

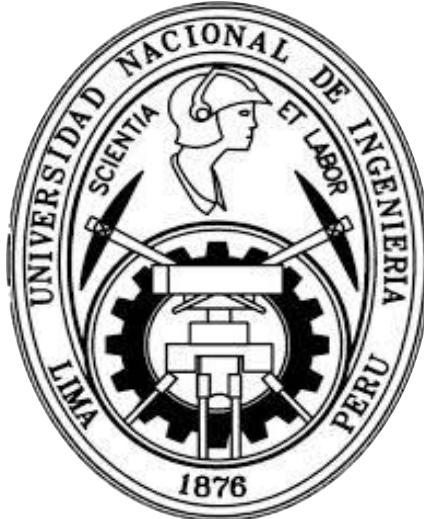


Universidad Nacional de Ingeniería

Facultad de Ingeniería Eléctrica y Electrónica

Proyecto Final para el curso de Programación

Orientada a Objetos



## BowserSecurity: Detección Inteligente de Armas

Autores:

- |                                |           |
|--------------------------------|-----------|
| - Flores Quiliche Fernando     | 20247516A |
| - Palomino Guzmán Bryan Raúl   | 20240549A |
| - Mejia Estrada Angel Fernando | 20242161K |

Supervisor: Yury Oscar Tello

Lima, 27 de Junio del 2025

# Introducción

En el contexto actual del Perú, la seguridad ciudadana se ha convertido en una de las principales preocupaciones de la población. El incremento de actos delictivos, como robos a viviendas, asaltos a mano armada e intrusiones en propiedades privadas, ha generado un sentimiento generalizado de vulnerabilidad, especialmente en zonas urbanas y periféricas donde la presencia policial resulta limitada o ineficiente. Esta situación ha impulsado a muchas familias y pequeños negocios a buscar soluciones tecnológicas para proteger sus espacios; sin embargo, gran parte de las alternativas en el mercado se centran en la simple grabación de video o en sistemas de vigilancia que requieren inversiones elevadas y el pago de servicios en la nube. En respuesta a esta realidad, nace BowserSecurity, un sistema de videovigilancia inteligente, económico y autónomo, desarrollado con tecnologías de Internet de las Cosas (IoT) y visión computacional, que busca prevenir antes que reaccionar.

A diferencia de las soluciones convencionales, BowserSecurity está diseñado no solo para captar imágenes, sino para interpretarlas en tiempo real y tomar decisiones automatizadas cuando se detectan comportamientos sospechosos o acciones potencialmente peligrosas. El corazón del sistema es una placa Raspberry Pi, que funciona como unidad de procesamiento local, gestionando una cámara de video conectada y ejecutando un modelo de inteligencia artificial optimizado para la detección de amenazas. Para lograrlo, se ha entrenado un modelo basado en el algoritmo YOLO (You Only Look Once), adaptado para reconocer no solo objetos, sino también acciones específicas como movimientos inusuales, ingreso no autorizado o desplazamientos anómalos dentro de una zona protegida. Esta capacidad de análisis contextual convierte a BowserSecurity en una herramienta preventiva eficaz, que puede lanzar alertas en tiempo real y permitir respuestas inmediatas frente a potenciales riesgos.

La parte lógica del sistema ha sido desarrollada en Python, aplicando la Programación Orientada a Objetos (POO) para estructurar el código de manera clara, modular y fácilmente escalable. Este enfoque permite integrar funcionalidades adicionales sin comprometer la estabilidad del sistema, tales como notificaciones automáticas, grabación de eventos críticos, autenticación de usuarios o incluso el control remoto a través de una interfaz gráfica o móvil. Al trabajar con tecnologías abiertas y accesibles, se garantiza que cualquier persona con conocimientos básicos pueda adaptar o extender el sistema según sus necesidades específicas.

# Objetivos:

El proyecto BowserSecurity tiene como propósito principal el diseño e implementación de un sistema inteligente de vigilancia capaz de detectar amenazas y comportamientos sospechosos en tiempo real, empleando tecnologías de Internet de las Cosas (IoT), visión por computadora e inteligencia artificial, con el fin de ofrecer una solución accesible, autónoma y adaptada a la problemática de seguridad actual.

## **Objetivo General:**

Desarrollar un sistema de videovigilancia inteligente, basado en una placa Raspberry Pi, visión computacional y un modelo de inteligencia artificial entrenado, que permita identificar acciones sospechosas y emitir alertas automáticas en tiempo real, utilizando un enfoque de programación modular con Python y principios de Programación Orientada a Objetos (POO).

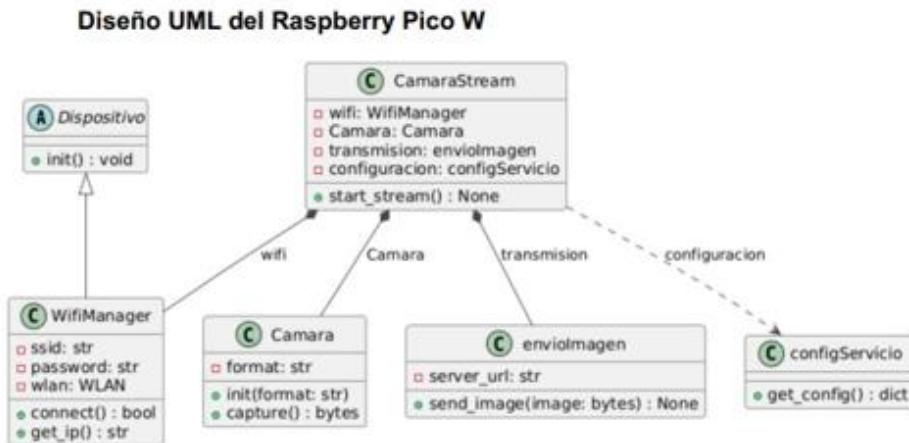
## **Objetivos Específicos:**

- Implementar una arquitectura de hardware utilizando una placa Raspberry Pi conectada a una cámara de videovigilancia, que funcione como unidad de monitoreo autónoma.
- Desarrollar un sistema de software en Python, utilizando Programación Orientada a Objetos (POO) para garantizar escalabilidad, mantenimiento y reutilización del código.
- Entrenar e integrar un modelo de detección basado en YOLO (You Only Look Once), optimizado para el reconocimiento de acciones específicas y amenazas dentro del entorno vigilado.
- Configurar el sistema para procesar video en tiempo real de forma local, permitiendo la identificación inmediata de comportamientos sospechosos y la emisión automática de alertas sin necesidad de intervención humana.
- Diseñar una lógica de control que permita registrar eventos detectados y, opcionalmente, activar módulos externos como alarmas o sistemas de notificación.
- Validar el funcionamiento del sistema en escenarios reales, simulando condiciones de vigilancia en espacios residenciales o comerciales, para evaluar su eficacia y eficiencia frente a sistemas convencionales.

