrl+Q
shortcutContext	WindowShortcut
autoRepeat	<input checked="" type="checkbox"/>

- <CTRL><R> me permite ver como queda
- **lo guardo en el directorio elegido CAPA VISTA y vemos un archivo XML**
- para ello instalar **extension xml tools**



Veamos ahora si funciona

- Creamos el archivo en directorio Bankito
 - SistemaBankito.py
 - copiamos el siguiente codigo
- y ejecutamos el programa que inicia la ejecucion del sistema el cual contiene el sgte codigo

```
import sys
from PySide6.QtWidgets import QApplication, QMainWindow
from PySide6.QtCore import QFile
from capa_Vista.ventanappal6_ui import Ui_MainWindow
from capa_Datos import *
```

```
class MainWindow(QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
```

```
app=QApplication()
window=MainWindow()
window.show()
sys.exit(app.exec())
```

y se abre la ventana!!!!!!!!!!!!!!!!!!!!!!

Comenzamos con las respuestas del **MENU** QUE SE ENCUENTRA LA VENTANA PRINCIPAL

La mas fácil **SALIR** o sea terminar el sistema.....

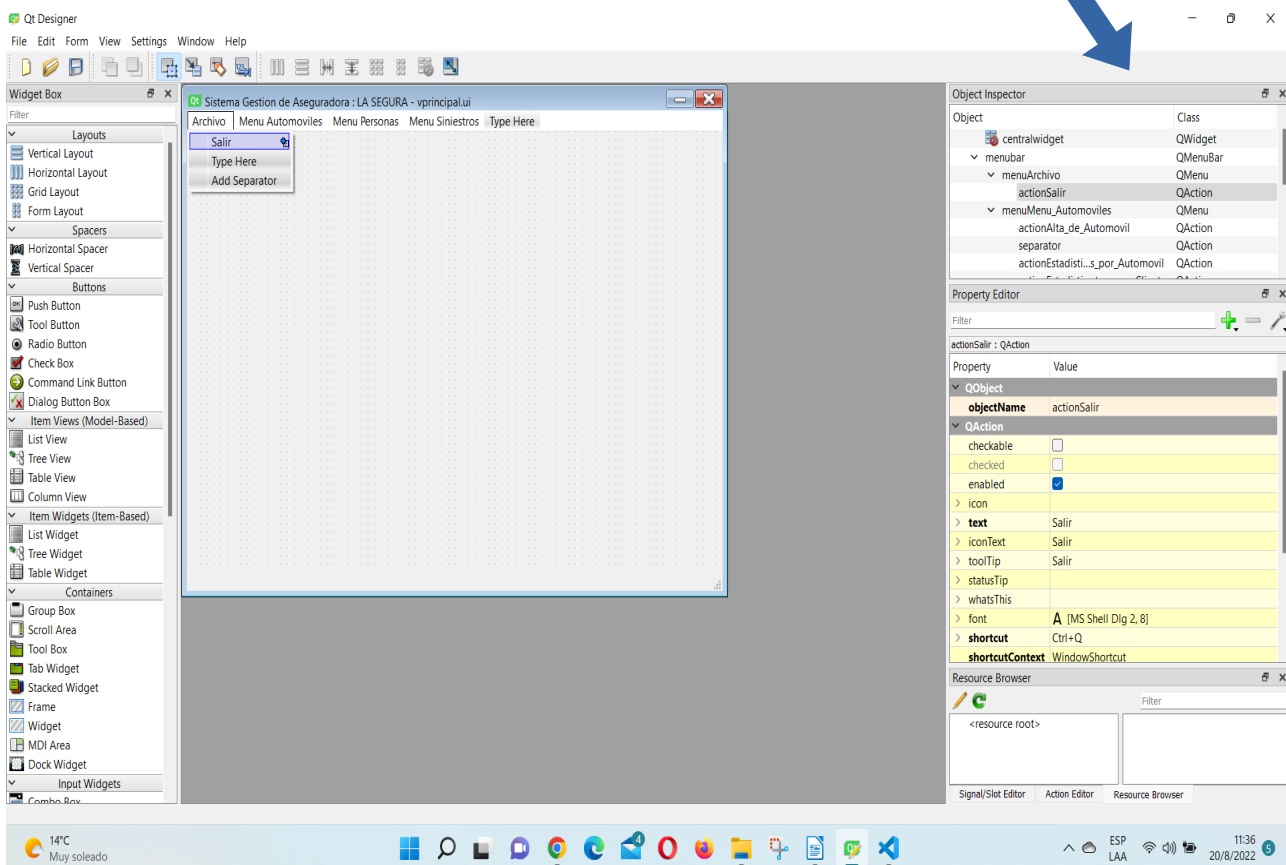
primero busco el nombre que le asignamos en qt **actionSALIR**

segundo lo conectamos la accion a una fn

```
self.actionSALIR.triggered.connect(self.salir)
```

tercero definimos la funcion

```
def salir(self):
    sys.exit()
```



Hay que comenzar a ejecutar **acciones en funcion de la opcion de menu elegida** para ello **conectamos una opción del menú con una función en el programa.**

```
from PySide6.QtWidgets import QApplication, QMainWindow
from capa_Vista.ventanappal6_ui import Ui_MainWindow
import sys
```

```
class MainWindow(QMainWindow, Ui_MainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.setupUi(self)
```

```
#Conectar las acciones a los metodos
```

AQUI Conectamos una ACCION de la UI con el CODIGO PYTHON que ACTUARÁ EN CONSECUENCIA a través del triggered=DISPARADOR

