UNIVERSIDAD DEL QUINDIO

PROGRAMA DE INGENIERÍA ELECTRÓNICA

LABORATORIO DE PROGRAMACIÓN

Entrega: Servidor del Sistema de Administración de Parqueadero

Gonzales Arias Samuel

López Peralta Aileen Susana

López Sánchez Juan Diego

Universidad del Quindío

Facultad de ingeniería Armenia

Quindío 2025

RESUMEN.

Esta práctica consistió en el desarrollo de un servidor para un Sistema de Administración de Parqueadero, el cual busca gestionar usuarios y puestos, y validar el acceso mediante códigos QR. A partir del archivo base parking_server.py y del módulo users.py. Se implementaron funciones para registrar usuarios, generar códigos QR, y validar el acceso según disponibilidad de puestos. Se realizaron pruebas locales usando test parking client.py, verificando la comunicación entre el cliente y el servidor mediante protocolos HTTP y uso de librerías para generación y decodificación de códigos QR. La correcta operación del servidor fue validada por medio de respuestas esperadas ante los distintos escenarios de pruebas.

PALABRAS CLAVE(servidor, Python, gestión de usuarios, código QR, red LAN.)

1. INTRODUCCIÓN.

La práctica tuvo como objetivo implementar un servidor funcional para la gestión de usuarios y puestos de parqueadero, integrando el uso de códigos QR temporales para el control de acceso. El desarrollo se enfocó en el archivo users.py siendo este el archivo base del sistema, el cual interactúa con parking server.py. Se implementan funcionalidades para el registro de usuarios mediante identificación y contraseña, la generación de códigos QR encriptados con fecha y rol y la validación de estos para determinar el acceso a puestos disponibles.

El sistema está orientado al uso en redes locales, con soporte para pruebas en el mismo equipo o en varios dispositivos conectados a la misma LAN. Se usaron estructuras de datos en Python, archivos JSON para persistencia y bibliotecas para el manejo de códigos QR. También se integró el algoritmo de detección de puestos del laboratorio anterior para completar la funcionalidad del servidor.

2. REQUERIMIENTOS.

2.1 Uso del archivo base parking_server.py sin ninguna modificación.

2.2 Implementación del módulo users.py con funciones para:

- Registro de usuarios.
- Generación de códigos QR encriptados.
- Validación de códigos QR para acceso.
- Algoritmo de detección de puestos entregado en laboratorios pasados.

2.3 Se requirió el uso de librerías:

- requests
- pycryptodome
- pyqrcode
- pypng
- pyzbar
- pillow
- numpy
- cv2

2.4 Conectividad en red LAN para pruebas distribuidas.

2.5 Ejecución local o distribuida de los scripts parking_server.py y test_parking_client.py.

3. PROCEDIMIENTO.

Se realizó la tabla con el objetivo de resumir o desglosar el proceso llevado a cabo, en el cual se implementaron los datos necesarios, procesos involucrados y las variables utilizadas; facilitando la comprensión del funcionamiento general y permitiendo evaluar el cumplimiento de los criterios esperados en el desarrollo del servidor del parqueadero.

Tabla I. Análisis de procedimientos

ANÁLISIS DE PROCEDIMIENTOS	
¿Cuáles resultados se esperan?	1. *Que el sistema registre usuarios nuevos correctamente → **no** que duplique registros existentes.* 2. *Que genere códigos QR válidos, cifrados y con fecha → **no** códigos genéricos ni sin expiración.* 3. *Que valide accesos solo si hay disponibilidad de puestos y el QR es vigente → **no** acceso libre sin control.* 4. *Que el servidor responda con mensajes claros en JSON → **no** errores silenciosos ni respuestas mal formateadas.*
¿Cuáles son los datos disponibles o que se suministren?	1. *ID (entero), contraseña (cadena), programa (cadena) y rol (cadena) → **no** datos incompletos o mal tipados.* 2. *Archivos JSON para almacenamiento estructurado → **no** bases de datos improvisadas o no persistentes.* 3. *Códigos QR con datos cifrados y fecha de generación → **no** códigos sin información o sin vigencia.* 4. *Estado de puestos en archivos JSON actualizados → **no** valores estáticos o sin conexión al sistema real.*

