UNIVERSIDAD DEL QUINDIO

PROGRAMA DE INGENIERÍA ELECTRÓNICA

LABORATORIO DE PROGRAMACIÓN

Entrega Final

Gonzales Arias Samuel

López Peralta Aileen Susana

López Sánchez Juan Diego

Universidad del Quindío

Facultad de ingeniería Armenia

Quindío 2025

RESUMEN

La presente práctica corresponde a la segunda fase del desarrollo de un sistema de administración de parqueadero inteligente, el cual tiene como propósito principal integrar una interfaz gráfica que facilite la interacción directa entre los usuarios y el servidor de control de acceso. A partir de la arquitectura previamente desarrollada, que contemplaba registro de usuarios, generación y validación de códigos QR, y asignación de puestos, esta nueva etapa amplía la funcionalidad al permitir el manejo visual de dichas operaciones.

Se implementará una interfaz que gestione registros de usuarios, genere códigos QR cifrados, permita su lectura desde archivos o dispositivos de captura (como una cámara) y realice la asignación automática de espacios según el rol del usuario. Asimismo, se reforzará la comunicación entre cliente y servidor sobre red LAN, garantizando una experiencia más cercana a una aplicación real. Esta fase tiene como objetivo mejorar la accesibilidad del sistema, fortalecer su modularidad y facilitar su escalabilidad futura.

Palabras clave: *(Interfaz gráfica, código QR, servidor Python, gestión de parqueadero, red LAN.)*

I. *INTRODUCCIÓN*

Actualmente, muchas instituciones enfrentan la necesidad de optimizar el control de acceso vehicular a sus instalaciones, especialmente en espacios limitados como los parqueaderos. En este contexto, el uso de tecnologías como el reconocimiento visual, la validación de credenciales digitales y la asignación automatizada de recursos se vuelve una necesidad práctica y académica.

Esta práctica se desarrolla como continuación del sistema servidor de parqueadero creado anteriormente, el cual logró gestionar usuarios y validar el acceso mediante códigos QR generados y cifrados. En esa primera etapa, se estableció la estructura base del servidor mediante archivos users.py y parking_server.py, integrando además un algoritmo de detección de disponibilidad de puestos. Como se menciona en el informe previo, "se implementaron funciones para registrar usuarios, generar códigos QR y validar el acceso según disponibilidad de puestos" (Gonzales et al., 2025, p. 1).

Sin embargo, este sistema carece aún de una interfaz visual que permita a cualquier usuario (docente, visitante o administrador) operar el sistema sin necesidad de ejecutar scripts directamente. Por ello, en esta segunda fase se propone el diseño y la implementación de una interfaz gráfica intuitiva que permita realizar todo el proceso: desde el registro hasta la asignación del puesto, incluyendo generación y lectura del código QR, todo ello integrado con el servidor ya funcional.

El desarrollo de esta práctica implica conocimientos de programación gráfica en Python (como tkinter o PyQt), manejo de archivos y estructuras JSON, cifrado de datos, y comunicación cliente-servidor mediante peticiones HTTP. Este enfoque permite abordar habilidades técnicas

aplicables a sistemas reales, a la vez que se refuerzan conceptos de modularidad, seguridad y experiencia de usuario.

II. *REQUERIMIENTOS*

El sistema a desarrollar en esta segunda fase debe cumplir una serie de requerimientos tanto funcionales como técnicos, derivados del análisis de necesidades reales de acceso automatizado a parqueaderos y de la continuidad del sistema previamente implementado.

2.1 Requerimientos funcionales

1. *Registro gráfico de usuarios: La interfaz debe permitir que los usuarios puedan ingresar sus datos (ID, contraseña, programa y rol) desde un formulario. Se debe verificar si el usuario ya está registrado y mostrar mensajes apropiados.*

2. *Generación de códigos QR personalizados: Luego del registro, se debe crear un código QR que contenga información cifrada del usuario y su rol, con vigencia limitada. Esta imagen debe mostrarse y poder descargarse desde la GUI.*

3. *Lectura y validación del código QR: La interfaz debe permitir escanear un QR desde imagen o cámara, desencriptarlo, y verificar la existencia y validez del usuario. Como se indica en el informe anterior, "el código QR tiene una validez cercana de 1 día… si ya expiró esto no va a coincidir con lo anteriormente dicho" (Gonzales et al., 2025, p. 4).*

4. *Asignación automática de puestos: Al validar un QR correcto, el sistema debe consultar la disponibilidad de espacios y asignar uno en función del rol (docente, visitante, etc.).*