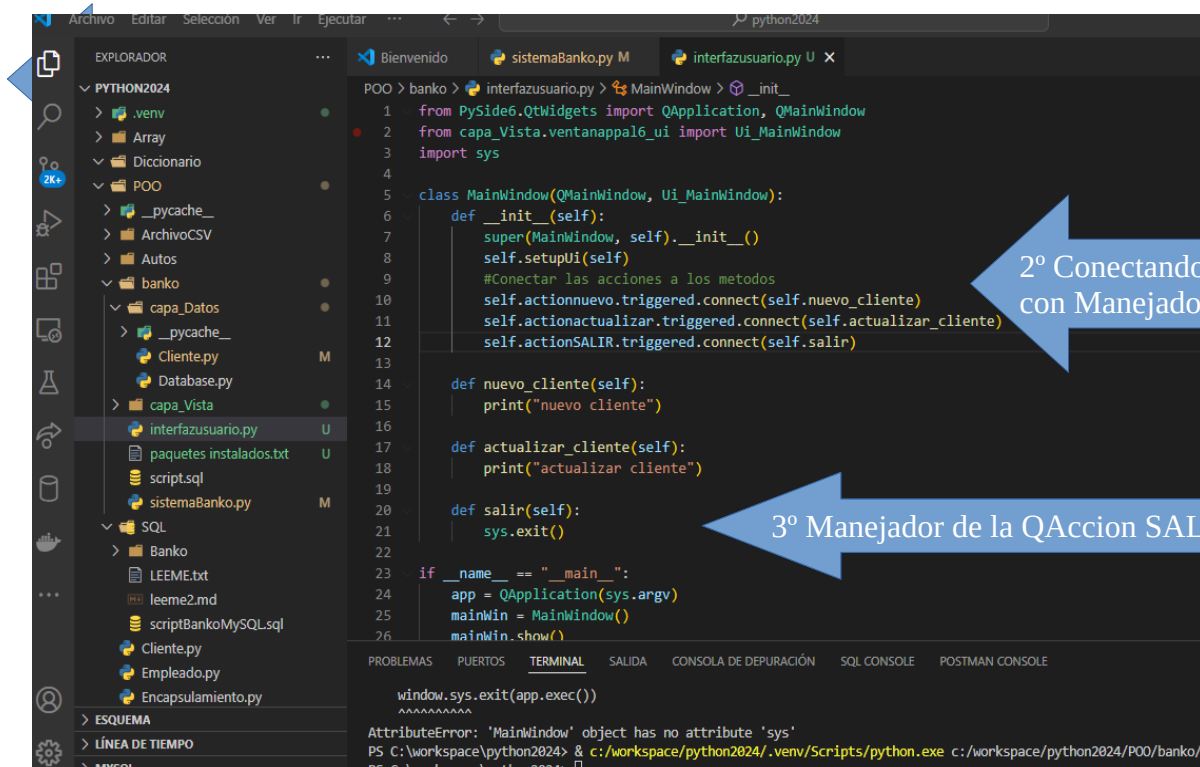
```
self.actionnuevo.triggered.connect(self.nuevo_cliente)
self.actionactualizar.triggered.connect(self.actualizar_cliente)
```

```
def nuevo_cliente(self):
    print("nuevo cliente")
```

```
def actualizar_cliente(self):
    print("actualizar cliente")
```

AQUI codificamos FUNCION PYTHON que SEA CAPAZ DE MANEJAR esa ACCION
Comenzamos con un simple print

