

Entrega No 1- Proyecto 1.

López Varón Yesli Dayanna

Código: 1090274518

Martínez Bedoya Richard

Código: 1096670916

Sánchez Soto Juan José

Código: 1090273203

Grupo 5



Alexander López Parrado

Julián Darío Barrero

Universidad del Quindío

Facultad de ingeniería

Ingeniería electrónica

Programación

Armenia, Quindío 2025

Resumen

El presente trabajo consistió en el análisis y prueba de un sistema de control del acceso a un parqueadero por medio de códigos QR, para empezar se tiene un servidor que ayuda la comunicacion atraves de solicitudes HTTP y por otra parte el archivo “user.py” que gestiona el registro, autenticación y generación de códigos QR para los usuarios. De acuerdo a esto se siguió una lógica para la validación de los puestos de parqueo, para así desplegar una lista con el estado del puesto (ocupado o libre) de acuerdo al rol de la persona.

Palabras claves

Cliente, servidor, control, decodificación, validación

Introducción

En diferentes lugares de parqueo se encuentra la problemática de que se ocupa el tope de espacios disponibles al momento de parquear lo que genera tráfico o congestión, por ello el manejo de estas situaciones es esencial, por lo que en este laboratorio se abordó la implementación de un sistema de parqueo inteligente .

Allí se encuentra un servidor que maneja las solicitudes, los usuarios pueden registrarse, obtener un código QR cifrado y recibir respuestas visuales desde el servidor. Por otro lado, el archivo “users.py” utiliza la lógica de seguridad y gestión de usuarios, incluyendo cifrado AES-GCM para proteger la información en los QR, validación de credenciales y asignación de plazas disponibles mediante el análisis de imágenes capturadas desde una cámara IP, de modo que se gestione rápidamente la asignación de parqueaderos o informe que todos los lugares se encuentran ocupados. Finalmente, se realizaron pruebas unitarias para garantizar el

correcto funcionamiento de las funciones del sistema y a continuación encontrará los análisis de los códigos principales y los procedimientos.

Procedimientos

- Se empezó analizando los códigos proporcionados, en el primero llamado “parking_server.py”, se hace uso de un servidor HTTP el cual permite la comunicación entre clientes y servidores de la web, respondiendo a solicitudes GET y POST, dónde GET se usa para pedir información mientras POST se usa para enviar información nueva.

Ahora bien ¿Cómo se usa esto en el código? Primero sabemos que este responde a solicitudes en la dirección IP y puerto especificados, GET crea y devuelve un QR mientras POST, registra y envía un QR al usuario. Además se hace uso de “users.py”, el cual sigue la lógica para por ejemplo registrar al usuario, enviar QR, etc. Finalmente se devuelve la imagen al cliente, la cual contiene el QR.

Por otra parte, en el archivo users.py se analizó lo siguiente:

Inicio del sistema:

Se importan las librerías necesarias para manejar cifrado, códigos QR, archivos, imágenes y video. Se define el archivo users.txt como la base de datos de usuarios, creándolo si aún no existe.

Cifrado y QR:

Para proteger la información del usuario, se usa cifrado AES en modo GCM. Cada día se genera una nueva clave de cifrado. Al generar un QR, los datos

del usuario (ID, programa y rol) se cifran y se convierten en texto codificado en base64, que luego se transforma en una imagen QR.

Registro de usuario:

Al registrar un nuevo usuario, el sistema revisa si el ID ya existe en el archivo. Si no está registrado, guarda el ID junto con la contraseña cifrada con SHA-256, el programa académico y el rol del usuario.

Obtención del QR:

Cuando un usuario solicita su QR, el sistema primero verifica que el ID y la contraseña sean correctos. Si coinciden con un registro, genera un QR personalizado con los datos cifrados y lo devuelve como imagen.

Lectura del QR y validación:

Cuando un QR es escaneado (por ejemplo, al ingresar al parqueadero), se lee la imagen, se decodifican los datos cifrados y se descifran. Luego, se verifica que el ID del QR pertenezca a un usuario válido en la base de datos.

Asignación de plazas por rol:

Según el rol del usuario, se definen las plazas que le corresponden:

- Estudiante: plazas 1 a 4
- Administrativo: plazas 5 a 7
- Profesor: plazas 8 a 10

Captura de plazas disponibles:

El sistema se conecta a una cámara IP y permite al operador capturar imágenes presionando la tecla [ESPACIO]. Se pueden capturar hasta 10

imágenes, una por cada posible plaza. Las imágenes se almacenan con un número identificador.