¿Qué procesos se deben realizar?	1. *Registro del usuario (registeruser) validando duplicados.* 2. *Generación del código QR (get_qr) con cifrado y fecha.* 3. *Envío del QR al cliente (send_qr).* 4. *Lectura y verificación del QR usando pyzbar y pillow.* 5. *Validación de rol, tiempo y disponibilidad.* 6. *Actualización del estado de los puestos.* 7. *Respuesta del servidor en formato JSON.*
¿Qué variables se deben utilizar?	1. ***`id`:*** *número identificador del usuario.* 2. ***`password`:*** *clave del usuario.* 3. ***`program`:*** *programa académico.* 4. ***`role`:*** *rol del usuario.* 5. ***`timestamp`:*** *fecha de creación del QR.* 6. ***`qr_data`:*** *contenido cifrado del QR.* 7. ***`spots_status`:*** *estado de los puestos.* 8. ***`response`:*** *mensaje del servidor.* 9. ***`ip_address`:*** *IP del cliente para conexión en red.*

Configuración inicial:

Para el desarrollo de esta práctica, se inició con la instalación de las dependencias necesarias mediante el uso de pip, lo que permitió el correcto funcionamiento de los módulos involucrados en la generación y lectura de códigos QR, así como la comunicación entre cliente y servidor. Las bibliotecas incluidas fueron requests, pycryptodome, pyqrcode, pypng, pyzbar y pillow. Esta instalación se llevó a cabo dentro de un entorno de desarrollo controlado para asegurar compatibilidad entre versiones y evitar conflictos.

Análisis del código base:

Posteriormente se procedió al análisis del archivo parking server.py, el cual ya contenía la estructura de un servidor funcional, pero sin la lógica interna de gestión. Se identificaron los puntos de entrada de las peticiones y cómo estaban definidos los endpoints que comunican con users.py. Esto permitió comprender cómo fluye la información desde el cliente hacia el servidor y cómo este responde en cada caso. La interacción con los usuarios se desarrolló mediante el archivo users.py, el cual fue complementado con funciones específicas para registrar usuarios, generar códigos QR, y validar el acceso de los mismos a los puestos de parqueadero.

Desarrollo del módulo users.py:

Se implementó una función de registro de usuarios que almacena los datos ingresados de identificación y contraseña, almacenados de forma estructurada en archivos JSON. Luego, se programó la generación de códigos QR que contienen información cifrada, incluyendo la fecha de generación y el rol del usuario. Estos códigos QR tienen validez por un día, lo que garantiza un nivel básico de seguridad temporal y permite segmentar el acceso según permisos. La validación del código se realizó con base en su codificación y verificación temporal, permitiendo o denegando el acceso según el caso. Siguiendo con las demás indicaciones que se dieron, se realizó un codigo el cual era dividido por 3 grandes funciones las cuales eran registeruser, get qr y send qr, las cuales se nos dieron distintas indicaciones para poder llegar a el objetivo propuesto para poder hacer un servidor funcional que nos sirviera para un parqueadero real.

- Registeruser: Se pedían 3 grandes requisitos e instrucciones para completar esta función los cuales eran que los argumentos entrantes eran de distinto tipo como el id (entero), password (cadena), program (cadena) y role (cadena), también, si el usuario ya existía en nuestra base de datos se debía retornar "User already registered" y por último, si el usuario no existe se debía registrar el usuario en la base de datos y retornar "User successfully registered", esto se logró haciendo el siguiente código:

```
133 def registerUser(id, password, program, role):
134 
135     # Verifica si el archivo existe y lo crea si no existe
136     archivo = open(usersFileName, "r")
137     #lee el contenido del archivo y lo guarda en una lista
138     users = archivo.readlines()
139     #cierra el archivo
140     archivo.close()
141     usuarioencontrado=False
142     #recorre cada usuario en la lista de usuarios y los separa por espacios
143     for user in users:
144         #quita los espacios en blanco que hallan quedado y los separa por comas
145         partes = user.strip().split()
146         #si la lista de partes es 0 significa que no hay nada en la lista y se continua con el siguiente usuario
147         if len(partes) == 0:
148             continue
149         #se le pone el str a id para que no de error al comparar el id con el id del usuario
150         #ya que el dato que nos dio el usuario esta en tipo entero y el id de la base de datos esta en string
151         if partes[0] == str(id):
152                 #si el id es igual al id del usuario se le pone el valor de verdadero a la variable usuarioencontrado
153             usuarioencontrado=True
154     #verificaa que el usuario no se haya sido encontrado en la base de datos
155     if usuarioencontrado==False:
156         #si no se encontro se va expandir el archivo y se le va a agregar el nuevo usuario
157         archivo = open(usersFileName, "a")
158         #se le pone el str a todo para evitar errores al guardar los datos en el archivo
159         #estos se guardan en tipo string ya que el archivo es de texto
160         archivo.write(str(id) + " " + str(password) + " " + str(program) + " " + str(role) + "\n")
161         #se cierra el archivo para que no se quede abierto y no se generen errores
162         archivo.close()
163         return "User succesfully registered"
164     ##si el usuario ya existe se retorna el mensaje de que ya esta registrado
165     else:
166         return "User already registered"
167 
```