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    mainWin = MainWindow()
    mainWin.show()
    sys.exit(app.exec())
```

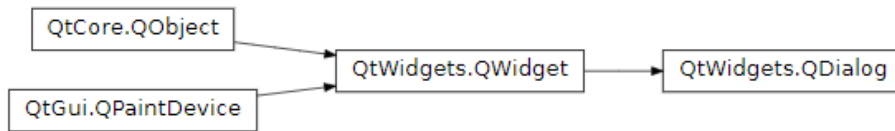


VENTANAS DIALOG

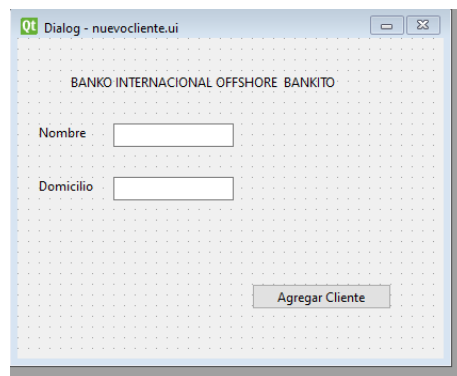
<https://doc.qt.io/qtforpython-6/PySide6/QtWidgets/QDialog.html>

class QDialog

The `QDialog` class is the base class of dialog windows. [More...](#)



Inherited by: `QWizard`, `QProgressDialog`, `QMessageBox`, `QInputDialog`, `QFontDialog`, `QErrorMessage`, `QColorDialog`, `QPrintPreviewDialog`, `QPageSetupDialog`, `QAbstractPrintDialog`, `QPrintDialog`, `QFileDialog`



Es muy importante que observar que por cada ventana :

Primero: con QT Designer creo una ventana obteniendo en código XML el archivo `nuevocliente.ui`

Segundo: guardar `ventana.ui` en la capa vista, con esto, automáticamente PySide6 crea la clase python que la maneja, hablamos del Archivo `nuevocliente_ui.py`

Tercero: en nuestro código generaremos una clase que herede de la que generó PySide6 para interactuar con nuestro código `ventanaNuevoCliente.py`

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
<class>Dialog</class>
<widget class="QDialog" name="Dialog">
<property name="geometry">
<rect>
<x>0</x>
<y>0</y>
<width>400</width>
<height>300</height>
</rect>
</property>
<property name="windowTitle">
<string>Dialog</string>
</property>
<widget class="QLabel" name="label">
<property name="geometry">
<rect>
<x>50</x>
<y>20</y>
<width>321</width>
<height>41</height>
</rect>
</property>
<property name="text">
<string>BANKO INTERNACIONAL
OFFSHORE BANKITO</string>
</property>
</widget>
```

```
class Ui_Dialog(object):
def setupUi(self, Dialog):
if not Dialog.setObjectName():
Dialog.setObjectName(u"Dialog")
Dialog.resize(400, 300)
self.label = QLabel(Dialog)
self.label.setObjectName(u"label")
self.label.setGeometry(QRect(50, 20, 321, 41))
self.label_2 = QLabel(Dialog)
self.label_2.setObjectName(u"label_2")
self.label_2.setGeometry(QRect(20, 80, 49, 16))
self.label_3 = QLabel(Dialog)
self.label_3.setObjectName(u"label_3")
self.label_3.setGeometry(QRect(20, 130, 51, 16))
self.txtNombre = QLineEdit(Dialog)
self.txtNombre.setObjectName(u"txtNombre")
self.txtNombre.setGeometry(QRect(90, 80, 113, 22))
self.txtDomicilio = QLineEdit(Dialog)
self.txtDomicilio.setObjectName(u"txtDomicilio")
self.txtDomicilio.setGeometry(QRect(90, 130, 113, 22))
self.btnAgregarCliente = QPushButton(Dialog)
self.btnAgregarCliente.setObjectName(u"btnAgregarCliente")
self.btnAgregarCliente.setGeometry(QRect(220, 230, 131, 24))
self.retranslateUi(Dialog)
QMetaObject.connectSlotsByName(Dialog)
# setupUi
def retranslateUi(self, Dialog):
Dialog.setWindowTitle(QCoreApplication.translate("Dialog",
u"Dialog", None))
```

```
class VentanaNuevoCliente(QDialog, Ui_Dialog):
def __init__(self):
super(VentanaNuevoCliente, self).__init__()
self.setupUi(self)
# Conectar el botón "Agregar Cliente" con su función
self.btnAgregarCliente.clicked.connect(self.agregar_cliente)

def agregar_cliente(self):
# Obtener el texto de los campos de texto
nombre = self.txtNombre.text()
domicilio = self.txtDomicilio.text()

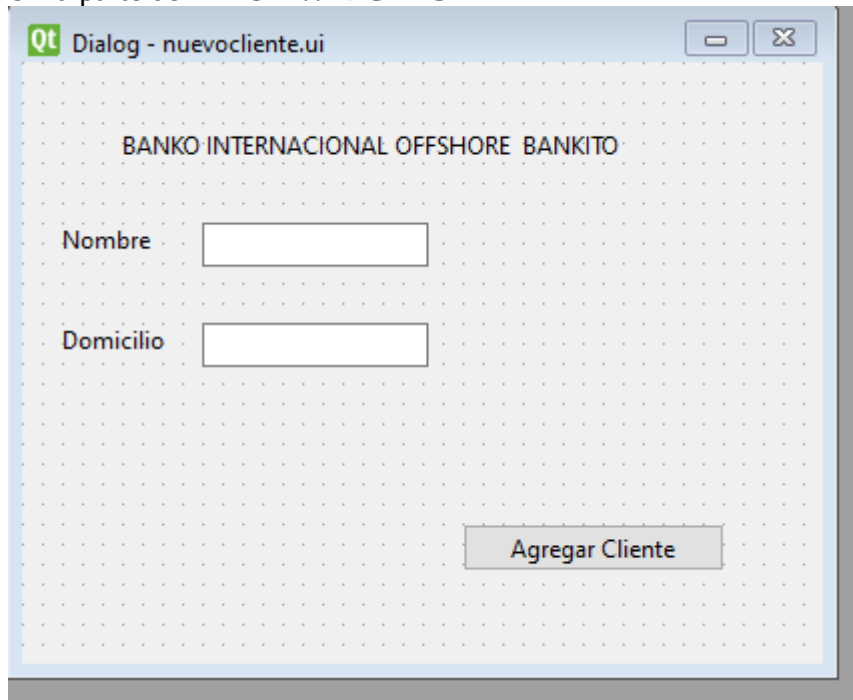
# Validar que los campos no estén vacíos
if nombre and domicilio:
# agregar el cliente a una base de datos
cliente=ClienteDB()
cliente.nuevo(6,nombre,domicilio)
print(f"Cliente agregado: {nombre}, {domicilio}")

# Mostrar un mensaje de confirmación
QMessageBox.information(self, "Cliente Agregado", "El cliente
ha sido agregado exitosamente.")
# Cerrar la ventana después de agregar el cliente
self.accept()
else: # Mostrar un mensaje de error si los campos están vacíos
QMessageBox.warning(self, "Campos vacíos", "Por favor,
complete todos los campos.)
```

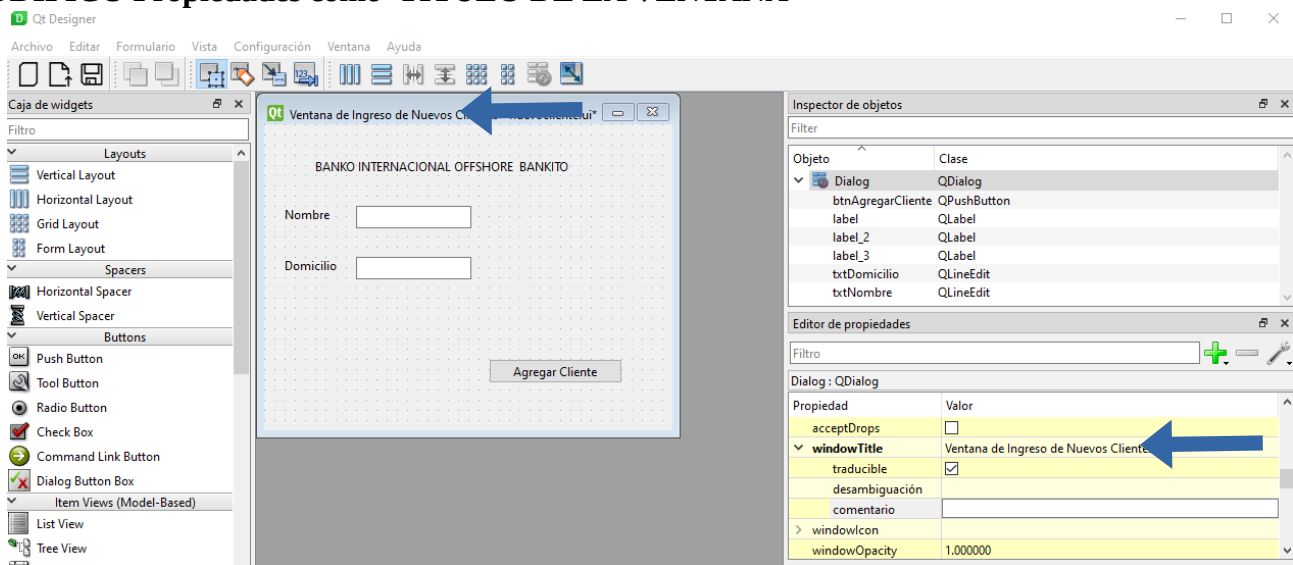
Capa negocio

EMPEZAMOS

- Es hora de comenzar a definir otras ventanas y llamar a nuestra capa de negocios
- Creamos una o muchas ventanas para poder manejar CRUD de CLIENTES
- **CRUD**
 - **CREATE**
 - **READ**
 - **UPDATE**
 - **DELETE**
- RECORDEMOS PRIMERO diseñamos en QT DESIGNER y luego las CLASES
- En QT diseñamos una ventana que sea de nuestro agrado con labels e Inputs etc.....
- en la parte de **DISPLAY WIDGETS** encontramos labels
- los Line Edit en la parte de **INPUT WIDGETS**