Procesamiento de imágenes:

Cada imagen capturada se convierte a escala de grises y se aplica detección de bordes. Si hay suficiente borde detectado (más de 12.500 píxeles), se considera que la plaza está ocupada.

Asignación de la plaza libre:

El sistema revisa las plazas capturadas. Si encuentra una plaza que pertenece al rol del usuario y está desocupada, la asigna. Si todas las plazas de su tipo están ocupadas, informa que no hay espacios disponibles.

Así mismo se completaron las tres funciones que pertenecen a este programa:

1. registerUser(id, password, program, role)

- Intenta abrir el archivo con los datos de los usuarios.
- with open(usersFileName, "r") as f:: Intenta abrir el archivo que contiene los usuarios registrados en modo lectura.
- for line in f:: Lee el archivo línea por línea.
- line = line.strip(): Elimina los espacios en blanco al inicio y final de cada línea.
- if not line: continue: Si la línea está vacía, pasa a la siguiente.

- `try: usuario = loads(line):` Intenta convertir la línea en un diccionario de Python. Si no puede, pasa a la siguiente línea.
- `if usuario["id"] == int(id)::` Verifica si el ID del usuario ya existe en los registros.
- `return "User already registered":` Si el usuario ya está registrado, devuelve un mensaje.
- `except FileNotFoundError: pass:` Si el archivo no se encuentra, no hace nada y continúa con el resto del código.
- Si el usuario no está registrado, lo agrega.
- `with open(usersFileName, "a") as f::` Abre el archivo en modo de escritura (añadir al final).
- `newUser = { ... }:` Crea un nuevo diccionario con los datos del usuario: ID, contraseña encriptada con SHA256, programa y rol.
- `f.write(dumps(newUser) + "\n"):` Convierte el diccionario a formato JSON y lo escribe en el archivo.
- Devuelve un mensaje indicando que el usuario se ha registrado con éxito.
- `return "User succesfully registered".`

En resumen esta función registra un nuevo usuario con ID, contraseña, programa y rol, guardando sus datos cifrados en un archivo, si el usuario ya está registrado lo informará.

2. getQR(id, password)

- Intenta leer el archivo de usuarios.
- `try: with open(usersFileName, "r") as f::` Abre el archivo de usuarios en modo lectura.
- `usuarios = f.readlines():` Lee todas las líneas del archivo.
- Verifica las credenciales de los usuarios.
- `for line in usuarios::` Itera sobre cada línea (usuario) en el archivo.
- `line = line.strip():` Elimina espacios innecesarios.
- `if not line: continue:` Si la línea está vacía, salta a la siguiente.
- `try: datos = loads(line):` Intenta convertir la línea en un diccionario. Si no puede, pasa a la siguiente.
- `if datos["id"] == int(id) and datos["password"] == sha256(password.encode()).hexdigest():` Verifica que el ID y la contraseña coincidan con los datos del usuario registrado.
- `buffer = io.BytesIO():` Crea un buffer en memoria para almacenar el QR.
- `generateQR(id, datos["program"], datos["role"], buffer):` Genera el código QR con los datos del usuario.
- `return buffer:` Devuelve el buffer con el QR generado.
- Si no se encuentra el usuario o las credenciales no coinciden, retorna None.

- return None.

Básicamente, la función verifica las credenciales de un usuario y, si son correctas, genera un código QR cifrado con su información.

3. sendQR(png)

- Importa las librerías necesarias.
- Se importan librerías para procesamiento de imágenes, decodificación de QR, encriptación AES, y captura de imágenes desde cámara.
- Decodifica el QR recibido.
- decoded = decode(Image.open(io.BytesIO(png)))[0].data.decode('ascii'): Abre la imagen PNG recibida, la decodifica como QR y obtiene su contenido como texto.
- data = loads(decoded): Convierte el texto decodificado en un diccionario.
- Desencripta los datos del QR.
- encrypted = (base64.b64decode(data["qr_text0"]), ...): Descodifica los fragmentos de datos encriptados del QR.
- decrypted = decrypt_AES_GCM(encrypted, key): Desencripta los datos usando una clave global.
- Verifica si el usuario está registrado.

