**Desarrollo de Interfaces**

Icono

Descripción generada automáticamente con confianza media

**Trabajo realizado por Equipo 01:**

**Molero Marín, Juan**

**Rodríguez López, Julio**

**Martínez Lorda, Julián**

**I.E.S. Carrillo Salcedo**

**2º Desarrollo de Aplicaciones Multiplataforma**

**Curso 2024/2025**

**Tema 02 – Boletín 03**

**Contenido**

[**1.- Integración de la interfaz creada con QT Designer (archivo .ui) en Python: 3**](#_Toc181639616)

[**2.- Integración de Eventos, Señales y Slots a la Interfaz en Python: 6**](#_Toc181639617)

[**2.1.- Slots implementados en el Registro: 6**](#_Toc181639618)

[**2.2.- Slots implementados en el Login: 7**](#_Toc181639619)

[**3.- Integración de la Base de Datos dentro de la Aplicación: 9**](#_Toc181639620)

[**4.- Resumen sobre la Aplicación: 11**](#_Toc181639621)

[**5.- Despliegue de la Aplicación: 13**](#_Toc181639622)

[**5.1.- Entorno Virtual: 13**](#_Toc181639623)

[**5.2.- Convertir Archivo .ui en .py: 15**](#_Toc181639624)

[**5.3.- Docker y BBDD PostgreSQL: 17**](#_Toc181639625)

[**6.- Bibliografía: 21**](#_Toc181639626)

# **1.- Integración de la interfaz creada con QT Designer (archivo .ui) en Python:**

Nuestro primer paso ha sido acceder a la plataforma QT Designer, desde la cual obtendremos nuestros dos archivos .ui respectivamente: el de la ventana de Login y el de la ventana de Registro.

Una captura de pantalla de una computadora

Descripción generada automáticamentePara guardar el archivo seguiremos los siguientes pasos:

Una vez realicemos esto con nuestros dos archivos (‘login\_equipo\_1.ui y registro\_equipo\_1.ui’), mediante la consola Powershell accederemos a la ruta donde se encuentran, y posteriormente **se ejecutará el siguiente comando** para generar dos archivos .py dentro del mismo directorio:

**pyside6-uic ‘archivo.ui’ -o ‘archivo.py’ 🡪 Donde archivo.ui será el archivo que deseamos convertir y archivo.py el nuevo archivo generado, con el nombre que nosotros le especifiquemos. De esta forma ejecutamos los comandos de la siguiente forma:**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamenteEn nuestro caso concreto, los hemos generado dentro del directorio ‘views’ en el proyecto actual, por lo cual si los abrimos con VSC, y se crea un archivo Python independiente que los ejecute (main.py), podremos visualizar nuestras interfaces con éxito respectivamente. Cabe destacar que se le ha dado una mejor legibilidad y se les han incluido comentarios a ambos archivos, con el objetivo de poder trabajar posteriormente de forma más cómoda con ellos. Por tanto, aquí podemos ver el resultado:

El archivo que nos permitirá ejecutar cada una de las dos interfaces generadas, por otra parte, es éste:

Captura de pantalla de computadora

Descripción generada automáticamente

Finalmente, para comprobar que el proceso ha salido con éxito, debemos de irnos a la consola de nuestro VSC, y ejecutar el comando **python main.py** desde la raíz del proyecto, que es donde tenemos ubicado dicho archivo. Una vez ejecutado, se abrirá la ventana de Login de nuestra futura aplicación, y nuestro siguiente objetivo será el de conectarla a la ventana de Registro mediante slots y señales:

Captura de pantalla de computadora

Descripción generada automáticamente

# **2.- Integración de Eventos, Señales y Slots a la Interfaz en Python:**

En el caso de nuestro equipo, hemos conseguido la integración de eventos mediante la creación de Slots en nuestras clases login\_window y registro\_window, las cuáles manejan la lógica de las clases QT donde tenemos nuestras interfaces.

## **2.1.- Slots implementados en el Registro:**

Texto

Descripción generada automáticamenteDentro del Registro, hemos implementado dos Slots diferentes:

Con este primer Slot, ocultaremos la ventana actual donde nos encontramos, se creará posteriormente una instancia de la ventana de Registro, mediante parent (estamos heredando de QMainWindow), y se terminará mostrando.

Texto

Descripción generada automáticamente

Con este segundo Slot, procesaremos el registro de un usuario cuando haya rellenado todos los campos, y las contraseñas coincidan respectivamente. En caso afirmativo, dicho usuario quedará registrado dentro de la BBDD, y podrá tener después una posterior interacción iniciando sesión en la otra ventana. Cabe destacar que este Slot interactúa con un método del controlador (register\_user), parte que explicaremos más adelante, el cual es un módulo encargado de conectarnos la vista con los modelos y permitir que todo el flujo de la aplicación tenga éxito.