# Diagrama UML

En este caso estamos utilizando 2 códigos, por lo tanto se presentarán 2 diagramas en UML, uno que es el que controla la aplicación y la otra que la de la placa Raspberry.

UML 1:



Descripción general:

La clase principal del sistema es CamaraStream, que coordina el funcionamiento de cuatro componentes fundamentales:

- **WifiManager**: gestiona la conexión inalámbrica a internet.
- **Camara**: se encarga de capturar imágenes en un formato determinado.
- **envoImagen**: transmite las imágenes capturadas a un servidor.
- **configServicio**: proporciona parámetros de configuración del sistema.

## 1. Clase abstracta Dispositivo

Esta clase sirve como clase base (superclase) para otros dispositivos del sistema. Define el método `init()` como inicializador genérico, aunque en esta implementación UML no contiene atributos propios. Sirve como punto común para representar a los dispositivos que requieren inicialización estándar.

## 2. Clase WifiManager

Heredando de Dispositivo, esta clase administra la conexión a la red WiFi.

Atributos:

**ssid: str:** nombre de la red WiFi.

**password: str:** contraseña de acceso.

**wlan: WLAN:** objeto WLAN para la conexión.

Métodos:

**connect() -> bool:** intenta conectar a la red WiFi, devolviendo True si tiene éxito.

**get\_ip() -> str:** retorna la dirección IP asignada al dispositivo.

## 3. Clase Camara

Se encarga de configurar y capturar imágenes desde el módulo de cámara.

Atributos:

**format: str:** define el formato de la imagen (por ejemplo, JPEG o PNG).

Métodos:

**init(format: str):** inicializa la cámara con el formato especificado.

**capture() -> bytes:** captura una imagen y la devuelve como un flujo de bytes, listo para su transmisión.

## 4. Clase envoImagen

Responsable de enviar la imagen capturada al servidor configurado.

Atributos:

**server\_url: str:** dirección del servidor receptor de imágenes.

Métodos:

**send\_image(image: bytes) -> None:** envía la imagen al servidor utilizando el protocolo o formato adecuado.

## 5. Clase configServicio

Esta clase proporciona parámetros de configuración del sistema (por ejemplo, dirección del servidor, parámetros de transmisión, formato de imagen).

Métodos:

**get\_config() -> dict:** devuelve un diccionario con la configuración actual del servicio.

## 6. Clase principal CamaraStream

Es la clase central que orquesta el funcionamiento de todo el sistema.

Atributos:

**wifi: WiFiManager:** objeto responsable de la conexión a internet.

**Camara: Camara:** objeto que gestiona la cámara.

**transmision: enviolImagen:** objeto que transmite la imagen.

**configuracion: configServicio:** configuración del sistema.

Métodos:

**start\_stream() -> None:** inicia el proceso completo de transmisión. Este método encapsula:

1. La conexión a la red WiFi.
2. La captura de imagen.
3. El envío de la imagen al servidor configurado.

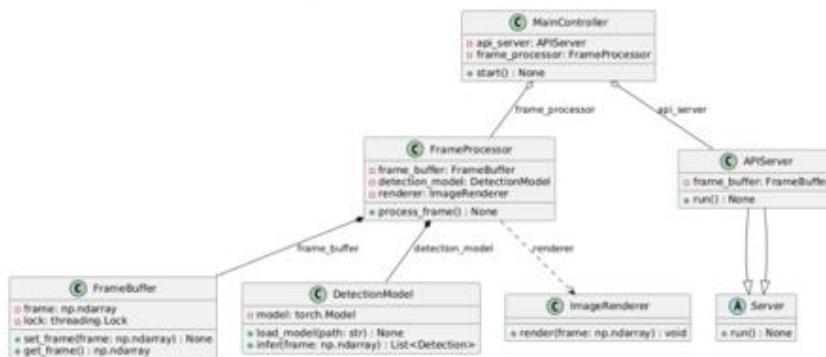
Relación entre clases

**CamaraStream** utiliza instancias de las clases **WiFiManager**, **Camara**, **enviolImagen** y **configServicio**.

**WiFiManager** hereda de la clase **Dispositivo**, lo que sugiere que en un futuro podría haber otras clases derivadas para distintos tipos de dispositivos conectados.

UML 2:

Diseño UML de la recepción del video



## Descripción general

La clase principal del sistema es MainController, que tiene la responsabilidad de inicializar y coordinar el trabajo entre los siguientes componentes:

- **APIServer:** recibe los datos de video entrantes y los guarda temporalmente en un buffer.
- **FrameProcessor:** toma los datos del buffer, los procesa usando un modelo de detección y envía los resultados visuales al módulo de renderizado.
- **FrameBuffer:** almacena temporalmente los fotogramas de video en tiempo real.
- **DetectionModel:** ejecuta el análisis de los fotogramas usando un modelo preentrenado.
- **ImageRenderer:** dibuja los resultados de la detección sobre la imagen para su visualización.

### Clase MainController

Es la clase orquestadora del sistema de recepción de video.

Atributos:

**api\_server: APIServer:** instancia del servidor que recibe los datos de video.

**frame\_processor: FrameProcessor:** instancia encargada del análisis del video.

Métodos:

**start() -> None:** inicia todo el flujo de operación, incluyendo la ejecución del servidor y el procesamiento del video.

### Clase APIServer

Responsable de recibir los fotogramas desde el exterior (por ejemplo, de la Raspberry Pi) y almacenarlos en el buffer.

Atributos:

**frame\_buffer: FrameBuffer:** referencia al buffer donde se almacenan los fotogramas entrantes.

Métodos:

**run() -> None:** ejecuta el servidor para comenzar a recibir los datos de video.

## Clase FrameProcessor

Es el módulo de procesamiento de video. Consumo los fotogramas del buffer, los analiza con un modelo de detección y los reenvía al renderizador.