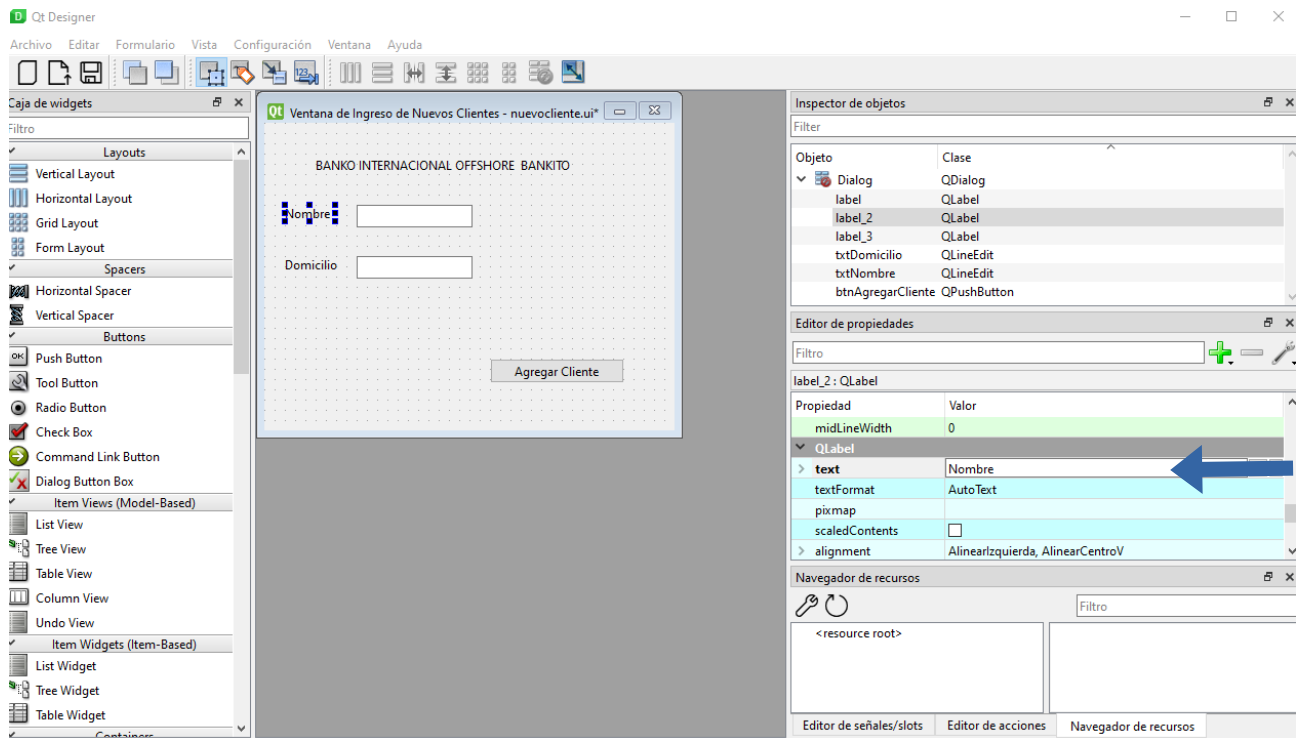


MODIFICO Propiedades como TITULO DE LA VENTANA



Modifico elementos

cada modificación tendrá su correspondiente reaccion en UI y la CLASE



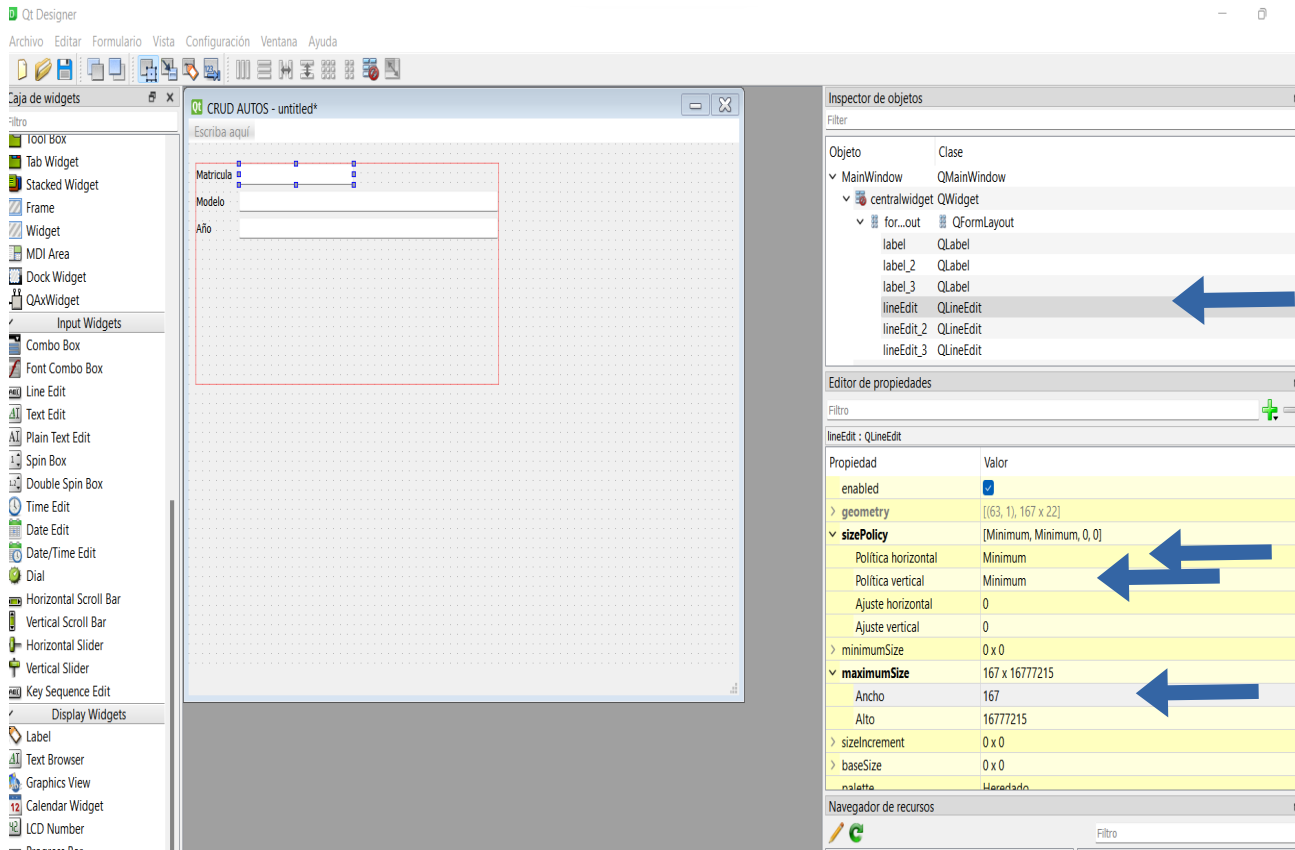
nuevocliente.ui

```
<property name="text">
<string>Nombre</string>
</property>
</widget>
<widget class="QLabel" name="label_3">
<property name="geometry">
<rect>
<x>20</x>
<y>130</y>
<width>51</width>
<height>16</height>
</rect>
</property>
```