**Conexión de los Slots:**

Texto

Descripción generada automáticamenteEstos dos slots serán conectados en el constructor principal de la clase con los elementos ui respectivos de las vistas. Además, también es donde se creará la instancia del controlador:

## **2.2.- Slots implementados en el Login:**

Dentro del módulo login\_window, por otra parte, nos encontramos con otros dos slots respectivamente:

Texto

Descripción generada automáticamente

Tal y como pasaba anteriormente, este slots es similar, pero para redirigirnos hacia la otra ventana (Registro), por lo cual a ser asociados a dos elementos UI diferentes y configurarlos de tal forma que la ventana actual se esconda, acabamos obteniendo una conexión fluida entre ambas vista.

Texto

Descripción generada automáticamente

Con este otro slot, el cual interactúa con otro método del módulo controlador, verificamos si los campos introducidos dentro de la ventana son válidos, es decir son de un usuario que ha sido previamente registrado. En caso afirmativo, el sistema enviará un mensaje de bienvenida junto a su ‘username’, y en caso contrario, se informará de que las credenciales no han sido válidas.

**Conexión de los slots:**

Como en el caso anterior, los hemos tenido que asociar dentro del constructor de la clase a los elementos ui correspondientes de las vistas, tal y como se puede apreciar en la siguiente imagen:

Imagen que contiene Texto

Descripción generada automáticamente

# **3.- Integración de la Base de Datos dentro de la Aplicación:**

La creación de todos los módulos para esta tarea se ha realizado dentro de la carpeta ‘models’, en la cual tenemos uno destinado a crear tanto la conexión de la base de datos como para cerrarla, así como otro enfocado en inicializar la misma, creando la tabla usuarios e insertando unos datos como ejemplo.

**Módulo db:**

Texto

Descripción generada automáticamente

**Módulo inicializacion\_db:**

Texto

Descripción generada automáticamente

Además de todo ello, se ha creado un módulo con el objeto u instancia que manipularemos mediante las vistas, en este caso Usuario, así como un módulo CRUD con diferentes métodos para poder operar con la Base de Datos:

**Módulo usuario:**

Captura de pantalla de computadora

Descripción generada automáticamente

**Texto

Descripción generada automáticamenteMuestra del Módulo CRUD:**

**Captura de pantalla de computadora

Descripción generada automáticamente**

De esta forma, se podrá crear e inicializar la Base de Datos, y mediante el controlador podremos manipular todos los métodos creados en el CRUD para poder ser conectados posteriormente con las vistas.

# **4.- Resumen sobre la Aplicación:**

Texto

Descripción generada automáticamenteLa aplicación funciona de la siguiente forma 🡪 Todo comienza en la clase main.py, donde crearemos una instancias de la ventana de Login y otra para poder inicializar la base de datos. Al arrancar el flujo, saltaremos a dicha ventana:

Una vez estemos en el Login, como se comento anteriormente, nos podríamos redirigir a la ventana de Registro, o bien iniciar sesión. Tanto en un módulo como en otro, tenemos métodos que nos acaban llevando al módulo usuario\_controller:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamenteEste será el módulo donde se gestione tanto el verificar un usuario (Login) como el registrarlo (Registro), el cual está utilizando métodos del módulo CRUD, específicamente el seleccionar usuario mediante el nombre y el insertar un usuario, respectivamente:

Aquí, apreciamos que se usan métodos de db, por lo que podemos afirmar que es el otro punto de origen en la aplicación. Por tanto, en conclusión, su funcionamiento podríamos decir que es el siguiente:

En los modelos, creamos 4 módulos 🡪 uno con el objeto/instancia a representar, uno para crear/iniciar la base de datos, otro para crear la conexión con los parámetros necesarios, y por último un módulo CRUD con el que se podrán realizar operaciones.

De forma posterior y mediante el Controlador, vamos a conectar métodos del CRUD con elementos de las vistas, al ser usados de forma final en los slots.

Es por eso que, desde nuestro main, instanciamos la primera ventana que queremos mostrar, creamos o inicializamos nuestra base de datos, y accedemos a la ventana que funciona según lo que acabamos de explicar.

Imagen que contiene biombo, reloj, microondas

Descripción generada automáticamente

# **5.- Despliegue de la Aplicación:**

## **5.1.- Entorno Virtual:**

De forma previa a comenzar la realización de este proyecto, es necesario instalar el entorno virtual para poder trabajar, ya que entre otros aspectos fundamentales, trae integrado todas las dependencias que necesitaremos para poder desarrollar nuestro código, permitiendo instalar en cada proyecto lo necesario sin tener que interferir dentro de otros.