Atributos:

**frame\_buffer: FrameBuffer:** fuente de entrada de los fotogramas.

**detection\_model: DetectionModel:** modelo de IA encargado de la detección.

**renderer: ImageRenderer:** módulo que renderiza los resultados sobre las imágenes.

Métodos:

**process\_frame() -> None:** procesa un fotograma específico, ejecuta el análisis y lo envía a renderizado.

## Clase FrameBuffer

Es una clase de almacenamiento temporal donde se colocan los fotogramas entrantes. Usa mecanismos de sincronización para garantizar la consistencia en entornos concurrentes.

Atributos:

**frame: np.ndarray:** imagen actual almacenada en formato de matriz NumPy.

**lock: threading.Lock:** bloqueo para sincronización segura de acceso.

Métodos:

**set\_frame(frame: np.ndarray) -> None:** actualiza el buffer con un nuevo fotograma.

**get\_frame() -> np.ndarray:** recupera el fotograma actual.

## Clase DetectionModel

Encapsula el modelo de inteligencia artificial utilizado para detectar objetos o acciones en los fotogramas.

Atributos:

**model: torch.Model:** modelo entrenado en PyTorch (por ejemplo, una versión de YOLO).

**model\_path: str:** ruta del archivo del modelo.

Métodos:

**infer(frame: np.ndarray) -> List[Deteciones]:** ejecuta la inferencia sobre una imagen y devuelve una lista de resultados (por ejemplo, personas detectadas, objetos, movimientos).

### Clase ImageRenderer

Encargada de dibujar los resultados de la detección (cajas, etiquetas, etc.) sobre el fotograma original, generando una imagen visualmente interpretativa.

Métodos:

**render\_frame(frame: np.ndarray) -> void:** toma el fotograma y dibuja los resultados de detección.

Relaciones entre clases:

- **MainController** crea y coordina instancias de **APIServer** y **FrameProcessor**.
- **APIServer** se comunica directamente con el **FrameBuffer** para almacenar los fotogramas.
- **FrameProcessor** accede al mismo **FrameBuffer** para leer los datos, pasa el contenido a **DetectionModel** para analizar, y luego a **ImageRenderer** para visualizar.
- Esta arquitectura crea un flujo secuencial eficiente: recepción → almacenamiento → procesamiento → renderizado.

### Relación entre los dos diagramas UML

El sistema **BowserSecurity** ha sido diseñado siguiendo una arquitectura distribuida y modular, en la cual los dos diagramas UML presentados representan **dos etapas complementarias del flujo de vigilancia inteligente**: la **captura y envío del video** (realizado en la Raspberry Pi Pico W) y la **recepción, procesamiento e interpretación del video** (realizado en un servidor local o máquina central). Ambos diagramas reflejan subsistemas distintos pero altamente interconectados, que colaboran para garantizar la detección eficiente de amenazas en tiempo real.

#### 1. Diagrama 1: Captura y transmisión desde Raspberry Pi Pico W

Este primer diagrama muestra cómo el dispositivo embebido (Raspberry Pi Pico W) gestiona la conexión WiFi, la cámara, la configuración del servicio y el envío de imágenes. Las clases WifiManager, Camara, envioImagen, configServicio y CamaraStream trabajan en conjunto para capturar fotogramas desde la cámara integrada, formatearlos y transmitirlos mediante HTTP o sockets al servidor de análisis.

En esta etapa, el **sistema actúa como cliente**, enviando los datos visuales hacia un servidor remoto sin realizar procesamiento local de imágenes. Su responsabilidad principal es capturar el entorno visual y transmitirlo de manera eficiente.

## 2. Diagrama 2: Recepción y procesamiento del video en el servidor

El segundo diagrama describe el lado servidor, que recibe los fotogramas desde la Raspberry Pi y los analiza en tiempo real utilizando un modelo de detección personalizado. Las clases MainController, APIServer, FrameProcessor, FrameBuffer, DetectionModel e ImageRenderer forman un pipeline secuencial de procesamiento:

- APIServer recibe los datos enviados por la clase envioImagen.
- FrameBuffer actúa como espacio compartido donde se almacenan temporalmente los fotogramas entrantes.
- FrameProcessor toma estos datos, los pasa al modelo (DetectionModel) y luego a ImageRenderer para visualizar los resultados.

Aquí, el sistema actúa como **centro de análisis inteligente**, capaz de detectar acciones, objetos o comportamientos sospechosos gracias a un modelo YOLO previamente entrenado.

Los dos diagramas UML representan **dos mitades de un mismo sistema distribuido**, donde la **captura embebida** y el **procesamiento centralizado** se integran a través de una interfaz bien definida: la transmisión de imágenes en tiempo real. Esta separación permite escalar el sistema, actualizar modelos de IA sin afectar el hardware, y mantener una arquitectura limpia, clara y orientada a objetos tanto en el entorno físico (IoT) como en el entorno lógico (servidor Python).

Gracias a esta integración, BowserSecurity puede funcionar como un sistema **modular, robusto y extensible**, en el cual se pueden sumar múltiples dispositivos Raspberry Pi como nodos de captura, todos conectados a un servidor inteligente central que analiza y responde ante amenazas de forma autónoma.

# Explicación del código de la aplicación en Python:

A continuación, se explicara el código empleado en Python para la aplicación, haciendo uso de conjuntos de lineas para una mejor vizualización. Las lineas en blanco han sido omitidas.

```
C: > Users > ASUS > OneDrive > Escritorio > Seguridad (renovado).py > ...
1 import tkinter as tk
2 from tkinter import ttk, messagebox, filedialog
3 from datetime import datetime
4 import time
5 import threading
6 import shutil
7 import cv2
8 from PIL import Image, ImageTk
9 import socket
10 import os
11 import logging
12 from ultralytics import YOLO
13 import winsound # Para reproducir sonido en Windows
14 from twilio.rest import Client # Para enviar SMS con Twilio
15
16 # Nuevas importaciones para el envío de correo:
17 from dotenv import load_dotenv
18 import smtplib
19 import ssl
20 from email.mime.multipart import MIMEMultipart
21 from email.mime.text import MIMEText
22
23 # Cargar variables de entorno desde .env (si las utilizas):
24 load_dotenv()
25
26 # ----- CONFIGURACIÓN DE LOGGING PARA AUTENTICACIÓN -----
27 logger_auth = logging.getLogger('auth_logger')
28 logger_auth.setLevel(logging.INFO)
29 fh = logging.FileHandler("auth.log", encoding="utf-8")
30 fh.setLevel(logging.INFO)
31 formatter = logging.Formatter("%(asctime)s - %(message)s", datefmt="%Y-%m-%d %H:%M:%S")
32 fh.setFormatter(formatter)
33 logger_auth.addHandler(fh)
34 #
```

## Lineas 1-34:

### Linea 1: import tkinter as tk

- **tkinter**: Biblioteca estándar de Python para crear interfaces gráficas de usuario (GUI).
- **as tk**: Se asigna un alias para usar tk.Label, tk.Button, etc., en lugar de tkinter.Label, etc.