Fig 1.

- Los pasos que se siguieron para crear este código es el siguiente, primero se debe abrir el archivo y leer sus líneas de información, donde va a arrojar una lista de datos que mayormente va a ser la información de cada usuario donde luego se debe recorrer cada parte de esta lista y verificar que este usuario que vamos a registrar no esté en esta lista de datos, al recorrer cada usuario vamos a verificar nuevamente que este usuario no haya sido encontrado donde sí esto nos sale que este no esta registrado, se tiene que registrar, esto se hace por medio de agrandar el archivo y escribir en este nuevo espacio la información del nuevo usuario.

- Se demuestra que se cumplio con los requisitos e instrucciones anteriormente planteadas, donde en por la parte de los argumentos en la línea 160 se da solución a estos cambiando cada dato entrante a un string para poderla comparar efectivamente a la información de la base de datos, por parte de la verificación del usuario si está inscrito a nuestra base de datos se realiza por medio de dos formas, una de ella que está dentro del for donde este se ubica en la línea 151, luego se vuelve a verificar en la línea 154 donde si no se encontró que es el otro requisito e instrucción debe hacer el proceso que sigue.

- get qr: En esta función se pedida poder detectar si el usuario estaba en la base de datos, donde si esto era verdadero debemos generar un código qr que contenga todos los datos del usuario, donde se debía utilizar generate qr para poder visualizar este código qr como una imagen png.

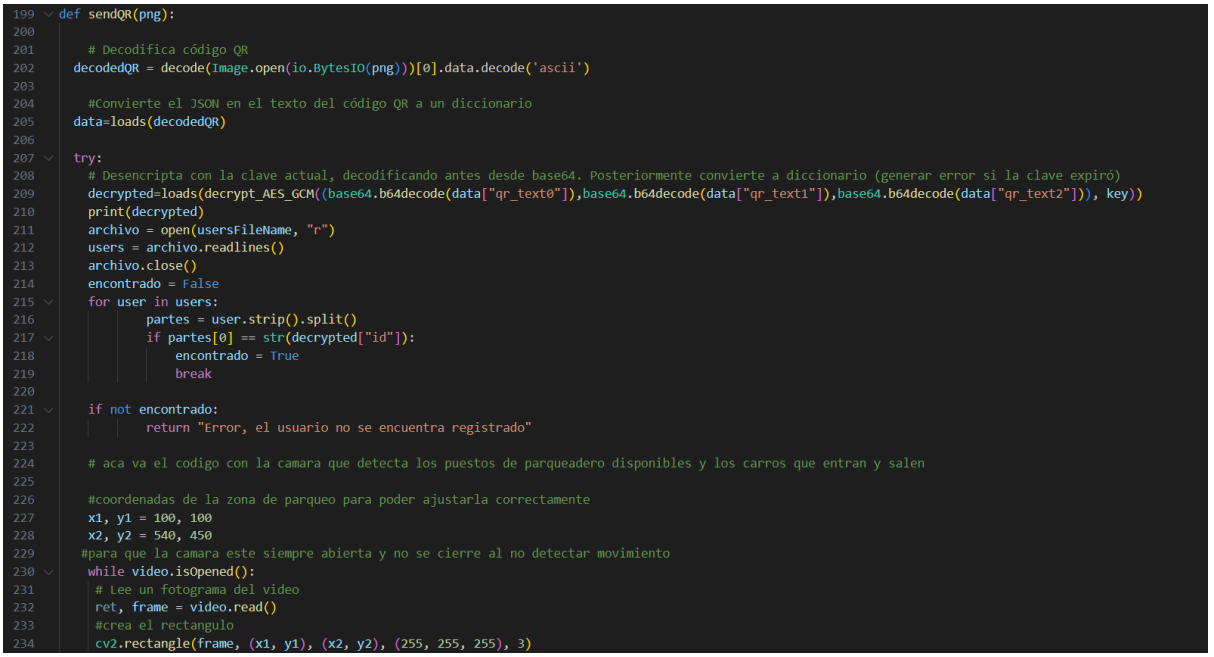
```
172 def getQR(id, password):
173     buffer = io.BytesIO()
174     archivo = open(usersFileName, "r")
175     users = archivo.readlines()
176     archivo.close()
177     usuarioencontrado2=False
178     for user in users:
179         #usa el .strip para quitar los espacios en blanco y el .split para separar las partes del usuario
180 
181         partes = user.strip().split()
182 
183         #se le pone el str a id para que no de error al comparar el id con el id del usuario
184         #las partes[0] es el id, partes[1] es la contraseña, partes[2] es el programa y partes[3] es el rol
185         if partes[0] == str(id) and partes[1] == password:
186             imagen=generateQR(id, partes[2], partes[3], buffer)
187             usuarioencontrado2=True
188             return buffer
189     if usuarioencontrado2==True:
190         pass
191     else:
192         return "Usuario no registrado"
```

