CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

Título	Práctica #12 - Manual tecnico
Profesor	Karla Avila Cardenas
Equipo	Priscila Sarahí González González Oscar Eduardo Arámbula Vega
Carrera	Ingeniería Informática
Materia	Seminario de Solución de Problemas de Ingeniería de Software I
Clave	I5899
NRC	78114
Sección	D01
Calendario	2021B

Índice

Manual técnico

Estructura del proyecto

Lo primero es echar un vistazo a la estructura y organización de carpetas del proyecto. Esta es:

- code
  - assets
    - images
  - controllers
  - database
    - sql
  - ui_files
  - views

Code

Es el directorio raíz del proyecto y es donde se encuentra el archivo principal, el **main.py** que es el que uno invoca para lanzar la aplicación principal.

Assets

Esta carpeta está destinada a almacenar todos los recursos de media que necesite la aplicación. Se subdivide esta carpeta según el tipo de medio que se esté almacenando. También se encuentra el archivo **resources.qrc** que utiliza un formato de XML para describir los recursos y la herramienta Qt Designer pueda utilizarlos con facilidad.

Ui_files

Esta carpeta se destina a almacenar todos los archivos generados por la herramienta Qt Designer, los cuales utilizan un lenguaje de etiquetas pero que no será necesario que el programador interactúe con los mismos. El formado de nombres usado es:

<nombre_ventana>_window.ui

Views

Esta será destinada a almacenar las clases generadas, o mejor dicho, el código de python que fue compilado gracias a la herramienta **pyside2-uic**. Estas clases son mis ventanas que se diseñan en Qt Creator pero totalmente manejadas por Python. No haremos modificaciones en estos archivos debido a que estaran en constante reescritura completa por parte de **pyside2-uic** y se perderán los cambios.

La forma de nombrar a estos archivos es:

ui_<nombre_ventana>_window.py

Controllers

El módulo controllers es usado para almacenar todos los submódulos que contengan a las clases hijas que se derivan de las clases generadas por **pyside2-uic** y son completamente generadas por el desarrollador.

El formato de nombre usado es:

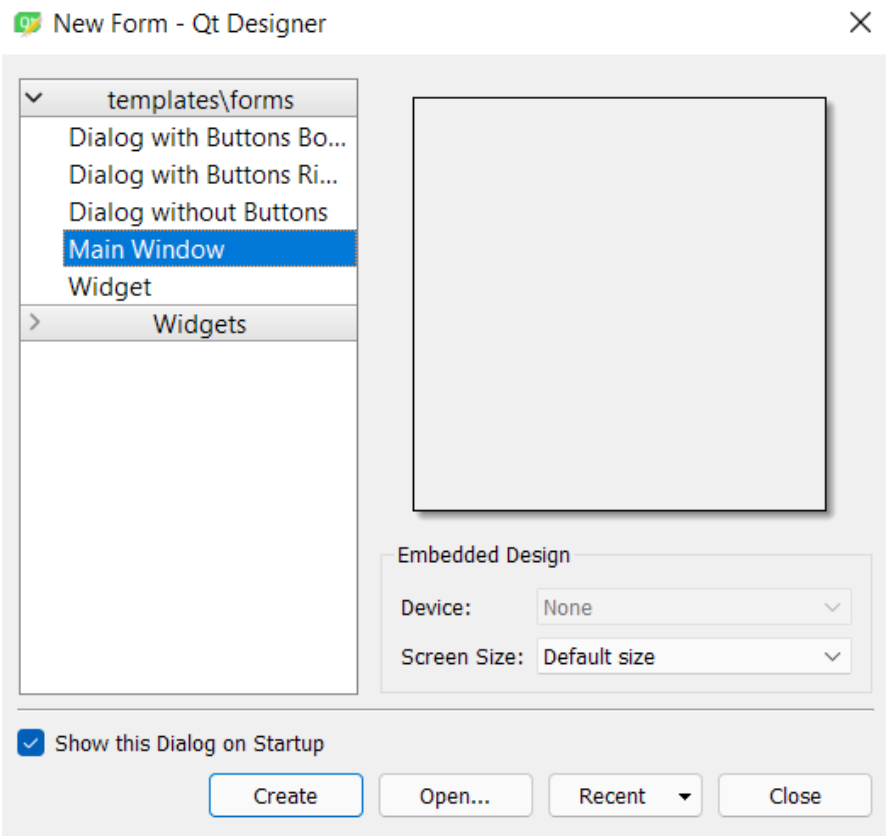
<nombre_ventana>_window.py

Database

Tiene consigo el conector a la base de datos usado para acceder a la misma, así como los submódulos para manejar cada tabla. Se encuentra dentro una carpeta llamada **sql** que tiene consigo la descripción de las tablas de la base de datos.