5. *Visualización de resultados: Toda acción realizada debe reflejarse en la interfaz mediante mensajes en pantalla o ventanas emergentes, mostrando si el acceso fue concedido, denegado o si el QR está vencido.*

2.2 Requerimientos técnicos

1. *Lenguaje y bibliotecas: El desarrollo se realizará en Python, utilizando bibliotecas como tkinter o PyQt para la interfaz; requests para las peticiones al servidor; pyzbar, pyqrcode, pillow y cv2 para manejo de imágenes y códigos QR.*

2. *Persistencia de datos: Se emplearán archivos JSON para almacenar los datos de usuarios y el estado de los puestos, como se ha hecho anteriormente: "archivos JSON para almacenamiento estructurado… y estado de puestos" (Gonzales et al., 2025, p. 3).*

3. *Conectividad LAN: La GUI debe estar en capacidad de comunicarse con el servidor en localhost o en una red local, usando los endpoints existentes en parking_server.py.*

4. *Escalabilidad y modularidad: El diseño debe permitir futuras ampliaciones, como el uso de sensores físicos, control de barreras o integración con bases de datos reales.*

III. PROCEDIMIENTO

Se creó un programa en python utilizando PyQt5 donde se pudo realizar una interfaz gráfica, la cual se es enfocada a la interacción con el usuario, la cual debía tener 3 características importantes las cuales fueron poder registrar usuarios, enviar código QR y obtener código QR, donde estas funciones se cumplen mediante el uso de las peticiones en el servidor, el cual se le debe enviar ciertos datos y nos va a devolver una respuesta, esto con el fin de poder gestionar los ingresos y registros de todos los usuarios de nuestro parqueadero.

primer paso a seguir debemos importar todas la bibliotecas y archivos con los cuales vamos a trabajar que son los siguientes en nuestro caso:

```
1   from PyQt5.QtCore import *
2   from PyQt5.QtGui import *
3   from PyQt5.QtWidgets import *
4   from new_window import NewWindow
5   from parking_client import getQR, registerUser, sendQR
6   import cv2
7   import os
```

Se importó cada biblioteca con sus respectivas funciones las cuales se extraen de ese archivo, ya que si utilizamos asterisco este nos ayuda a reunir todas las funcionalidades necesarias para nuestro código, donde cada biblioteca y archivo contiene distintas funcionalidades la cual son:

PyQt5: Es una biblioteca enfocada al diseño de interfaces gráficas en python las cuales contienen ventanas, botones, cuadros de texto y menús donde la principal estrategia es la interacción de usuario con la aplicación, esto se reparte en submódulos donde algunos son:

- *PyQt5.QtCore: está diseñado para poder analizar los eventos y maneja la organización de la interfaz.*
- *PyQt5.QtGui: es el encargado del manejo de las imágenes, fuentes y estilos.*
- *PyQt5.Qt Widgets: Es el encargado de crear los botones y cuadros de texto en la interfaz.*

new window: Es un archivo el cual contiene un código que nos ayuda a crear ventanas fácilmente puesto que podemos llamar esta función para poder crear una nueva ventana emergente.

parking client: Es un archivo el cual se encarga de poder realizar las funciones que como registrar un nuevo usuario, asignarle un puesto, recibir un QR y poder dar un QR personalizado; este también cumple la funcionalidad de poder gestionar de manera adecuada los ingresos .

Cv2: es una biblioteca las cual nos ayuda al manejo de imágenes donde podemos llegar a capturar, procesar y dar una respuesta mediante esta biblioteca.

Os: Ayuda al manejo de las funcionalidades del equipó local donde podemos llegar a manipular las rutas de archivos.

```
9   url = "http://localhost:80"
```

Se inicializó el puerto local del computador para poder tener un enlace de llegada de la información.

```
class MainWindow(QMainWindow):
```