Fig 2.

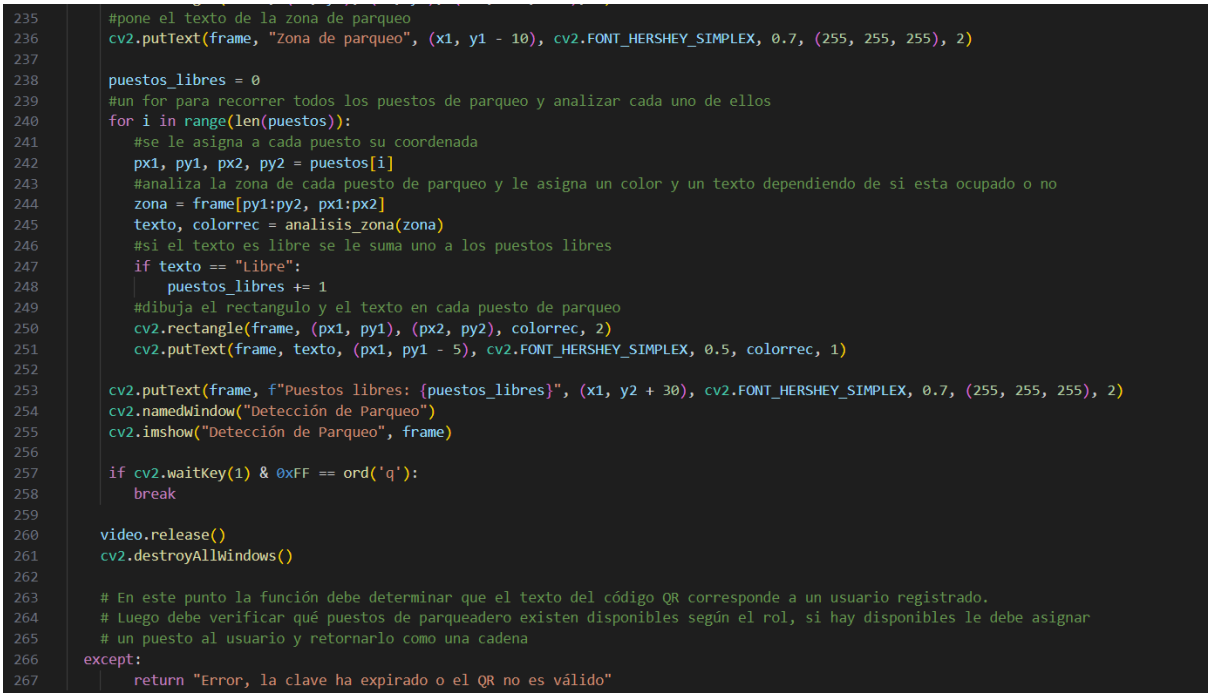
- El código primero debía leer el archivo que guarda los datos de los usuarios, luego de tener estos datos se deben almacenar en una variable cuando

cumplimos esto usamos un for para poder recorrer cada usuario y verificar si este usuario que está requiriendo el código qr está en nuestra base datos, donde este no dara ningun codigo qr si el usuario no está registrado ya que al no retornar el buffer que significa los bits que deben tener la imagen la función generate qr no podrá hacer esta imagen.