- `with open(usersFileName, "r") as archivo::` Abre el archivo de usuarios.
- `for linea in archivo::` Lee cada línea.
- `if usuario["id"] == int(info["id"]):` Si el ID del usuario coincide con el proporcionado en el QR.
- `user_found = True:` Marca al usuario como encontrado si el ID coincide.
- `if not user_found::` Si el usuario no está registrado, retorna un mensaje de error.
- Determina los puestos disponibles según el rol del usuario.
- Si el rol es "Student", asigna un conjunto de puestos; lo mismo para "Administrative" y "Teacher".
- Si el rol no es válido, retorna un error.
- Captura imágenes de las plazas de parqueo.
- Se configura la cámara IP para capturar imágenes de las plazas de parqueo.
- Dentro del bucle, se muestra una vista previa de la cámara y se permite al usuario capturar las imágenes de las plazas con la tecla ESPACIO.
- Las imágenes capturadas se guardan en una carpeta.
- Identifica si las plazas están ocupadas o desocupadas.

- Se utiliza la función `identificarPlaza` para analizar las imágenes de las plazas y determinar si están ocupadas (basado en la cantidad de bordes detectados).

- La función devuelve un mensaje con el estado de cada plaza.
- Asigna la primera plaza desocupada disponible.
- Si se encuentra una plaza desocupada, se asigna.
- Si todas las plazas están ocupadas, retorna un mensaje indicándolo.

En pocas palabras, gracias a esta función se lee el QR desde una imagen, descripta y valida al usuario, luego le asigna una plaza libre según su rol, usando imágenes de una cámara para determinar si las plazas están ocupadas y así desplegar la validación de cada plaza.

- Pruebas

Las pruebas `"test_register_usuario_ya_registrado"` y `"test_register_nuevo_usuario"` están diseñadas para verificar el correcto funcionamiento de la función `"registerUser"`, la que es encargada de registrar nuevos usuarios.

En la primera prueba, se simuló un usuario ya registrado escribiendo directamente en el archivo `"usersFileName"` mediante el uso de `with` y usando `"open()"` as `f`, en modo `"w"` que esto sirve para poder abrir el archivo y modificarlo. Dentro de ese archivo, se guardó un diccionario con los datos del usuario. Luego se llamó a la función `"registerUser"` con los datos ya creados y

se utilizó `assert` para comprobar que el resultado sea "User already registered" en caso de que no funcione daría un error.

En la segunda prueba, se utilizó el `with` para borrar los usuarios que ya estuviesen registrados, lo cual simula que no hay usuarios registrados y evitar fallos al ejecutar la prueba. Se llamo nuevamente "registerUser" esperando como resultado "User succesfully registered".

Por otro lado, las pruebas "test_getQR_usuario_valido" y "test_getQR_usuario_invalido" las utilizamos para analizar la función "getQR", entonces en "test_getQR_usuario_valido", primero utilizamos "registerUser" para poder registrar al usuario y luego al usar "getQR" el cual debería recibir un objeto de tipo "io.BytesIO", que representaría la imagen del QR en la memoria.

En este sentido para comprobar su validez, se posiciona el buffer al inicio usando "buffer.seek(0)" y se usamos "Image.open()" para que se abriera como imagen, para después verificar con `assert` que su formato si sea PNG, con `img.format == "PNG"`. En "test_getQR_usuario_invalido", llamamos directamente a "getQR" con un ID inexistente y una contraseña incorrecta, y gracias a esto se comprueba que la función devuelva "None", esto demuestra que la función no crea el QR si el usuario no está registrado.

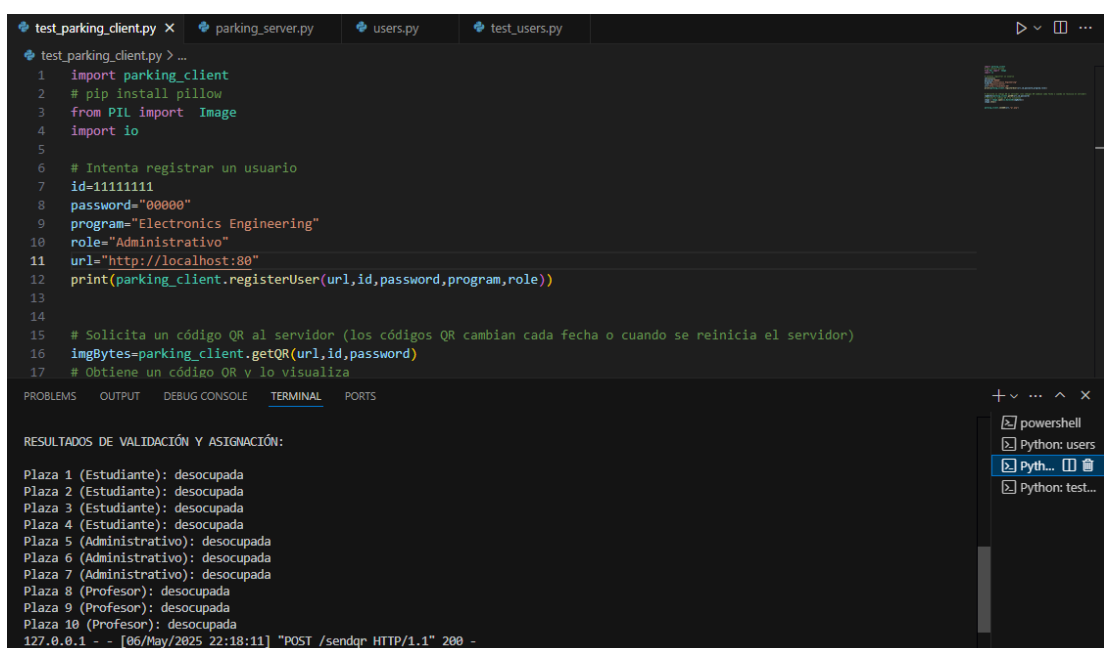
A su vez las pruebas "test_sendQR_usuario_valido" y "test_sendQR_usuario_no_registrado" fueron creadas para permitir analizar la funcionalidad de sendQR. Entonces en la prueba "test_sendQR_usuario_valido", primero se registro un nuevo usuario