### Linea 2: from tkinter import ttk, messagebox, filedialog

- **ttk**: Submódulo de tkinter con widgets mejorados (estilo moderno).
- **messagebox**: Para mostrar cuadros de diálogo de alerta, información, error, etc.
- **filedialog**: Para abrir ventanas de selección de archivos o carpetas.

### Linea 3: from datetime import datetime

- Permite trabajar con fechas y horas actuales (datetime.now()), convertir formatos, etc.

### Linea 4: import time

- Proporciona funciones para manejar tiempos de espera (sleep), medir tiempo, etc.

**Línea 5:** import threading

- Permite ejecutar **tareas en segundo plano** sin bloquear la interfaz gráfica.

**Línea 6:** import shutil

- Sirve para copiar, mover, eliminar archivos y manipular estructuras de directorios.

**Línea 7:** import cv2

- Importa OpenCV, biblioteca de visión artificial. Se usa para procesar imágenes, capturar video, etc.

**Línea 8:** from PIL import Image, ImageTk

- PIL (Pillow): Librería para trabajar con imágenes.
- Image: Para abrir, modificar, guardar imágenes.
- ImageTk: Para convertir imágenes en formato compatible con tkinter.

**Línea 9:** import socket

- Permite comunicaciones de red (por ejemplo, obtener IPs, hacer servidores, etc.).

**Línea 10:** import os

- Acceso al sistema operativo: rutas, variables de entorno, archivos, procesos.

**Línea 11:** import logging

- Módulo estándar para crear registros de eventos (logs). Útil para depuración y seguimiento.

**Línea 12:** from ultralytics import YOLO

- Importa el modelo **YOLO (You Only Look Once)** desde la librería ultralytics para realizar detección de objetos en tiempo real.

**Línea 13:** import winsound # Para reproducir sonido en Windows

- Permite emitir sonidos en sistemas Windows (ej. una alarma al detectar algo).

**Línea 14:** from twilio.rest import Client # Para enviar SMS con Twilio

- Cliente oficial de la API de Twilio para enviar mensajes de texto, llamadas, etc.

**Línea 17:** from dotenv import load\_dotenv

- Carga variables desde un archivo .env (por ejemplo, usuario, clave, API keys) para mayor seguridad.

**Línea 18:** import smtplib

- Librería para enviar correos usando el protocolo SMTP (Simple Mail Transfer Protocol).

**Línea 19:** import ssl

- Proporciona soporte para conexiones cifradas (TLS/SSL), usadas en envío seguro de emails.

**Línea 20:** from email.mime.multipart import MIME\_Multipart

**Línea 21:** from email.mime.text import MIMEText

- Permiten componer correos con texto plano o HTML.
- MIME\_Multipart: Cuerpo del correo que puede tener varias partes (texto + archivos).
- MIMEText: Define el contenido del mensaje (texto o HTML).

**Línea 27:** logger\_auth = logging.getLogger("auth\_logger")

- Crea un **logger** personalizado llamado "auth\_logger" para registrar eventos relacionados con la autenticación.

**Línea 28:** logger\_auth.setLevel(logging.INFO)

- Establece que el nivel mínimo de mensajes a registrar será INFO (no registra DEBUG, sí INFO, WARNING, ERROR...).

**Línea 29:** fh = logging.FileHandler("auth.log", encoding="utf-8")

- Crea un **manejador de archivo** (FileHandler) para guardar los logs en un archivo llamado auth.log.

**Línea 30:** fh.setLevel(logging.INFO)

- Define que este archivo recibirá mensajes desde nivel INFO hacia arriba.

**Línea 31:** formatter = logging.Formatter("%(asctime)s - %(message)s", datefmt="%Y-%m-%d %H:%M:%S")

- Establece el **formato** del mensaje del log. Incluirá:
  - %(asctime)s: Fecha y hora del evento
  - %(message)s: Contenido del mensaje

**Línea 32:** fh.setFormatter(formatter)

- Aplica el formato definido al manejador de archivo.

**Línea 33:** logger\_auth.addHandler(fh)

- Asocia el manejador al logger personalizado, de modo que cualquier mensaje enviado a logger\_auth se escriba en auth.log.

```

35     def obtener_ip_local():
36         try:
37             s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
38             s.connect(("8.8.8.8", 80))
39             ip_local = s.getsockname()[0]
40             s.close()
41             return ip_local
42         except Exception:
43             return "127.0.0.1"
44
45     class LoginApp:
46         def __init__(self):
47             self.login_window = tk.Tk()
48             self.login_window.title("Login - Sistema de Seguridad")
49             self.login_window.geometry("400x300")
50             self.login_window.configure(bg="#F1F1F1")
51             self.login_window.resizable(False, False)
52
53             self.failed_attempts = 0
54
55             style_login = ttk.Style()
56             style_login.theme_use("clam")
57             style_login.configure("TLabel", background="#F1F1F1", foreground="#E0E0E0", font=("Arial", 12))
58             style_login.configure("TEntry", fieldbackground="#F1F1F1", foreground="white", font=("Arial", 12))
59             style_login.configure("TButton", padding=10, relief="flat", background="#4A90E2", foreground="white",
60                                 font=("Arial", 12, "bold"))
61
62             main_frame = ttk.Frame(self.login_window, padding=20)
63             main_frame.place(relx=0.5, rely=0.5, anchor=tk.CENTER)
64
65             ttk.Label(main_frame, text="Usuario:").grid(row=0, column=0, sticky='w', pady=10)
66             self.user_entry = ttk.Entry(main_frame, width=30)
67             self.user_entry.grid(row=0, column=1, sticky='w')
68
69             ttk.Label(main_frame, text="Contraseña:").grid(row=1, column=0, sticky='w', pady=10)

```

### Lineas 36-69:

#### Linea 36: def obtener\_ip\_local():

- Define una función llamada obtener\_ip\_local que retorna la dirección IP local del equipo.