- send qr:En esta función debemos manejar un imagen png la cual es el código qr que contiene todos los datos del usuario donde nos dan como instrucción que el código debe verificar si el código qr puede ser desencriptados, si el usuario está registrado y asignarle un puesto de parqueo a la persona que envio el codigo qr.



```python
def sendQR(png):

    # Decodifica código QR
    decodedQR = decode(Image.open(io.BytesIO(png)))[0].data.decode('ascii')

    #Convierte el JSON en el texto del código QR a un diccionario
    data=loads(decodedQR)

    try:
        # Desencripta con la clave actual, decodificando antes desde base64. Posteriormente convierte a diccionario (generar error si la clave expiró)
        decrypted=loads(decrypt_AES_GCM((base64.b64decode(data["qr_text0"]),base64.b64decode(data["qr_text1"]),base64.b64decode(data["qr_text2"])), key))
        print(decrypted)
        archivo = open(usersFileName, "r")
        users = archivo.readlines()
        archivo.close()
        encontrado = False
        for user in users:
                partes = user.strip().split()
                if partes[0] == str(decrypted["id"]):
                    encontrado = True
                    break

        if not encontrado:
                return "Error, el usuario no se encuentra registrado"

        # aca va el codigo con la camara que detecta los puestos de parqueadero disponibles y los carros que entran y salen

        #coordenadas de la zona de parqueo para poder ajustarla correctamente
        x1, y1 = 100, 100
        x2, y2 = 540, 450
       #para que la camara este siempre abierta y no se cierre al no detectar movimiento
        while video.isOpened():
         # Lee un fotograma del video
         ret, frame = video.read()
         #crea el rectangulo
         cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 255, 255), 3)
```



```
235        #pone el texto de la zona de parqueo
236        cv2.putText(frame, "Zona de parqueo", (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
237
238        puestos_libres = 0
239        #un for para recorrer todos los puestos de parqueo y analizar cada uno de ellos
240        for i in range(len(puestos)):
241            #se le asigna a cada puesto su coordenada
242            px1, py1, px2, py2 = puestos[i]
243            #analiza la zona de cada puesto de parqueo y le asigna un color y un texto dependiendo de si esta ocupado o no
244            zona = frame[py1:py2, px1:px2]
245            texto, colorrec = analisis_zona(zona)
246            #si el texto es libre se le suma uno a los puestos libres
247            if texto == "Libre":
248                puestos_libres += 1
249            #dibuja el rectangulo y el texto en cada puesto de parqueo
250            cv2.rectangle(frame, (px1, py1), (px2, py2), colorrec, 2)
251            cv2.putText(frame, texto, (px1, py1 - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, colorrec, 1)
252
253        cv2.putText(frame, f"Puestos libres: {puestos_libres}", (x1, y2 + 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
254        cv2.namedWindow("Detección de Parqueo")
255        cv2.imshow("Detección de Parqueo", frame)
256
257        if cv2.waitKey(1) & 0xFF == ord('q'):
258            break
259
260    video.release()
261    cv2.destroyAllWindows()
262
263    # En este punto la función debe determinar que el texto del código QR corresponde a un usuario registrado.
264    # Luego debe verificar qué puestos de parqueadero existen disponibles según el rol, si hay disponibles le debe asignar
265    # un puesto al usuario y retornarlo como una cadena
266  except:
267        return "Error, la clave ha expirado o el QR no es válido"
```

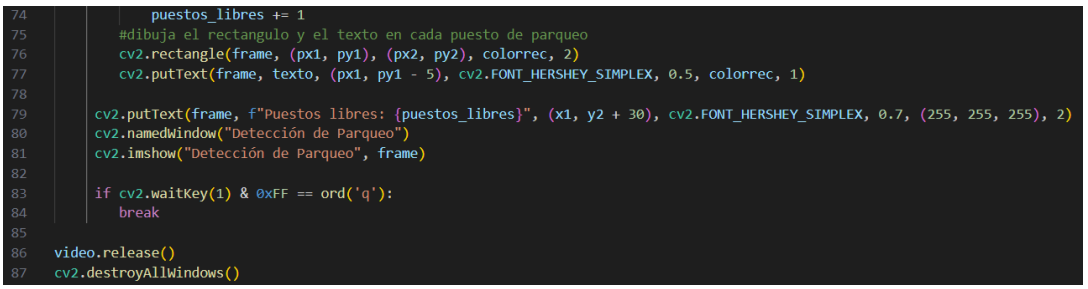
Fig 3.