utilizando "registerUser", para luego generar su respectivo código QR con "getQR", y se obtuvo su representación en bytes usando "buffer.getvalue()".

Para poder simular una respuesta del sistema, se usó "monkeypatch" ya que permite modificar la función encargada de identificar si una plaza está ocupada (identifySpot), forzando un resultado deseado como False. Luego se evaluó si el resultado de "sendQR" para mirar si contiene mensajes esperados como "puesto disponible asignado" o "plazas están ocupadas".

En la segunda prueba, se simuló un QR fraudulento. Para ello, se crea un diccionario con datos de un usuario inexistente y se cifra con "encrypt_AES_GCM", utilizando claves generadas con "users.key" o "urandom". El QR se generó con "pyqrcode.create" y se guarda en un "io.BytesIO". Finalmente, se le pasa este QR a "sendQR" y se espera como resultado un mensaje indicando que el usuario no está registrado.

Ahora bien, probar el servidor funciona correctamente, valida los puestos y muestra si está desocupado el rol al que corresponde.



The screenshot shows a VS Code editor with four tabs: test_parking_client.py, parking_server.py, users.py, and test_users.py. The active tab is test_parking_client.py, which contains the following Python code:

```
1 import parking_client
2 # pip install pillow
3 from PIL import Image
4 import io
5
6 # Intenta registrar un usuario
7 id="11111111"
8 password="00000"
9 program="Electronics Engineering"
10 role="Administrativo"
11 url="http://localhost:80"
12 print(parking_client.registerUser(url,id,password,program,role))
13
14
15 # Solicita un código QR al servidor (los códigos QR cambian cada fecha o cuando se reinicia el servidor)
16 imgBytes=parking_client.getQR(url,id,password)
17 # Obtiene un código QR y lo visualiza
```

Below the code, the TERMINAL panel shows the output of the script:

```
RESULTADOS DE VALIDACIÓN Y ASIGNACIÓN:
Plaza 1 (Estudiante): desocupada
Plaza 2 (Estudiante): desocupada
Plaza 3 (Estudiante): desocupada
Plaza 4 (Estudiante): desocupada
Plaza 5 (Administrativo): desocupada
Plaza 6 (Administrativo): desocupada
Plaza 7 (Administrativo): desocupada
Plaza 8 (Profesor): desocupada
Plaza 9 (Profesor): desocupada
Plaza 10 (Profesor): desocupada
127.0.0.1 - - [06/May/2025 22:18:11] "POST /sendqr HTTP/1.1" 200 -
```

Conclusiones

Se logró comprender la construcción del sistema y por ende el uso de solicitudes HTTP, lo cual permitió la comunicación entre cliente y servidor.

Así mismo, se verificó que la lógica de asignación de plazas según el rol, el cual funciona correctamente y además, el sistema puede detectar plazas ocupadas mediante procesamiento de imágenes. Por ende las pruebas reforzaron la fiabilidad del sistema al validar comportamientos esperados en diferentes escenarios, tanto válidos como inválidos.

Este enfoque puede extenderse a otras aplicaciones de control de acceso seguras y automatizadas, lo cual es muy importante a lo largo de la carrera permitiendo explorar nuevos conceptos que pueden ser útiles más adelante, empezando así con inconvenientes de la vida real que podemos solucionar por medio de herramientas de software.