Se creó la clase mainwindow la cual va a ser el menú principal que va a contener toda funcionalidad de este, donde se va a heredar las funciones de Qmainwindow el cual es el manejo de la interfaz gráfica.

```
12      def __init__(self):
13          super().__init__()
14          self.setWindowTitle("Interfaz de usuario")
15
16          # Labels
17          L1 = QLabel('Id: ')
18          L2 = QLabel('Contraseña: ')
19          L3 = QLabel('Rol: ')
20          L4 = QLabel("Programa: ")
21
22          # Campos de entrada
23          self.e1 = QLineEdit()
24          self.e2 = QLineEdit()
25          self.e3 = QLineEdit()
26          self.e4 = QLineEdit()
27
28          # Botones
29          b1 = QPushButton('Obtener QR')
30          b2 = QPushButton('Registrar Usuario')
31          b3 = QPushButton('Ver Parqueadero (Admin)')
32          b4 = QPushButton('Escanear QR')
33
34          b1.clicked.connect(self.GetQr)
35          b2.clicked.connect(self.RegisterUser)
36          b3.clicked.connect(self.ParqueaderoAdmin)
37          b4.clicked.connect(self.SendQr)
38
39          gridLayout = QGridLayout()
40
41          gridLayout.addWidget(L1, 0, 0)
42          gridLayout.addWidget(L2, 1, 0)
```

```
        gridLayout.addWidget(L3, 2, 0)
        gridLayout.addWidget(L4, 3, 0)

        gridLayout.addWidget(self.e1, 0, 1)
        gridLayout.addWidget(self.e2, 1, 1)
        gridLayout.addWidget(self.e3, 2, 1)
        gridLayout.addWidget(self.e4, 3, 1)

        gridLayout.addWidget(b1, 0, 3, 1, 3)
        gridLayout.addWidget(b2, 1, 3, 1, 3)
        gridLayout.addWidget(b3, 2, 3, 1, 3)
        gridLayout.addWidget(b4, 3, 3, 1, 3)

        widget = QWidget()
        widget.setLayout(gridLayout)
        self.setCentralWidget(widget)
        self.setWindowFlags(Qt.MSWindowsFixedSizeDialogHint)

```

ACÁ SE LLEGA AL CONSTRUCTOR DE LA CLASE DONDE SE VA A INICIALIZAR LA POSICIÓN TAMAÑO Y FUNCIONALIDAD DE LOS BOTONES Y ENTRADAS DE INFORMACIÓN PARA LA INTERFAZ GRÁFICA EL CUAL TAMBIÉN VA A CONTENER ALGUNAS ETIQUETAS PARA GUIAR AL USUARIO SOBRE QUÉ INFORMACIÓN DEBE INGRESAR A ESTA INTERFAZ Y ASÍ ENVIAR CORRECTAMENTE LA PETICIÓN AL SERVIDOR

```
    def GetQr(self):
        id = self.e1.text()
        password = self.e2.text()

        if len(id) and len(password):
            imgBytes = getQR(url, id, password)

            if type(imgBytes) is bytes and len(imgBytes):
                self.nw = NewWindow(imgBytes)
                self.nw.show()

                self.qr_filename = os.path.join(os.getcwd(), "temp_qr.png")
                with open(self.qr_filename, "wb") as f:
                    f.write(imgBytes)

                self.Q1 = QPushButton("Guardar en el escritorio")
                self.Q1.clicked.connect(self.guardaimagen)

                gridLayout2 = QGridLayout()
                gridLayout2.addWidget(self.Q1, 1, 3, 1, 3)
                widget2 = QWidget()
                widget2.setLayout(gridLayout2)
                self.setCentralWidget(widget2)
            else:
                msgBox = QMessageBox()
                msgBox.setIcon(QMessageBox.Warning)
                msgBox.setText("Usuario no Existe o Contraseña Incorrecta")
                msgBox.setWindowTitle("Alerta")
                msgBox.setStandardButtons(QMessageBox.Ok)
                msgBox.exec()
```

En esta parte del código se realiza la funcionalidad del botón para obtener el código QR el cual se inicia llamando las variables que se van a necesitar para poder realizar la función de esta luego se verifica que el usuario haya ingresado datos en el sistema si esto llega a ser correcto se debe enviar esta información a getQR para poder obtener una respuesta al obtener esta respuesta se verifica cuál fue la que arroja y dependiendo de esta se puede llegar a avisarle al usuario cuál fue la respuesta del servidor

```
92      def RegisterUser(self):
93          id = self.e1.text()
94          password = self.e2.text()
95          rol = self.e3.text()
96          programa = self.e4.text()
97
98          if len(id) and len(password) and len(rol) and len(programa):
99              Usuario = registerUser(url, id, password, programa, rol)
100
101             msgBox = QMessageBox()
102             if Usuario == "User succesfully registered":
103                 msgBox.setIcon(QMessageBox.Information)
104                 msgBox.setText("Usuario Registrado")
105                 msgBox.setWindowTitle("Mensaje informativo")
106             else:
107                 msgBox.setIcon(QMessageBox.Warning)
108                 msgBox.setText("Usuario ya existe")
109                 msgBox.setWindowTitle("Alerta")
110             msgBox.setStandardButtons(QMessageBox.Ok)
111             msgBox.exec()
```