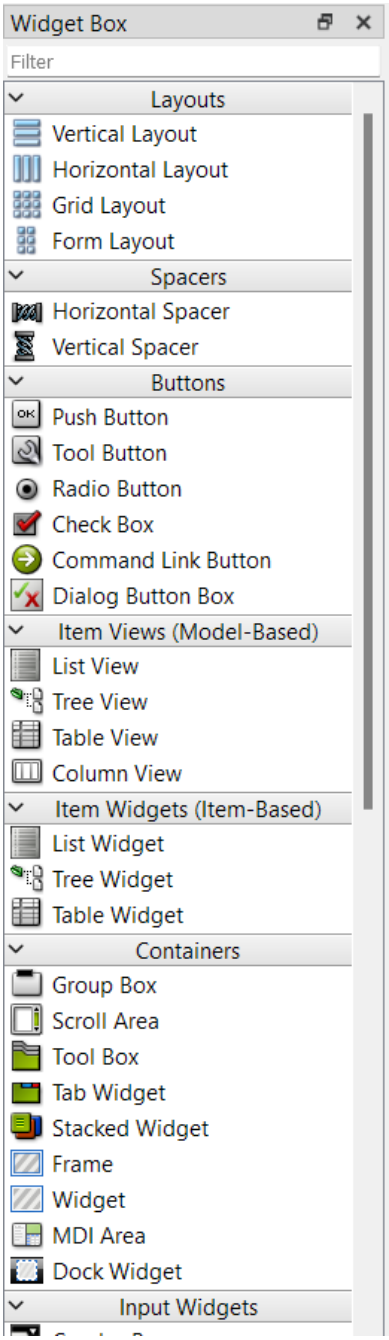
Generar una nueva ventana

Debe abrir la herramienta Qt Designer y se le mostrará lo siguiente:



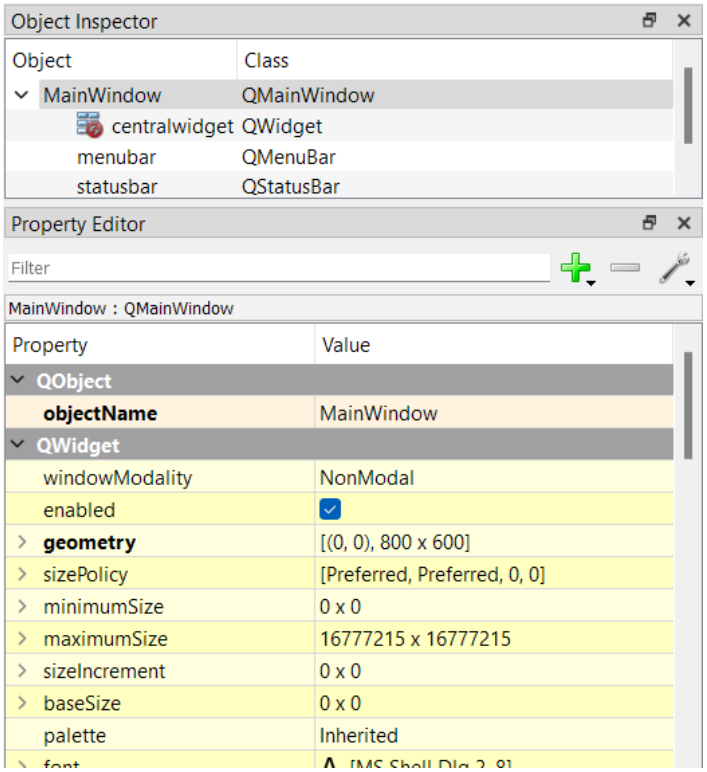
Ahí se le recomienda usar Main window pero puede usar la que se ajuste a sus necesidades.

Estas son los elementos de la interfaz que puede usar:



Y puede arrastrarlos hacia la ventana que está diseñando.