principalmente este código empieza decodificando la imagen png y la convierte a un diccionario, luego de tener esto debemos utilizar un try y un except puesto a que este código qr tiene un validez cercana de 1 dia entonces al pasarla a desencriptarla con la clave y fecha esta dará un error puesto que si ya expiró esto no va a coincidir con lo anteriormente dicho, luego de verificar que este código qr si es válido debemos abrir nuestra base de datos y verificar que este usuario esté en esta, luego de confirmar que este usuario si existe en nuestra base de datos podemos llegar a identificar qué espacios están vacíos en nuestro parqueadero mediante el análisis de cada espacio el cual se va a realizar una descripción más a fondo en el siguiente apartado,

Integración del algoritmo de detección de puestos:

Además, se integró el algoritmo de detección de puestos del laboratorio anterior, permitiendo identificar si había disponibilidad de espacios según el rol del usuario (administrador, visitante, docente, etc.). Este algoritmo fue adaptado para trabajar con los datos extraídos de los archivos JSON, actualizando el estado de ocupación de los puestos conforme se autorizan los accesos.

```python
import numpy as np
import cv2


def color_predominante(region):
    promedio_b = np.mean(region[:, :, 0])
    promedio_g = np.mean(region[:, :, 1])
    promedio_r = np.mean(region[:, :, 2])

    if promedio_b > promedio_g and promedio_b > promedio_r:
        return "Azul", (255, 0, 0)
    elif promedio_g > promedio_b and promedio_g > promedio_r:
        return "Verde", (0, 255, 0)
    elif promedio_r > promedio_b and promedio_r > promedio_g:
        return "Rojo", (0, 0, 255)
    else:
        return "Indefinido", (255, 255, 255)

def analisis_zona(zona):
    gris = cv2.cvtColor(zona, cv2.COLOR_BGR2GRAY)
    bordes = cv2.Canny(gris, 100, 200)
    porcentaje_bordes = np.count_nonzero(bordes) / bordes.size

    if porcentaje_bordes < 0.05:
        return "Libre", (255, 255, 255)
    else:
        color, colorrec = color_predominante(zona)
        return f"Ocupado ({color})", colorrec

video = cv2.VideoCapture("https://192.168.1.118:8080/video")

puestos = [
    # Primera fila (5 puestos en la parte superior izquierda)
    (100, 100, 167, 196),  # Puesto 1
    (167, 100, 234, 196),  # Puesto 2
    (234, 100, 301, 196),  # Puesto 3
    (301, 100, 368, 196),  # Puesto 4
    (368, 100, 435, 196),  # Puesto 5

    # Segunda fila (4 puestos, 10 cm abajo de la primera fila)
    (100, 229, 167, 325),  # Puesto 6
    (167, 229, 234, 325),  # Puesto 7
    (234, 229, 301, 325),  # Puesto 8
    (301, 229, 368, 325),  # Puesto 9

    # Tercera fila (4 puestos, pegada a la segunda fila)
    (100, 325, 167, 421),  # Puesto 10
    (167, 325, 234, 421),  # Puesto 11
    (234, 325, 301, 421),  # Puesto 12
    (301, 325, 368, 421),  # Puesto 13
]

x1, y1 = 100, 100
x2, y2 = 540, 450
   #para que la camara este siempre abierta y no se cierre al no detectar movimiento
while video.isOpened():
    # Lee un fotograma del video
    ret, frame = video.read()
    #crea el rectangulo
    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 255, 255), 3)
    #pone el texto de la zona de parqueo
    cv2.putText(frame, "Zona de parqueo", (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)

    puestos_libres = 0
    #un for para recorrer todos los puestos de parqueo y analizar cada uno de ellos
    for i in range(len(puestos)):
       #se le asigna a cada puesto su coordenada
       px1, py1, px2, py2 = puestos[i]
       #analiza la zona de cada puesto de parqueo y le asigna un color y un texto dependiendo de si esta ocupado o no
       zona = frame[py1:py2, px1:px2]
       texto, colorrec = analisis_zona(zona)
       #si el texto es libre se le suma uno a los puestos libres
       if texto == "Libre":
```



```
                puestos_libres += 1
            #dibuja el rectangulo y el texto en cada puesto de parqueo
            cv2.rectangle(frame, (px1, py1), (px2, py2), colorrec, 2)
            cv2.putText(frame, texto, (px1, py1 - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, colorrec, 1)

        cv2.putText(frame, f"Puestos libres: {puestos_libres}", (x1, y2 + 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
        cv2.namedWindow("Detección de Parqueo")
        cv2.imshow("Detección de Parqueo", frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

video.release()
cv2.destroyAllWindows()
```