En esta parte del código se realiza la funcionalidad de registrar el usuario donde primero se deben inicializar las variables con las cuales se va a trabajar luego de esto se debe verificar si los espacios que se necesitan están rellenados se procede a llamar a la función de registerUser y poder ingresarle los datos que se tienen para poder registrar este usuario en la base de datos por último se verifica qué respuesta dio el servidor y con esto se procede a llamar una ventana emergente que informa al usuario

```
156     def SendQr(self):
157         from qrscan import MainApp
158         self.scanner_window = MainApp()
159         self.scanner_window.show()
```

Se llama al archivo qr scan donde se importa la función de MainApp la cual tiene como objetivo escanear el código QR y enviarlo al servidor para obtener esta función luego se procede a mostrar la respuesta al usuario en esta parte se importó la biblioteca debido a que por su funcionalidad hace que el programa inicie la cámara sin haber recibido esa instrucción directamente

```python
"""Identificación de código QR válido"""
"""Alexander López Parrado"""

# Módulos de Qt
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
# Módulos OpenCV
import cv2,imutils #pip install imutils
from parking_client import getQR, registerUser, sendQR


# Módulo de código QR
from pyzbar.pyzbar import decode

# Clase hilo para captura de la cámara
class MyThread(QThread):

    # Señal que se emite a la ventana para indicar un nuevo frame
    frame_signal = pyqtSignal(QImage,bool)


    # Método run del hilo
    def run(self):
        # Inicialmente está corriendo
        self.is_running=True

        # Abre cámara
        self.cap = cv2.VideoCapture(0,cv2.CAP_DSHOW)

        # Ciclo mientras esté corriendo
        while self.is_running:

            # Lee frame
            _,frame = self.cap.read()

```

```
            # Se convierte a imagende Qt
            qframe = self.cvimage_to_qimage(frame)

            # Decodifica el código QR
            decodedQR = decode(frame)

            # Se retorna una lista, si no está vacía hay un código QR válido
            if len(decodedQR):
                # Se detiene la captura
                self.cap.release()
                # Se emite la señal con el frame y la bandera en True
                self.frame_signal.emit(qframe,True)
                # Se detiene el hilo
                self.is_running=False
                return
            # Emite señal que visualiza imagen
            self.frame_signal.emit(qframe,False)

        # Si se detiene la captura se cierra la cámara
        self.cap.release()

    # Se convierte imagen de OpenCV a imagen de Qt
    def cvimage_to_qimage(self,image):
        image = imutils.resize(image,width = 640)
        image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
        image = QImage(image,
                       image.shape[1],
                       image.shape[0],
                       QImage.Format_RGB888)
        return image

    # Slot para detener la captura como una señal desde la ventana
    @pyqtSlot()
```

```python
    def stop_capture(self):
        self.is_running=False


# Ventana principal
class MainApp(QMainWindow):

    # Señal para detener la captura que se emite hacia el hilo
    stop_signal = pyqtSignal()

    def __init__(self):
        super().__init__()
        self.init_ui()
        self.show()

    def init_ui(self):
        self.setFixedSize(640,640)
        self.setWindowTitle("QR Scanner")
        self.is_streaming=False

        # Widget central
        widget = QWidget(self)

        # Con layout vertical
        layout = QVBoxLayout()
        widget.setLayout(layout)

        # Label para la visualización de los frames de la cámara
        self.label = QLabel()
        layout.addWidget(self.label)

        # Botón para abrir/cerrar la cámara
        self.open_btn = QPushButton("Open The Camera", clicked=self.open_close_camera)
        layout.addWidget(self.open_btn)