Para editar los atributos de un objeto, primero selecciónelo y después en la ventana derecha modifiquelo:



Una vez termine de editar y dar forma a su ventana, guarde el archivo en la carpeta **ui_files** con el formato de nombres correspondiente.

Convertir archivos .ui a .py

Para ello, utilice el siguiente comando:

pyside2-uic <nombre_archivo_origen>.ui -o <nombre_archivo_destino>.py

Este es específico para su uso en windows con CMD. Por ejemplo, si usa Powershell deberá usar **-o** en vez de **>,** esto para evitar problemas con la introducción de valores inválidos en el archivo. El comando varea dependiendo del sistema operativo.

Usar la nueva ventana

Primero asegúrese de importar adecuadamente el módulo, ejemplo:

```
from views.ui_agregar_producto_window import Ui_agregar_producto_window
```

Luego cree una clase que derive de la clase QMainWindow o QWidget, dependiendo de los escogido.

```
class AgregarProductoWindow(QMainWindow):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.ui = Ui_agregar_producto_window()
        self.ui.setupUi(self)
        self.setWindowFlag(Qt.Window)
```

De la clase que creo pyside2-uic que esta tendrá el nombre de la ventana o widget que le asignó en la interfaz. Deberá asignarle una instancia de la misma en un atributo que denominaremos **ui.**

Para invocar su ventana, basta con llamarla de la siguiente manera:

```
    from controllers.agregar_producto_window import AgregarProductoWindow
    window = AgregarProductoWindow(self)
    window.show()
```

Hacer uso de los módulos de base de datos

Para ellos siempre importe con **database/<nombre_modulo>.** Es a elección del programador si importa todo el módulo o solo las funciones que requiera pero se incita a importar solo las necesarias.

Aquí un ejemplo de cómo se hace uso de un modulo de agregar producto:

```
from database.producto import Producto, agregar, mostrar_todos
p = Producto()
p.nombre = self.ui.nombre_lineEdit.text()
p.precio = self.ui.precio_doubleSpinBox.value()
p.codigo = self.ui.codigo_lineEdit.text()

if len(p.nombre) == 0:
    self.ui.nombre_warning.setText("El nombre no puede estar en blanco")
elif p.precio <= 0:
    self.ui.precio_warning.setText("el precio no puede ser cero")
elif len(p.codigo) == 0:
    self.ui.codigo_warning.setText("El codigo no puede estar en blanco")
else:
    agregar(p.to_dict())
    self.ui.nombre_lineEdit.clear()
    self.ui.precio_doubleSpinBox.setValue(0.00)
    self.ui.codigo_lineEdit.clear()
    self.parent().mostrar_productos(mostrar_todos())
```

Expandir módulos de base de datos

Si quieres expandir la funcionalidad de un módulo, basta con agregar nuevas funciones. No se recomienda modificar las funciones ya establecidas a no ser que mejores la eficiencia pero sigan recibiendo la mismas entradas y produciendo las mismas salidas.

Crear tus propios módulos de la base de datos

Es sencillo, crea un nuevo archivo .py en la carpeta **database** y asignarle un nombre corto, que haga referencia a su tabla en la base de datos. Se recomienda construir tu modelo con las funciones atómicas de base de datos que fueron creadas pero puedes hacer uso del conector a tu gusto si así lo deseas.

Un ejemplo de la sencillez de un módulo usando las funciones de **coneccion.py**

```python
from database.conexion import insert, update, select, delete

def agregar(datos:dict):
    return insert("producto", **datos)

def mostrar_todos():
    return select("producto")

def editar(datos:dict, condicion:str=None, **condiciones):
    return update("producto", datos, condition=condicion, **condiciones)

def eliminar(condicion:str=None, **condiciones):
    return delete("producto", condition=condicion, **condiciones)

def mostrar(id_producto:int, *columnas):
    if len(columnas) != 1:
        columnas = ["nombre", "precio", "codigo"]
        consulta = select("producto", *columnas, id=id_producto)
        return consulta[0] if consulta else None
    else:
        consulta = select("producto", *columnas, id=id_producto)
        return consulta[0][columnas[0]] if consulta else None

def mostrar_con_nombre(nombre):
    return select("producto", condition=f"nombre LIKE '%{nombre}%'")

def mostrar_con_precio(precio):
    return select("producto", condition=f"precio = {precio}")

def mostrar_con_codigo(codigo):
    return select("producto", condition=f"codigo LIKE '%{codigo}%'")
```