#### Linea 37: try:

- Inicia un bloque try para capturar errores potenciales durante la obtención de la IP.

#### Linea 38: s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

- Crea un socket de red (s) usando IPv4 (AF\_INET) y protocolo UDP (SOCK\_DGRAM).

#### Linea 39: s.connect(("8.8.8.8", 80))

- Conecta el socket a la IP pública de Google (8.8.8.8) en el puerto 80; no envía datos, solo fuerza al SO a elegir la interfaz de salida.

#### Linea 40: ip\_local = s.getsockname()[0]

- Obtiene la tupla (IP, puerto) asociada al socket y extrae la IP local asignada ([0]).

#### Linea 41: s.close()

- Cierra el socket para liberar recursos del sistema.

#### Linea 42: return ip\_local

- Devuelve la dirección IP local obtenida.

#### Linea 43: except Exception:

- Captura cualquier excepción que ocurra dentro del bloque try.

#### Linea 44: return "127.0.0.1"

- Si hay un error (por ejemplo, sin conexión a red), devuelve la IP de loopback como fallback.

**Línea 46:** class LoginApp:

- Define la clase LoginApp, responsable de la ventana de inicio de sesión del sistema.

**Línea 47:** def \_\_init\_\_(self):

- Método constructor que inicializa todos los componentes de la interfaz de login.

**Línea 48:** self.login\_window = tk.Tk()

- Crea la ventana principal de tkinter y la asigna a self.login\_window.

**Línea 49:** self.login\_window.title("Login - Sistema de Seguridad")

- Establece el título de la ventana como "Login – Sistema de Seguridad".

**Línea 50:** self.login\_window.geometry("400x300")

- Define el tamaño inicial de la ventana a 400×300 píxeles.

**Línea 51:** self.login\_window.configure(bg="#1E1E2F")

- Configura el color de fondo de la ventana (un gris oscuro).

**Línea 52:** self.login\_window.resizable(False, False)

- Deshabilita la opción de redimensionar la ventana en ancho y alto.

**Línea 54:** self.failed\_attempts = 0

- Inicializa un contador de intentos fallidos de login en cero.

**Línea 56:** style\_login = ttk.Style()

- Crea un objeto Style para personalizar los estilos de los widgets ttk.

**Línea 57:** style\_login.theme\_use('clam')

- Aplica el tema "clam" (neutral y moderno) a los widgets.

**Línea 58:** style\_login.configure("TLabel", background="#1E1E2F", foreground="#E0E0E0", font=("Arial", 12))

- Establece estilo para etiquetas: fondo oscuro, texto claro y fuente Arial 12.

**Línea 59:** style\_login.configure("TEntry", fieldbackground="#2A2A3D", foreground="white", font=("Arial", 12))

- Define estilo para campos de entrada: color de fondo medio oscuro y texto blanco.

**Línea 60:** style\_login.configure("TButton", padding=10, relief="flat", background="#4A90E2", foreground="white",

**Línea 61:** font=("Arial", 12, "bold"))

- Configura botones con 10 px de relleno, borde plano, azul de fondo, texto blanco y fuente negrita.

**Línea 63:** main\_frame = ttk.Frame(self.login\_window, padding=20)

- Crea un contenedor (Frame) con 20 px de padding en la ventana de login.

**Línea 64:** main\_frame.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

- Posiciona el main\_frame centrado en la ventana mediante coordenadas relativas.

**Línea 66:** ttk.Label(main\_frame, text="Usuario:").grid(row=0, column=0, sticky='w', pady=10)

- Inserta una etiqueta “Usuario:” en la fila 0, columna 0, alineada al oeste, con espacio vertical.

**Línea 67:** self.user\_entry = ttk.Entry(main\_frame, width=30)

- Crea el campo de texto para ingresar el usuario (30 caracteres de ancho).

**Línea 68:** self.user\_entry.grid(row=0, column=1)

- Coloca el Entry de usuario en la fila 0, columna 1 del main\_frame.

```

70     ttk.Label(main_frame, text="Contraseña:").grid(row=1, column=0, sticky='w', pady=10)
71     self.pass_entry = ttk.Entry(main_frame, width=30, show="*")
72     self.pass_entry.grid(row=1, column=1)
73
74     self.login_button = ttk.Button(main_frame, text="Acceder", command=self.verificar_login)
75     self.login_button.grid(row=2, column=0, columnspan=2, pady=20)
76
77     self.segundos_restantes = 0
78     self.contador_label = ttk.Label(main_frame, text="", font=("Arial", 10), foreground="#F5A623")
79     self.contador_label.grid(row=3, column=0, columnspan=2)
80
81 def verificar_login(self):
82     user = self.user_entry.get()
83     pwd = self.pass_entry.get()
84
85     if self.failed_attempts >= 3:
86         messagebox.showwarning("Bloqueado", "Demasiados intentos fallidos. Intenta de nuevo más tarde.")
87         return
88
89     if user == "Bowser" and pwd == "12345678":
90         logger_auth.info(f'LOGIN SUCCESS user={user}')
91         self.login_window.destroy()
92         app = Seguridadapp()
93         app.iniciar_app()
94     else:
95         logger_auth.info(f'LOGIN FAIL user={user}')
96         self.failed_attempts += 1
97         remaining = max(0, 3 - self.failed_attempts)
98         messagebox.showerror("Error", f"Usuario o contraseña incorrectos.\nIntentos restantes: {remaining}")
99
100    if self.failed_attempts >= 3:
101        self.login_button.config(state="disabled")
102        self.segundos_restantes = 30
103        self.actualizar_contador()
104        self.login_window.after(30000, self.reactivar_login)

```

Lin. 104, Col. 69 (52 seleccionada) Espacios: 4 UTF-8 CRLF {} Python 3.13.2

**Lineas 70-104:**

**Línea 70:** ttk.Label(main\_frame, text="Contraseña:").grid(row=1, column=0, sticky='w', pady=10)

- Inserta una etiqueta “Contraseña:” en main\_frame, fila 1 columna 0.
- sticky='w' alinea el texto a la izquierda; pady=10 agrega espacio vertical.

**Línea 71:** self.pass\_entry = ttk.Entry(main\_frame, width=30, show="\*")

- Crea un campo de entrada para la contraseña de 30 caracteres de ancho.
- show="\*" oculta los caracteres ingresados mostrando asteriscos.