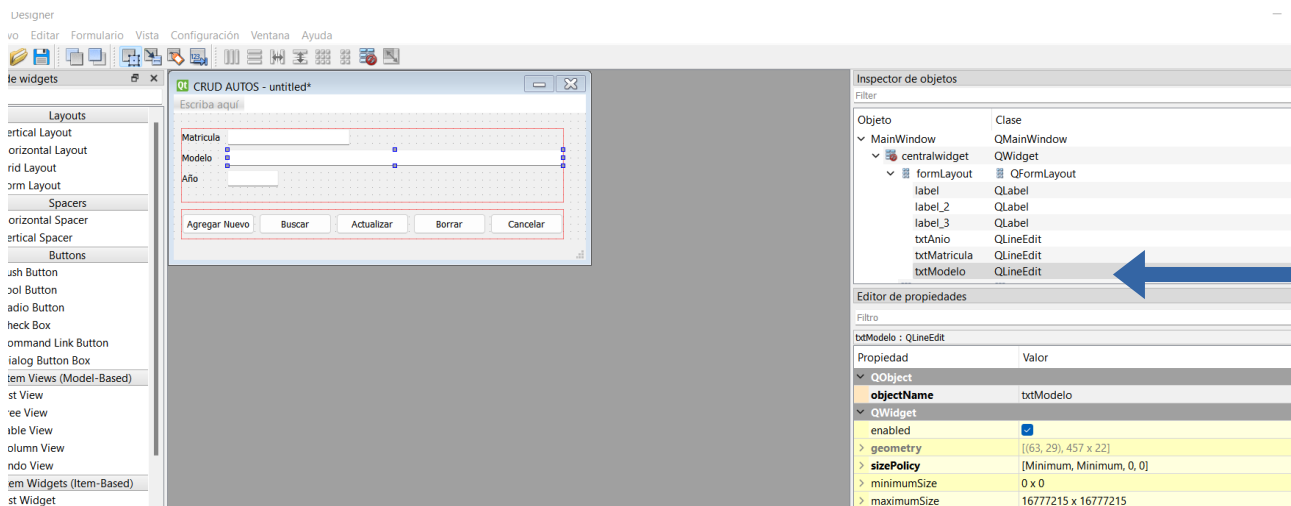
nuevocliente_ui.py

```
self.label_2 = QLabel(Dialog)
self.label_2.setObjectName(u"label_2")
self.label_2.setGeometry(QRect(20, 80, 49, 16))
```

- Podemos cambiar **propiedades**
- cada cambio se vera reflejado en **nuevocliente.ui** y **nuevocliente_ui.py**
-



- Agregamos un Horizontal Layout
- agregamos push boton
 - propiedad **objectName = btnCancelar** propiedad objectName es importante en los elementos como un **button** ya que ante un evento como un **CLICK** sobre el boton reacciona ejecutando un **CODIGO** por lo tanto tendremos nosotros que codificarlo a traves de la definicion del trigger y el codigo de la fn.
 - propiedad **text=Cancelar** la propiedad text refiere al texto que se muestra en la pantalla
- Continuaremos agregando elementos al todos los botones lo mismo los lineEdit le ponemos nombre **object name = txtModelo**



El siguiente nivel es hacer que la opcion del MENU abra una ventana que a su vez tenga actions y manejadores

REPASEMOS

- En la MAIN WINDOW (VENTANA PPAL) establecimos MENUS cada MENU establece una QAction x ej QAction: accion_nuevoCliente
- La respuesta a esta accion es que se abra una ventana que maneje lo necesario para crear un cliente (la ventana no es una ventana ppal, es una ventana de DIALOGO con el usuario) por tanto:
 - creamos un DIALOG x ej: VentanaXXX
 - Obtenemos el archivo VentanaXXX.ui
 - AL copiarlo en la CAPA VISTA se crea automaticamente por PySide6 la clase VentanaXXX_ui.py (pyside6-uic VentanaListadoClientes.ui -o VentanaListadoClientes_ui.py)
 - CREAMOS NOSOTROS una CLASE QUE HEREDA DE QDialog y de la que genero PYSIDE6 VentanaXXX_ui.py con los manejadores y triggered´s necesarios para manejar los eventos disparados en esa ventana como por ejemplo el bonto GUARDAR o CANCELAR
 - el manejador puede hacer uso de las clases definimos en la capa negocio