Fig 4.

para poder identificar si el espacio está ocupado en un sitio predeterminado se puede llegar a recrear los puestos del parqueadero en una cámara para que ella solo analice estos espacios y así no generar errores, por lo cual se trabaja a una escala de píxeles como lo es 1cm vale 10px entonces desde este punto podemos recrear el mapa de estacionamiento llevándolo a esta escala y ubicándolo en el plano, donde para esto se guardaron en la variable de puestos, luego se debe declarar la ip de la camara, despues de tener todo esto va a entrar a while que va a hacer que la cámara siempre esté encendida donde va a realizar una creación de los rectángulos apropiados, tanto para que se alinee de acuerdo a la maqueta con la cámara, cada puesto de parqueo se va a verificar si está ocupado o no está ocupado para poder saber qué pasos va a seguir el código.

Se tienen dos principales funciones las cuales son color predominante y análisis de zona. Cada una de estás enfocada en poder identificar a fondo cada puesto de parqueo, el color predominante tiene la tarea de sacar los promedios de cada dimensión de la imagen y poder promediarlas para poder distinguir qué color predomina en el carro que está estacionado. Por parte del análisis de zona, es la función que va a verificar si el espacio está libre o está ocupado, el cual lo hace mediante la identificación de los porcentajes de bordes que tiene el espacio.

Pruebas funcionales:

Para verificar el funcionamiento general del sistema, se realizaron pruebas controladas con el script test_parking_client.py, configurando su conexión con localhost y también con una IP dentro de una red LAN. Se probaron distintos escenarios: registro exitoso, generación y validación de QR válido, validación de QR vencido, y denegación de acceso por falta de puestos. En cada caso, se verificó que el servidor respondiera correctamente mediante mensajes JSON interpretables por el cliente. Esto también se puso verificar su correcto funcionamiento por medio las prueba unitarias las cual es la siguiente.

```python
import users  # Importa el módulo de usuarios

def test_registeruser():
    # Datos de prueba
    id = 1234556
    contraseña = "456723"
    programa = "matematicas"
    rol = "estudiante"
        # Llama a la función para registrar un usuario
    resultado = users.registerUser(id, contraseña, programa, rol)
    # Verifica que el resultado sea el esperado
    assert resultado == "Usuario registrado exitosamente", f"Se esperaba 'Usuario registrado exitosamente', pero se obtuvo '{resultado}'"

def test_getQR():
    # datos de prueba
    id = 1234556
    contraseña = "456723"
    programa = "matematicas"
    rol = "estudiante"
    #llama la funcion
    resultado = users.getQR(id, contraseña, programa, rol)
    # Verifica que el resultado sea del tipo bytes ya que nos si vemos el codigo original el buffer nos esta devolviendo los bits de la imagen
    assert type(resultado)is bytes, "Se esperaba que el resultado fuera del tipo bytes"


def test_sendQR():
    #los datos de prueba
    id = 1234556
    contraseña = "456723"
    programa = "matematicas"
    rol = "estudiante"

    # Llama a la función para enviar el código QR
    resultado = users.enviarQR(id, contraseña, programa, rol)
    # Verifica que el resultado sea el esperado
    assert resultado == "Código QR enviado exitosamente", f"Se esperaba 'Código QR enviado exitosamente', pero se obtuvo '{resultado}'"
```

Fig 5.

- Primero se debe importar las funciones de nuestro código de user.py, luego especificamos cada función que vamos a probar donde en este caso es registeruser, getqr y sendqr, luego de tener cada función a probar se declaran las variables que van a utilizar para la prueba unitaria los cuales van a ser id, contraseña, programa y rol, luego se llama la función con los datos ya anteriormente declarados y se verifica con assert que los resultados este igual a la forma en la que especificamos.