**Línea 72:** self.pass\_entry.grid(row=1, column=1)

- Coloca el Entry de contraseña en la fila 1, columna 1 del main\_frame.

**Línea 74:** self.login\_button = ttk.Button(main\_frame, text="Acceder", command=self.verificar\_login)

- Crea un botón “Acceder” que, al pulsarlo, invoca el método self.verificar\_login.

**Línea 75:** self.login\_button.grid(row=2, column=0, columnspan=2, pady=20)

- Ubica el botón en la fila 2 abarcando dos columnas, con 20 px de espacio vertical.

**Línea 77:** self.segundos\_restantes = 0

- Inicializa en cero el contador de segundos que queda bloqueado tras 3 intentos fallidos.

**Línea 78:** self.contador\_label = ttk.Label(main\_frame, text="", font=("Arial", 10), foreground="#F5A623")

- Crea una etiqueta vacía para mostrar la cuenta regresiva de bloqueo.
- Usa fuente Arial 10 y texto color naranja claro (#F5A623).

**Línea 79:** self.contador\_label.grid(row=3, column=0, columnspan=2)

- Coloca la etiqueta de contador en la fila 3, cubriendo ambas columnas.

**Línea 81:** def verificar\_login(self):

- Define el método que valida usuario y contraseña al pulsar “Acceder”.

**Línea 82:** user = self.user\_entry.get()

- Obtiene el texto ingresado en el campo de usuario.

**Línea 83:** pwd = self.pass\_entry.get()

- Obtiene el texto (oculto) ingresado en el campo de contraseña.

**Línea 85:** if self.failed\_attempts >= 3:

- Verifica si ya hubo tres o más intentos fallidos seguidos.

**Línea 86:** messagebox.showwarning("Bloqueado", "Demasiados intentos fallidos. Intenta de nuevo más tarde.")

- Muestra un cuadro de advertencia indicando bloqueo temporal.

**Línea 87:** return

- Sale del método sin procesar más validaciones.

**Línea 89:** if user == "Bowser" and pwd == "12345678":

- Comprueba credenciales: usuario “Bowser” y contraseña “12345678”.

**Línea 90:** logger\_auth.info(f'LOGIN SUCCESS user="{user}"')

- Registra en el log de autenticación un mensaje de éxito.

**Línea 91:** self.login\_window.destroy()

- Cierra la ventana de login.

**Línea 92:** app = SeguridadApp()

- Instancia la clase principal de la aplicación de seguridad.

**Línea 93:** app.iniciar\_app()

- Lanza el bucle principal (mainloop) de la interfaz de seguridad.

**Línea 94:** else:

Si las credenciales no coinciden, ejecuta el bloqueo de falla.

**Línea 95:** logger\_auth.info(f'LOGIN FAIL user="{user}"')

- Registra en el log un mensaje de intento fallido.

**Línea 96:** self.failed\_attempts += 1

- Incrementa el contador de intentos fallidos.

**Línea 97:** remaining = max(0, 3 - self.failed\_attempts)

- Calcula cuántos intentos quedan antes del bloqueo (no negativo).

**Línea 98:** messagebox.showerror("Error", f"Usuario o contraseña incorrectos.\nIntentos restantes: {remaining}")

- Muestra un cuadro de error con los intentos que quedan.

**Línea 100:** if self.failed\_attempts >= 3:

- Si tras este fallo se llega a 3 intentos, activa el bloqueo.

**Línea 101:** self.login\_button.config(state="disabled")

- Deshabilita el botón de acceso para evitar más clics.

**Línea 102:** self.segundos\_restantes = 30

- Establece 30 s de bloqueo antes de reactivar el login.

**Línea 103:** self.actualizar\_contador()

- Inicia la función que actualiza cada segundo la etiqueta del contador.

**Línea 104:** self.login\_window.after(30000, self.reactivar\_login)

```

104     self.login_window.after(30000, self.reactivar_login)
105
106 def actualizar_contador(self):
107     if self.segundos_restantes > 0:
108         self.contador_label.config(text=f"Bloqueado. Intenta en {self.segundos_restantes}s")
109         self.segundos_restantes -= 1
110         self.login_window.after(1000, self.actualizar_contador)
111
112 def reactivar_login(self):
113     self.failed_attempts = 0
114     self.login_button.config(state="normal")
115     self.contador_label.config(text="")
116     messagebox.showinfo("Reactivado", "Ahora puedes intentar iniciar sesión de nuevo.")
117
118 def run(self):
119     self.login_window.mainloop()
120
121 class SeguridadApp:
122     def __init__(self):
123         self.dark_mode = True
124         self.live_activado = False
125         self.cap = None
126         self.hilo_live = None
127         self.imagen_capturada_path = None
128         self.model = YOLO("yolov8n.pt")
129         self.ultimos_objetos_detectados = [] # Almacenar los últimos objetos detectados
130
131     # Configuración de Twilio para enviar SMS
132     self.twilio_account_sid = "AC0d739fd805cc024dbba19958dc5fa1f3"
133     self.twilio_auth_token = "cde421f7afe63733ce76378b708e7ffe"
134     self.twilio_phone_number = "+18635922506"
135     self.destinatarios_phone_numbers = ["+51940317018", "+51925446899"]
136     self.twilio_client = Client(self.twilio_account_sid, self.twilio_auth_token)
137
138     if not os.path.exists("capturas"):
139         os.makedirs("capturas")
140
141

```

Lín. 104, Col. 69 (52 seleccionada) Espacios: 4 UTF-8 CRLF {} Python 3.13.2

**Lineas 106-138:**

**Línea 106:** def actualizar\_contador(self):

- Define el método que actualiza cada segundo la etiqueta de cuenta regresiva durante el bloqueo.

**Línea 107:** if self.segundos\_restantes > 0:

- Comprueba si aún quedan segundos por contar antes de reactivar el login.

**Línea 108:** self.contador\_label.config(text=f"Bloqueado. Intenta en {self.segundos\_restantes}s")

- Actualiza el texto de contador\_label mostrando cuántos segundos faltan.

**Línea 109:** self.segundos\_restantes -= 1

- Decrementa en 1 el valor de self.segundos\_restantes.

**Línea 110:** self.login\_window.after(1000, self.actualizar\_contador)

- Programa la siguiente llamada a este método tras 1 000 ms para continuar la cuenta atrás.