```

```
107         self.setCentralWidget(widget)
108
109     # Método para detener el streaming
110     def stop_streaming(self):
111         self.open_btn.setText("Open The Camera")
112         self.is_streaming=False
113         self.camera_thread.frame_signal.disconnect()
114
115     # Método para abrir/cerrar la cámara
116     def open_close_camera(self):
117         if self.is_streaming:
118             self.stop_streaming()
119             self.stop_signal.emit()
120             self.camera_thread.wait()
121         else:
122             # Se crea el hilo
123             self.camera_thread = MyThread()
124             # Conecta la señal de los frames para que sea bloqueante
125             self.camera_thread.frame_signal.connect(self.setImage,Qt.BlockingQueuedConnection)
126             # Conecta la señal de detener la captura
127             self.stop_signal.connect(self.camera_thread.stop_capture)
128             self.is_streaming=True
129             self.open_btn.setText("Close The Camera")
130             # Inicia el hilo de captura
131             self.camera_thread.start()
132
133
134     # Slot de la señal que proviene de hilo que envía el frame
135     @pyqtSlot(QImage,bool)
136     def setImage(self,image,flag):
137         # Si no ha detectado código Qr visualiza la imagen
138         if flag==False:
139             self.label.setPixmap(QPixmap.fromImage(image))
140         else:
```

```
142                 # Imprime que el código Qr es válido
143                 print('Valid Qr')
144 
145                 # Imprime que el código Qr es válido
146 
147                 # Guarda la imagen escaneada
148                 image.save("scanned_qr.png", "PNG")
149                 print("Código guardado como scanned_qr.png")
150 
151     # Envía la imagen al servidor
152                 url = "http://localhost:80"
153                 spot = sendQR(url, "scanned_qr.png")
154 
155                 if image:
156                     url="http://localhost:80"
157                     spot=sendQR(url,"scanned_qr.png")
158                     if spot:
159                         msgBox=QMessageBox()
160                         msgBox.setIcon(QMessageBox.Information)
161                         msgBox.setText(spot.decode("utf-8"))
162                         msgBox.setWindowTitle("Puesto Asignado")
163                         msgBox.setStandardButtons(QMessageBox.Ok)
164                         msgBox.exec()
165 
166                 # Dibuja un cuadro verde
167                 painter=QPainter(self.label.pixmap())
168                 painter.setBrush(QBrush(QColor("green")))
169                 painter.setPen(QPen(QColor("green")))
170                 x1,y1,x2,y2=image.rect().getCoords()
171                 painter.drawRect(x1,y1,x2,y2)
172                 painter.end()
173                 self.label.repaint()
174                 self.stop_streaming()
175 
176
```

SE INICIA IMPORTANDO TODAS LAS BIBLIOTECAS QUE SE VAN A UTILIZAR DONDE SE PROCEDE A CREAR EL HILO QUE VA A CAPTURAR EL QR MEDIANTE LA CÁMARA LA CUAL SE HACE POR MEDIO DE OPENCV LUEGO DE PROCESAR CADA FOTOGRAMA BUSCANDO EL CÓDIGO QR VA A DETENER LA CAPTURA Y LO VA A ENVIAR A LA INTERFAZ DONDE LUEGO SE VA A GUARDAR EN UNA VARIABLE DESPUÉS DE TENERLA EN LA VARIABLE SE DEBE ENVIAR AL SERVIDOR PARA OBTENER UNA RESPUESTA DESCRIBIENDO EL PASO A PASO DEL CÓDIGO ES PODER INICIALIZAR LAS BIBLIOTECAS LUEGO SE VA A CREAR EL HILO QUE VA A CAPTURAR EL CÓDIGO QR EL CUAL VA A ANALIZAR EL FRAME DE LA CÁMARA.

DONDE TAMBIÉN VA A CORRER UNO DE LOS HILOS DONDE VA A ESTAR VERIFICANDO Y BUSCANDO EL CÓDIGO QR EL CUAL VA A DECODIFICAR Y VA A VERIFICAR QUE SEA UN CÓDIGO QR SI ESTO ES CORRECTO VA A DETENER LA CÁMARA Y LA VA A CERRAR DONDE DESPUÉS DE TENER ESTA IMAGEN SE VA A PONER EN FORMATO DE IMAGEN DE QT LUEGO DE TENER ESTO SE DEBE CREAR LA VENTANA PRINCIPAL LA CUAL VA A CONTENER EL CONSTRUCTOR EL CUAL VA A HEREDAR LAS FUNCIONES QUE YA SE TENÍAN EN LA BIBLIOTECA AHORA SE VA A INICIALIZAR LA VENTANA CON EL QR ESCANEADO EL CUAL SE VA A ORGANIZAR EN LA MITAD DE LA VENTANA.