Ventana Ppal- Main Window

D Sistema de Control de Bankito - [Previsualizar] - Qt Designer

Cliente Cuenta Prestamo Salida

```
from PySide6.QtCore import (QCoreApplication, QDate, QDateTime, QLocale,
    QMetaObject, QObject, QPoint, QRect,
    QSize, QTime, QUrl, Qt)
from PySide6.QtGui import (QAction, QBrush, QColor, QConicalGradient,
    QCursor, QFont, QFontDatabase, QGradient,
    QIcon, QImage, QKeySequence, QLinearGradient,
    QPainter, QPalette, QPixmap, QRadialGradient,
    QTransform)
from PySide6.QtWidgets import (QApplication, QMainWindow, QMenu, QMenuBar,
    QSizePolicy, QStatusBar, QWidget)

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        if not MainWindow.setObjectName():
            MainWindow.setObjectName(u"MainWindow")
            MainWindow.resize(1105, 680)
            MainWindow.setStyleSheet(u"MainWindow {
background-image: url(C:/workspace/python2024/P00/Bankito/imagenes/banco.jpg);
background-repeat: no-repeat;
background-position: center;
}")

        self.actionNuevo_Cliente = QAction(MainWindow)
        self.actionNuevo_Cliente.setObjectName(u"actionNuevo_Cliente")
        self.actionActualizar_Cliente = QAction(MainWindow)
        self.actionActualizar_Cliente.setObjectName(u"actionActualizar_Cliente")
        self.actionBorrar_Cliente = QAction(MainWindow)
        self.actionBorrar_Cliente.setObjectName(u"actionBorrar_Cliente")
        self.actionBuscar_Cliente = QAction(MainWindow)
        self.actionBuscar_Cliente.setObjectName(u"actionBuscar_Cliente")
        self.actionListar_todos_los_clientes = QAction(MainWindow)
        self.actionListar_todos_los_clientes.setObjectName(u"actionListar_todos_los_clientes")
        self.actionNuevo_Prestamo = QAction(MainWindow)
        self.actionNuevo_Prestamo.setObjectName(u"actionNuevo_Prestamo")
        self.actionSALIR = QAction(MainWindow)
```

```
ESQA2024 > POO > Bankito > capa_vista > > ventanaPpal.ui
2 <ui version="4.0">
4 <widget class="QMainWindow" name="MainWindow">
16 <property name="toolTip">
17 <string>ventana de inicio</string>
18 </property>
19 <property name="styleSheet">
20 <string>notr="true">QMainWindow {
21 background-image: url(C:/workspace/python2024/P00/Bankito/imagenes/banco.jpg);
22 background-repeat: no-repeat;
23 background-position: center;
24 </string>
25 </property>
26 <widget class="QWidget" name="centralwidget"/>
27 <widget class="QMenuBar" name="menubar">
28 <property name="geometry">
29 <rect>
30 <x0</x>
31 <y0</y>
32 <width>1105</width>
33 <height>26</height>
34 </rect>
35 </property>
36 <widget class="QMenu" name="menuCliente">
37 <property name="title">
38 <string>Cliente</string>
39 </property>
40 <addaction name="actionNuevo_Cliente"/>
41 <addaction name="actionActualizar_Cliente"/>
42 <addaction name="separator"/>
43 <addaction name="actionBorrar_Cliente"/>
44 <addaction name="separator"/>
45 <addaction name="actionBuscar_Cliente"/>
46 <addaction name="actionListar_todos_los_clientes"/>
47 </widget>
48 <widget class="QMenu" name="menuCuenta">
49 <property name="title">
```

1

2

```

from PySide6.QtWidgets import QApplication, QMainWindow
from capa_vista.ventanaPpal_ui import Ui_MainWindow
from capa_vista.ventanaNuevoCliente import *

# from formNuevoCliente import *
import sys

class MainWindow(QMainWindow, Ui_MainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.setupUi(self)

        #Conectar las acciones del MENU a los metodos
        self.actionNuevo_Cliente.triggered.connect(self.nuevo_cliente)
        self.actionActualizar_Cliente.triggered.connect(self.actualizar_cliente)
        self.actionSALIR.triggered.connect(self.salir)

    def nuevo_cliente(self):
        # # Crear una instancia de la ventana "Nuevo Cliente"
        ventana = VentanaNuevoCliente()
        # Mostrar la ventana y esperar a que se cierre
        ventana.exec_()

    def actualizar_cliente(self):
        print("actualizar cliente")

    def salir(self):
        sys.exit()

```

VENTANA NUEVO CLIENTE como las acciones son en botones se dispara CLICKED

Sistema BANKITO - Ingreso de Nuevos clientes - [Previsualizar] - Qt Designer

Sistema Bankito

Id

Nombre y Apellido

Direccion

```

<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
<class>ventanaNuevoCliente</class>
<widget class="QDialog" name="ventanaNuevoCliente">
<property name="geometry">
<rect>
<x>0</x>
<y>0</y>
<width>772</width>
<height>537</height>
</rect>
</property>
<property name="windowTitle">
<string>Sistema BANKITO - Ingreso de Nuevos clientes</string>
</property>
<widget class="QLabel" name="label">
<property name="geometry">
<rect>
<x>250</x>
<y>30</y>
<width>311</width>
<height>81</height>
</rect>
</property>
<property name="font">
<font>
<family>Noto Sans Cond</family>
<pointsize>22</pointsize>
</font>
</property>
</widget>
</ui>

```

```

#####
## Form generated from reading UI file 'ventanaNuevoCliente.ui'
##
## Created by: Qt User Interface Compiler version 6.7.2
##
## WARNING! All changes made in this file will be lost when recompiling UI file!
#####

from PySide6.QtCore import (QCoreApplication, QDate, QDateTime, QMetaObject, QObject, QPoint, QRect, QSize, QTime, QUrl, Qt)
from PySide6.QtGui import (QBrush, QColor, QConicalGradient, QCursor, QFont, QFontDatabase, QGradient, QIcon, QImage, QKeySequence, QLinearGradient, QPainter, QPalette, QPixmap, QRadialGradient, QTransform)
from PySide6.QtWidgets import (QApplication, QDialog, QLabel, QLineEdit, QPushButton, QSizePolicy, QWidget)

class Ui_ventanaNuevoCliente(object):
    def setupUi(self, ventanaNuevoCliente):
        if not ventanaNuevoCliente.setObjectName():
            ventanaNuevoCliente.setObjectName(u"ventanaNuevoCliente")
            ventanaNuevoCliente.resize(772, 537)
            self.label = QLabel(ventanaNuevoCliente)
            self.label.setObjectName(u"label")
            self.label.setGeometry(QRect(250, 30, 311, 81))
            font = QFont()
            font.setFamilies([u"Noto Sans Cond"])
            font.setPointSize(22)
            self.label.setFont(font)
            self.label_2 = QLabel(ventanaNuevoCliente)
            self.label_2.setObjectName(u"label_2")
            self.label_2.setGeometry(QRect(70, 290, 55, 16))
            font1 = QFont()