Para ello, lo primero que tenemos que hacer es crearnos un directorio, y desde VSC navegar hacia él. Una vez estemos dentro de la raíz del mismo, ejecutaremos el siguiente comando:

**python -m venv ‘nombre del entorno’:**

**Captura de pantalla de computadora

Descripción generada automáticamente**

Una vez se haya creado el entorno virtual, lo siguiente será instalar las dependencias respectivamente, las cuáles se incluyen dentro de un archivo denominado “requirements.txt”, el cual deberá ser modificado en función de las especificaciones de cada proyecto, respectivamente. Es por ello que nos traemos dicho archivo a la raíz de nuestro proyecto, y posteriormente, ejecutamos el comando:

**Interfaz de usuario gráfica

Descripción generada automáticamentepip install -r requirements.txt**

Una vez tengamos todas las dependencias que necesitamos instaladas, nuestro siguiente paso será el de activar el entorno previamente. Esto lo conseguimos mediante un comando, así como para desactivarlo y/o eliminarlo tendremos otros respectivamente:

**.\.env\Scripts\activate 🡪** Comando con el que activaremos el entorno virtual.

**deactivate** 🡪 Nos desactiva el entorno virtual.

Captura de pantalla de un celular

Descripción generada automáticamente**Remove-Item -Recurse -Force .\nombre\_entorno 🡪** Nos elimina el entorno virtual, para lo cual primero debe de ser desactivado.

## **5.2.- Convertir Archivo .ui en .py:**

Este paso ya fue explicado dentro del proyecto, sin embargo, vamos a volver a recordarlo brevemente: los archivos ui serán obtenidos desde Qt Designer de la siguiente forma:

Una captura de pantalla de una computadora

Descripción generada automáticamente

Una vez guardados en el directorio donde queramos, el siguiente paso es ejecutar el siguiente comando para obtener un archivo .py a partir del .ui en el mismo directorio:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente**pyside6-uic ‘archivo.ui’ -o ‘archivo.py’ 🡪 Donde archivo.ui será el archivo que deseamos convertir y archivo.py el nuevo archivo generado, con el nombre que nosotros le especifiquemos. De esta forma ejecutamos los comandos de la siguiente forma:**

# **5.3.- Docker y BBDD PostgreSQL:**

Para esta parte, se ha empleado, una herramienta de Docker denominada Docker Compose, la cual nos permite definir, ejecutar y gestionar respectivamente un conjunto de contenedores o un solo contenedor, con el objetivo de poder iniciarlos, detenerlos y/o configurarlos para que puedan gestionar el servicios postgres.

Es por ello que hemos tenido que realizar y configurar el archivo **docker-compose.yml** en la raíz del proyecto, mediante el cual con el comando **docker-compose up** podremos levantar y ejecutar nuestro contenedor de Docker, y mediante **docker-compose down** lo detendremos y eliminaremos posteriormente:

Texto

Descripción generada automáticamente

Con el archivo ya configurado y listo, es hora de ejecutar el comando, crear e iniciar el contenedor, y por lo tanto, iniciar el servicio para poder tener conexión con la base de datos:

Texto

Descripción generada automáticamente

**Resultado obtenido:**

**Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente**

Con todo arrancado ya, es hora de probar dentro de nuestra aplicación, si nos deja insertar a un usuario y loguearlo posteriormente, así como consultar la lista de usuarios posteriormente en un software de gestión de Base de Datos, en este caso, **DBeaver:**

**Creación de Usuario:**

**Captura de pantalla de computadora

Descripción generada automáticamente**

**Login con el Usuario:**

**Captura de pantalla de computadora

Descripción generada automáticamente**

**Comprobación en DBeaver:**

Para ello, tras acceder, tenemos que realizar lo siguiente:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Si todo ha resultado de forma exitosa, deberías ver un mensaje de confirmación al darle a probar la conexión, tal y como éste:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Tras haber creado una conexión de forma exitosa, el siguiente paso sería acceder a la siguiente ruta para poder realizar cualquier consulta SQL:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Para terminar el proyecto y comprobar que todo ha sido un éxito, hemos realizado una consulta sencilla para listar a todos los usuarios: SELECT \* FROM usuarios:

Interfaz de usuario gráfica, Aplicación, Tabla, Word

Descripción generada automáticamente

# 6.- Bibliografía:

Atoche Ortega, J. (2024). *Tema 02: Generación de interfaces gráficos con editores visuales y XML.* Dpto de Informática, IES Carrillo Salcedo.

*Qt Documentation*. (s. f.). Doc.qt.io. Recuperado 4 de noviembre de 2024, de <https://doc.qt.io/qt.html#qtforpython>