SE VA A AGREGAR UN BOTÓN PARA TENER LA OPCIÓN DE ABRIR Y CERRAR LA CÁMARA DONDE SE VA A TENER UN MÉTODO EL CUAL VA A DETENER LA CÁMARA CUANDO DETECTE EL QR DONDE SE VA A TENER OTRO HILO QUE VA A SER PARA PODER CONECTAR LAS DEMÁS SEÑALES Y ASÍ PODER TENER UNA MEJOR IMAGEN LUEGO SE VA A CREAR UN NUEVO HILO QUE VA A REVISAR SI YA SE VISUALIZÓ LA IMAGEN Y SI ESTO ES CIERTO VA A IMPRIMIR QUE EL CÓDIGO QR ES

VÁLIDO DESPUÉS DE ESTA PARTE SE VA A GUARDAR LA IMAGEN QUE RESULTÓ DE TODO ESTE PROCESO Y SE VA A ENVIAR AL SERVIDOR DONDE LUEGO SE VA A ASIGNAR EN UNA VENTANA EMERGENTE EL PUESTO ESCOGIDO PARA EL USUARIO LUEGO DE ESTO SE VA A DIBUJAR UN CUADRO VERDE QUE INDICA QUE YA SE PUDO ESCANEAR EL CÓDIGO QR BIEN

```
161    # Crea la aplicación principal
162    app = QApplication([])
163    ex = MainWindow()
164    ex.show()
165    app.exec()
166
```

SE CREA LA VENTANA DONDE SE GESTIONA LA INTERFAZ POR MEDIO DE QAPPLICATION, LUEGO CON MAINWINDOW SE CREA DIRECTAMENTE EL OBJETO EL CUAL VA A SER LA GUI DESPUÉS SE PROCEDE A MOSTRAR ESA INFORMACIÓN Y CUANDO YA SE HAYA ACABADO DE MOSTRARLA SE CIERRAN TODAS LAS VENTANAS.

IV. RESULTADOS

EL OBJETIVO PRINCIPAL DEL PROYECTO ES DISEÑAR E IMPLEMENTAR UNA INTERFAZ GRÁFICA QUE FACILITE LA INTERACCIÓN DEL SERVIDOR CON EL USUARIO DE MANERA CLARA Y EFICIENTE, CON EL FIN DE GENERAR EN EL REGISTRO DE USUARIOS UN BUEN MANEJO Y ORDEN AL ENTRAR Y SALIR DEL PARQUEADERO.

COMO RESULTADO SE DESARROLLÓ UNA INTERFAZ GRÁFICA FUNCIONAL QUE PERMITE LA INTERACCIÓN DEL USUARIO CON EL PARQUEADERO. VERIFICANDO SI EL USUARIO YA ESTÁ REGISTRADO EN LA BASE DE DATOS Y SI NO LO ESTÁ, LO REGISTRA AUTOMÁTICAMENTE Y LE GENERA UN CÓDIGO QR CON SU INFORMACIÓN CIFRADA; MEJORANDO LA EXPERIENCIA DEL USUARIO, FACILITANDO EL ACCESO AL SISTEMA.

PARA LOGRAR ESTE RESULTADO FUE FUNDAMENTAL REALIZAR CIERTAS ACTIVIDADES COMO:

- *ANALIZAR Y DISTRIBUIR CADA FUNCIÓN EN UN ARCHIVO DISTINTO SI ESTE TIENE UNA ALTA COMPLEJIDAD, YA QUE NOS AYUDA A LA DETECCIÓN DE ERRORES Y SU DEPURACIÓN.*

- *SE DEFINIERON LOS CAMPOS DE ENTRADA PARA LA VERIFICACIÓN DE USUARIOS (NOMBRE, ID, ROL, ETC).*

- *SE IMPLEMENTÓ UNA CONEXIÓN ENTRE LA INTERFAZ GRÁFICA Y LA BASE DEL SISTEMA.*

- *SE REALIZARON PRUEBAS PARA GARANTIZAR LA FUNCIONALIDAD CORRECTA DEL SISTEMA.*

V. CONCLUSIÓN

Al implementar una interfaz gráfica se mejora significativamente la experiencia del usuario, permitiendo una relación al sistema clara, segura y rápida. Este sistema automatiza los procesos del registro, evitando errores de manejo que podría provocar con mucha facilidad una persona. Este desarrollo permitió aplicar conocimientos de manera práctica que se adquirieron en la teoría de programación.

VI. REFERENCIAS

[1] Gonzales, S., López, A. S., & López, J. D. (2025). Servidor del Sistema de Administración de Parqueadero. Universidad del Quindío. Facultad de Ingeniería.
[2] Python Software Foundation. (2024). Python Documentation. https://docs.python.org/3/
[3] PyCryptodome Documentation. (2024). https://www.pycryptodome.org/
[4] PyQRCode Documentation. (2024). https://pypi.org/project/PyQRCode/
[5] IEEE. (2008). Standard for Software Test Documentation (IEEE Std 829-2008).