**Línea 112:** def reactivar\_login(self):

- Define el método que restaura el estado del login tras el periodo de bloqueo.

**Línea 113:** self.failed\_attempts = 0

- Reinicia el contador de intentos fallidos a cero.

**Línea 114:** self.login\_button.config(state="normal")

- Vuelve a habilitar el botón “Acceder”.

**Línea 115:** self.contador\_label.config(text="")

- Limpia el texto del contador.

**Línea 116:** messagebox.showinfo("Reactivado", "Ahora puedes intentar iniciar sesión de nuevo.")

- Muestra un cuadro informativo indicando que el login está disponible de nuevo.

**Línea 118:** def run(self):

- Define el método que inicia el bucle principal de la ventana de login.

**Línea 119:** self.login\_window.mainloop()

- Ejecuta el mainloop() de tkinter, mostrando la interfaz y gestionando eventos.

**Línea 121:** class SeguridadApp:

- Declara la clase principal de la aplicación de seguridad, que gestiona la cámara, detección y alertas.

**Línea 122:** def \_\_init\_\_(self):

- Método constructor que inicializa todos los atributos y la interfaz de SeguridadApp.

**Línea 123:** self.dark\_mode = True

- Variable booleana para controlar si la interfaz está en modo oscuro.

**Línea 124:** self.live\_activado = False

- Indica si el vídeo en vivo está activo o no.

**Línea 125:** self.cap = None

- Referencia al objeto VideoCapture de OpenCV, inicialmente None.

**Línea 126:** self.hilo\_live = None

- Referencia al hilo que procesa el vídeo en vivo, inicialmente None.

**Línea 127:** self.imagen\_capturada\_path = None

- Almacena la ruta de la última imagen capturada, inicialmente None.

**Línea 128:** self.model = YOLO("yolov8n.pt")

- Carga el modelo YOLOv8 (archivo yolov8n.pt) para detección de objetos.

**Línea 129:** self.ultimos\_objetos\_detectados = []

- Lista para guardar los nombres de los últimos objetos detectados en un fotograma.

**Línea 131:** # Configuración de Twilio para enviar SMS

- Comentario que indica el inicio de la sección de configuración de la API de Twilio.

**Línea 132:** self.twilio\_account\_sid = "AC0d739fd805cc024dbba19958dc5fa1f3"

- Almacena el SID de la cuenta de Twilio (identificador público).

**Línea 133:** self.twilio\_auth\_token = "cde421f7afe63733ce76378b708e7ffe"

- Guarda el token de autenticación de Twilio (secreto).

**Línea 134:** self.twilio\_phone\_number = "+18635922506"

- Número de teléfono registrado en Twilio desde el que se enviarán los SMS.

**Línea 135:** self.destinatarios\_phone\_numbers = ["+51940317018", "+51925446899"]

- Lista de números de destinatarios a los que se enviarán las alertas por SMS.

**Línea 136:** self.twilio\_client = Client(self.twilio\_account\_sid, self.twilio\_auth\_token)

- Crea el cliente de Twilio usando SID y token para enviar mensajes.

**Línea 138:** if not os.path.exists("capturas"):

- Comprueba si no existe la carpeta capturas en el directorio actual para almacenar imágenes.

```

139     |     os.makedirs("capturas", exist_ok=True)
140
141     |     self.root = tk.Tk()
142     |     self.root.title("🔒 Sistema de Seguridad Inteligente")
143     |     self.root.geometry("1200x800")
144     |     self.root.configure(bg="#1E1E2E")
145     |     self.root.protocol("WM_DELETE_WINDOW", self.cerrar_app)
146
147     |     menubar = tk.Menu(self.root)
148     |     file_menu = tk.Menu(menubar, tearoff=0)
149     |     file_menu.add_command(label="Salir", command=self.cerrar_app)
150     |     menubar.add_cascade(label="Archivo", menu=file_menu)
151
152     |     view_menu = tk.Menu(menubar, tearoff=0)
153     |     self.var_darkmode = tk.BooleanVar(value=self.dark_mode)
154     |     view_menu.add_checkbutton(label="Modo Oscuro", onvalue=True, offvalue=False, variable=self.var_darkmode,
155                               command=self.toggle_dark_mode)
156     |     menubar.add_cascade(label="Ver", menu=view_menu)
157     |     self.root.config(menu=menubar)
158
159     |     self.style = ttk.Style()
160     |     self.style.theme_use("clam")
161     |     self._configurar_estilos()
162
163     |     container = ttk.Frame(self.root)
164     |     container.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
165
166     |     self.video_frame = ttk.Frame(container, width=800, relief=tk.RIDGE, borderwidth=2)
167     |     self.video_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=(0, 10))
168
169     |     right_frame = ttk.Frame(container, width=350)
170     |     right_frame.pack(side=tk.RIGHT, fill=tk.Y, expand=False)
171
172     |     titulo = ttk.Label(self.video_frame, text="🔒 Sistema de Seguridad Inteligente", font=("Arial", 24, "bold"),
173                         foreground="#4A90E2")

```

Lín. 173, Col. 49 (21 seleccionada) Espacios: 4 UTF-8 CRLF {} Python 3.13.2 □

**Lineas 139-173:**

**Línea 139:** os.makedirs("capturas", exist\_ok=True)

- Crea la carpeta capturas si no existe.
- exist\_ok=True evita error si ya existe.

**Línea 141:** self.root = tk.Tk()

- Crea la ventana principal de la aplicación de seguridad.

**Línea 142:** self.root.title("🔒 Sistema de Seguridad Inteligente")

- Establece el título de la ventana con un emoji de candado.

**Línea 143:** self.root.geometry("1200x800")

- Define el tamaño inicial de la ventana a 1200×800 píxeles.

**Línea 144:** self.root.configure(bg="#1E1E2F")

- Configura el color de fondo de la ventana (gris oscuro).

**Línea 145:** self.root.protocol("WM\_DELETE\_WINDOW", self.cerrar\_app)

- Intercepta el cierre de ventana para ejecutar self.cerrar\_app y liberar recursos.

**Línea 147:** menubar = tk.Menu(self.root)

- Crea la barra de menús en la ventana principal.

**Línea 148:** file\_menu = tk.Menu(menubar, tearoff=0)

- Crea el menú “Archivo” sin la línea de separación flotante.

**Línea 149:** file\_menu.add\_command(label="Salir", command=self.cerrar\_app)

- Añade opción “Salir” que invoca self.cerrar\_app.