4. RESULTADOS

4.1 Objetivo:

El objetivo principal de esta práctica fue desarrollar e implementar un servidor funcional orientado a la gestión de un sistema de parqueadero inteligente. Para lograr esto, se propuso la creación de un módulo capaz de manejar el registro de usuarios, generar códigos QR personalizados y encriptados con vigencia temporal, y validar dichos códigos para controlar el acceso a los puestos de parqueo.

El sistema debía operar de manera segura, diferenciando a los usuarios por su rol (administrador, docente, visitante, entre otros), y permitiendo el acceso solo si existía disponibilidad de puestos correspondiente a ese rol. Este proceso debía ejecutarse de forma automática y controlada, garantizando trazabilidad en las acciones del usuario mediante archivos estructurados (JSON) y respuestas claras en formato JSON por parte del servidor.

Además, el sistema debía estar en capacidad de comunicarse con clientes en red local o mediante localhost, gestionando correctamente las peticiones desde el cliente hacia el servidor. La arquitectura del software debía facilitar pruebas, modularidad y escalabilidad, pensando en una posible implementación real o ampliación futura con dispositivos físicos y una interfaz gráfica de usuario.

4.2 Resultado:

El resultado obtenido fue la correcta implementación de todas las funcionalidades requeridas en el módulo users.py, incluyendo el registro seguro de usuarios, la generación dinámica de códigos QR válidos y cifrados, y la validación efectiva de estos para controlar el acceso. Las pruebas confirmaron que el servidor respondía correctamente ante múltiples escenarios posibles, como códigos vencidos, roles inválidos, usuarios no registrados o falta de disponibilidad en los puestos. También se logró establecer comunicación efectiva entre cliente y servidor tanto en modo local como en red LAN, demostrando así su operatividad en distintos entornos.

4.3 Acciones:

Entre las acciones que contribuyeron al logro de los objetivos se destacan la adecuada planificación modular del desarrollo, que permitió trabajar por partes y hacer pruebas individualizadas; la comprensión del flujo de datos entre cliente y servidor; y la integración estructurada del algoritmo de disponibilidad de puestos del laboratorio anterior. Como aspectos que dificultaron el cumplimiento inicial de algunos objetivos, se encontraron problemas con la validación de fechas en los códigos QR y errores al configurar la conexión entre diferentes dispositivos dentro de la red LAN. Estos problemas fueron solucionados progresivamente con ajustes en la lógica de validación y en las direcciones IP utilizadas en los scripts de prueba.

4.4 Resultado esperado futuro:

A futuro se espera expandir el sistema integrando una interfaz gráfica que permita la interacción directa del usuario con el servidor de forma visual y amigable. Asimismo, se plantea la posibilidad de conectar sensores y dispositivos físicos que automaticen el control de ingreso y salida en el parqueadero, permitiendo una solución más robusta y práctica para un entorno real.

4.5 Estrategia:

Para alcanzar estos resultados se propone trabajar en el diseño de la GUI utilizando bibliotecas como tkinter o PyQt, realizar pruebas de campo con múltiples dispositivos en distintas configuraciones de red, y documentar cuidadosamente todos los servicios ofrecidos por el servidor para facilitar la escalabilidad del proyecto.

5. CONCLUSIONES

- La utilización de un código base permitió enfocar el trabajo en el análisis, comprensión y extensión de las funcionalidades, promoviendo buenas prácticas de programación y trabajo modular.

- Se reforzaron habilidades para la detección de errores, usando en los códigos excepciones, depuración y pruebas unitarias, con el fin de detectar errores y verificar que todo funcionara correctamente.

- El laboratorio contribuyó en el refuerzo de habilidades para programación y análisis de problemas. Se aprendieron distintos conceptos técnicos, al igual que entender y modificar códigos que ya se encontraban planteados. El trabajo en equipo y la comunicación fueron clave para repartir las tareas de manera óptima y fluida.

5. REFERENCIAS

[1] Python Software Foundation. (2024). *Python Documentation*.

[2] PyCryptodome Documentation.

[3] PyQRCode Documentation.

[4] IEEE Std 829-2008. *Standard for Software Test Documentation*.