```

```

from PySide6.QtWidgets import QApplication, QMainWindow
from capa_vista.ventanaNuevoCliente_ui import Ui_ventanaNuevoCliente
from PySide6.QtWidgets import QDialog, QMessageBox
from capa_Negocio.ClienteDB import *

import sys

class VentanaNuevoCliente(QDialog, Ui_ventanaNuevoCliente):
    def __init__(self):
        super(VentanaNuevoCliente, self).__init__()
        self.setupUi(self)

        # Conectar el botón "Agregar Cliente" con su función
        self.btnGuardar.clicked.connect(self.agregar_cliente)
        self.btnCancel.clicked.connect(self.close)

    def agregar_cliente(self):
        # Obtener el texto de los campos de texto
        nombre = self.txtNombre.text()
        domicilio = self.txtDomicilio.text()

        # Validar que los campos no estén vacíos
        if nombre and domicilio:
            # Aquí podrías agregar el cliente a una base de datos o realizar otras acciones
            cliente=ClienteDB()
            id=cliente.proximoId()
            cliente.crear_nuevo(id, nombre, domicilio)

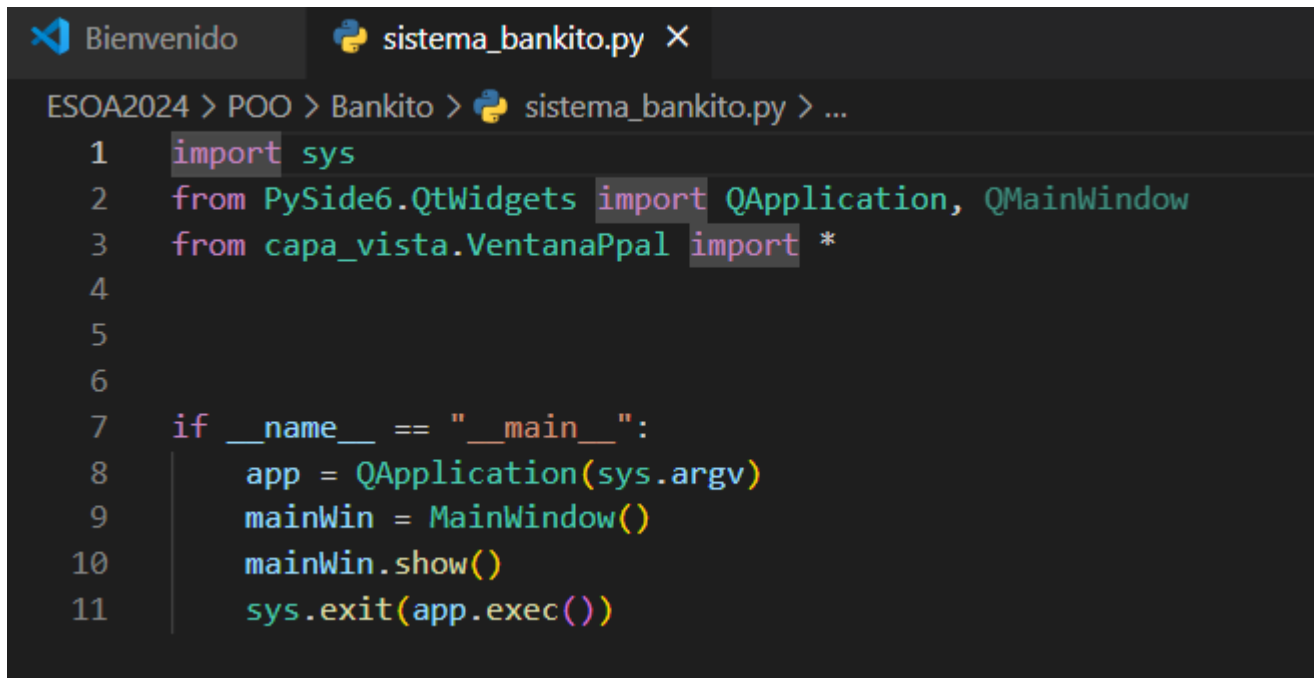
            print(f"Cliente agregado:{id} {nombre}, {domicilio}")

            # Mostrar un mensaje de confirmación
            QMessageBox.information(self, "Cliente Agregado", "El cliente ha sido agregado correctamente.")

            # Cerrar la ventana después de agregar el cliente
            self.accept()
        else:
            # Mostrar un mensaje de error si los campos están vacíos
            QMessageBox.warning(self, "Error", "Los campos no deben estar vacíos.")

```

TODO EL SISTEMA se reduce a levantar la ventana ppal



The screenshot shows a code editor with a dark theme. The top bar has two tabs: 'Bienvenido' and 'sistema_bankito.py'. The breadcrumb navigation shows the path: 'ESOA2024 > POO > Bankito > sistema_bankito.py > ...'. The code is as follows:

```
1 import sys
2 from PySide6.QtWidgets import QApplication, QMainWindow
3 from capa_vista.VentanaPpal import *
4
5
6
7 if __name__ == "__main__":
8     app = QApplication(sys.argv)
9     mainWin = MainWindow()
10    mainWin.show()
11    sys.exit(app.exec())
```

SUBAMOS OTRO NIVEL LISTADOS

Crear ejecutable

Intalar el instalador (vaya la redundancia)
`pip install pyinstaller`

luego me posiciono en el directorio dode esta la app y ejecuto

```
PS D:\Proyectos\VSCode2022> cd claseU1
PS D:\Proyectos\VSCode2022\ClaseU1> cd aseguradora
PS D:\Proyectos\VSCode2022\ClaseU1\Aseguradora> dir

Directory: D:\Proyectos\VSCode2022\ClaseU1\Aseguradora

Mode                LastWriteTime         Length Name
----                -
d----             20/8/2022   10:17             Capa_Negocios
d----             21/8/2022   22:24             Capa_Vista
d----             21/8/2022   19:09             CapaNegocios
-a---             21/8/2022   22:54          2529 SistemaAseguradora.py
```

`pyinstaller --windowed --onefile tuappp.py`

si tengo un icono

`pyinstaller --windowed --onefile --icon=./logo.ico tuappp.py`

carpeta dist esta el exe

Algunos materiales PYTHON

- https://perso.limsi.fr/poital/_media/python:cours:mementopython3-espanol.pdf
- http://do1.dr-chuck.com/pythonlearn/ES_es/pythonlearn.pdf
- <https://argentinaenpython.com/quiero-aprender-python/TutorialPython3.pdf>
-

PyQT5 /PySide6

Referencia rápida:

- <https://doc.qt.io/qtforpython-6/PySide6/QtWidgets/index.html>
- <https://www.qt.io/>
- <https://build-system.fman.io/pyqt5-tutorial>
- <https://wiki.python.org/moin/PyQt>
- <https://build-system.fman.io/docs/>
- <https://build-system.fman.io/qt-designer-download>
- <https://www.youtube.com/watch?v=5a7H7y0a5yc>
-

MySQL-Connector-Python:

- <https://dev.mysql.com/downloads/connector/python/>
- <https://pypi.org/project/mysql-connector-python/>
- <https://www.mysqltutorial.org/getting-started-mysql-python-connector/>

QT CAMBIAR FONDO

Cambiar color de fondo e imagen = STYLESHEET

Pasos para poner una imagen de fondo en Qt Designer:

1. **Abrir el Qt Designer** y cargar tu archivo `.ui` (o crear uno nuevo).
2. Selecciona la **ventana principal** o el widget sobre el cual quieres aplicar la imagen de fondo.
3. En la parte derecha, en la **propiedad "stylesheet"** del objeto seleccionado, haz clic en el botón de tres puntos (`...`) para abrir el editor de hojas de estilo.
4. En el editor que se abre, añade el siguiente código:

```
css
Copiar código
QMainWindow {
    background-image: url(ruta/a/tu_imagen.jpg);
    background-repeat: no-repeat;
    background-position: center;
}
```


5. Haz clic en **Aceptar** para aplicar los cambios y Guarda el archivo `.ui` y carga el diseño en tu aplicación con el código generado automáticamente.

OTROS WIDGETS como calendar etc

Qt BANCO LOS ALERCES - Pantalla Principal - BancoLosALERCES.ui

Cientes Sucursales Escriba aquí

Bienvenido al
Banco LOS ALERCES
su banco amigo
de los alerces y
del bolsillo del gerente



☐ Eleccion1 ☐ Opcion1
☐ Eleccion2 ☐ Opcion2
☐ Eleccion3 ☐ Opcion3
☐ Eleccion4

agosto 2023

	lu.	ma.	mi.	ju.	vi.	sá.	do.
31	31	1	2	3	4	5	6
32	7	8	9	10	11	12	13
33	14	15	16	17	18	19	20
34	21	22	23	24	25	26	27
35	28	29	30	31	1	2	3

Qt Banco LOS ALERCES - CLIENTES - AlercesClientes.ui

Pantalla de Carga
de Nuevos Clientes

Cliente N° 1

Nombre Andrea

Calle ESPORA 12 - Punta Alta -Pcia BsAs

0

Aceptar Cancelar

LISTADOS

1 Mostrar Datos en pantalla

Tenemos 2 problemas :

1.1 Traer Datos desde el BackEnd => `capa_datos.CienteDB()`

1.2 Utilizar un widgets para mostrarlo en la pantalla. Podemos utilizar diferentes widgets, cada uno tiene sus particularidades aunque basicamente el “poblar” o “llenado” se trata de recorrer la lista que viene del backend,

2 Exportar los datos a Excel, Word y PDF

Para cada formato de archivo, puedes utilizar diferentes bibliotecas:

- **Excel:** Usa `pandas` o `openpyxl`.
- **Word:** Usa `python-docx`.
- **PDF:** Usa `reportlab`.

Codigo EXCEL con librería pandas

```
import pandas as pd

def exportar_excel(self):
    # Lógica para exportar los datos a Excel usando libreria pandas
    print("Exportando a Excel...")
    clientesTabla = []
    for row in range(self.tablaClientes.rowCount()):
        cliente = {
            "Id": self.tablaClientes.item(row, 0).text(),
            "Nombre": self.tablaClientes.item(row, 1).text(),
            "Domicilio": self.tablaClientes.item(row, 2).text(),
        }
        clientesTabla.append(cliente)

    df = pd.DataFrame(clientesTabla) #crea un dataframe con los datos de la tabla
    df.to_excel("clientes.xlsx", index=False)
    print("Archivo Excel creado.")
```

WORD

```
from docx import Document

print("Exportando a WORD...")
doc = Document()
doc.add_heading('Listado de Clientes', 0)

for row in range(self.tablaClientes.rowCount()):
    id_cliente = self.tablaClientes.item(row, 0).text()
    nombre_cliente = self.tablaClientes.item(row, 1).text()
    domicilio_cliente = self.tablaClientes.item(row, 2).text()
    doc.add_paragraph(f"{id_cliente} - {nombre_cliente} - {domicilio_cliente}")
```

```
doc.save("clientes.docx")
print("Archivo Word creado.")
```

PDF

Vamos a usar la librería reportlab por lo que la instalamos primero con:

```
pip install reportlab
```

```
pip list
```

si da error probar con

```
py -m pip install reportlab
```

o sino desde el cmd en modo administrador

codigo:

```
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
```

```
def exportar_pdf(self):
    # Lógica para exportar los datos a PDF
    print("Exportando a PDF..")
```

```
c = canvas.Canvas("clientes.pdf", pagesize=letter)
c.drawString(100, 750, "Listado de Clientes")
```

```
y = 730
for row in range(self.tablaClientes.rowCount()):
    id_cliente = self.tablaClientes.item(row, 0).text()
    nombre_cliente = self.tablaClientes.item(row, 1).text()
    domicilio_cliente = self.tablaClientes.item(row, 2).text()
    c.drawString(100, y, f"{id_cliente} - {nombre_cliente} - {domicilio_cliente}")
    y -= 20

c.save()
print("Archivo PDF creado.")
```