**Línea 150:** menubar.add\_cascade(label="Archivo", menu=file\_menu)

- Agrega el menú “Archivo” a la barra de menús.

**Línea 152:** view\_menu = tk.Menu(menubar, tearoff=0)

- Crea el menú “Ver” (para opciones de vista) sin tearoff.

**Línea 153:** self.var\_darkmode = tk.BooleanVar(value=self.dark\_mode)

- Variable booleana vinculada al estado de modo oscuro.

**Línea 154:** view\_menu.add\_checkbutton(label="Modo Oscuro", onvalue=True, offvalue=False, variable=self.var\_darkmode,

**Línea 155:** command=self.toggle\_dark\_mode)

- Añade casilla al menú “Ver” para activar/desactivar modo oscuro.
- Cambia self.var\_darkmode y llama a self.toggle\_dark\_mode.

**Línea 156:** menubar.add\_cascade(label="Ver", menu=view\_menu)

- Agrega el menú “Ver” a la barra de menús.

**Línea 157:** self.root.config(menu=menubar)

- Asocia la barra de menús (menubar) a la ventana.

**Línea 159:** self.style = ttk.Style()

- Crea un objeto Style para personalizar estilos en toda la app.

**Línea 160:** self.style.theme\_use('clam')

- Aplica el tema “clam” a los widgets ttk.

**Línea 161:** self.\_configurar\_estilos()

- Llama al método que ajusta colores y fuentes según el modo oscuro.

**Línea 163:** container = ttk.Frame(self.root)

- Crea un contenedor principal para organizar los paneles.

**Línea 164:** container.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

- Empaquea el contenedor para llenar la ventana con padding de 10 px.

**Línea 166:** self.video\_frame = ttk.Frame(container, width=800, relief=tk.RIDGE, borderwidth=2)

- Crea un marco para mostrar el video con borde en relieve.

**Línea 167:** self.video\_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=(0, 10))

- Empaquea video\_frame a la izquierda, llenando espacio y dejando 10 px a la derecha.

**Línea 169:** right\_frame = ttk.Frame(container, width=350)

- Crea el panel derecho para controles y registros.

**Línea 170:** right\_frame.pack(side=tk.RIGHT, fill=tk.Y, expand=False)

- Empaquea right\_frame a la derecha, llenando verticalmente.

**Línea 172:** titulo = ttk.Label(self.video\_frame, text="🕒 Sistema de Seguridad Inteligente", font=("Arial", 24, "bold"), foreground="#4A90E2")

**Línea 173:** foreground="#4A90E2")

- Crea una etiqueta grande con título y color azul para el encabezado del sistema.

```
174     titulo.pack(pady=10)
175
176     self.imagen_label = tk.Label(self.video_frame, borderwidth=2, relief="flat", bg="#2A2A3D")
177     self.imagen_label.pack(pady=0, fill=tk.BOTH, expand=True)
178
179     self.reloj_label = ttk.Label(self.video_frame, font=("Arial", 14), foreground="#E0E0E0")
180     self.reloj_label.pack(pady=5)
181
182     ttk.Label(right_frame, text="Configuración", font=("Arial", 16, "bold"), foreground="#4A90E2").pack(pady=(10, 5))
183
184     config_frame = ttk.Frame(right_frame)
185     config_frame.pack(pady=5)
186     ttk.Label(config_frame, text=" IP Local:").grid(row=0, column=0, sticky='e', pady=5)
187     self.ip_entry = ttk.Entry(config_frame, width=20, style="Custom.TEntry")
188     self.ip_entry.grid(row=0, column=1, pady=5, padx=(5, 0))
189     self.ip_entry.insert(0, obtener_ip_local())
190
191     ttk.Label(config_frame, text=" Usuario:").grid(row=1, column=0, sticky='e', pady=5)
192     self.user_display = ttk.Entry(config_frame, width=20, style="Custom.TEntry")
193     self.user_display.insert(0, 'Bowser')
194     self.user_display.config(state="readonly")
195     self.user_display.grid(row=1, column=1, pady=5, padx=(5, 0))
196
197     ttk.Separator(right_frame, orient='horizontal').pack(fill='x', pady=10)
198
199     ttk.Label(right_frame, text="Acciones", font=("Arial", 16, "bold"), foreground="#4A90E2").pack(pady=(5, 5))
200
201     acciones_frame = ttk.Frame(right_frame)
202     acciones_frame.pack(pady=5)
203
204     self.boton_live = ttk.Button(acciones_frame, text="▶ Live", command=self.toggle_live, style="Custom.TButton")
205     self.boton_live.grid(row=0, column=0, padx=5, pady=5, sticky='ew')
206
207     boton_capturar = ttk.Button(acciones_frame, text="📸 Capturar", command=self.capturar_imagen, style="Custom.TButton")
208     boton_capturar.grid(row=1, column=0, padx=5, pady=5, sticky='ew')
```

## Lineas 174-208:

**Línea 174:** titulo.pack(pady=10)

- Empaque la etiqueta del título con 10 px de padding vertical.

**Línea 176:** self.imagen\_label = tk.Label(self.video\_frame, borderwidth=2, relief="flat", bg="#2A2A3D")

- Crea un Label de tkinter en video\_frame que mostrará la imagen (video o captura).
- borderwidth=2, relief="flat": Define un borde fino y sin relieve.
- bg="#2A2A3D": Color de fondo oscuro para que destaque la imagen.

**Línea 177:** self.imagen\_label.pack(pady=10, fill=tk.BOTH, expand=True)

- Empaque el imagen\_label con 10 px de espacio vertical.
- fill=tk.BOTH, expand=True: Hace que ocupe todo el espacio disponible del marco.

**Línea 179:** self.reloj\_label = ttk.Label(self.video\_frame, font=("Arial", 14), foreground="#E0E0E0")

- Crea una etiqueta (ttk.Label) para mostrar la hora actual.
- Fuente Arial 14 y texto en color gris claro.

**Línea 180:** self.reloj\_label.pack(pady=5)

- Empaque reloj\_label con 5 px de padding vertical.

**Línea 182:** ttk.Label(right\_frame, text="Configuración", font=("Arial", 16, "bold"), foreground="#4A90E2").pack(pady=(10, 5))

- En el panel derecho, crea y empaque un subtítulo “Configuración”.
- Fuente Arial 16 negrita, color azul, con 10 px arriba y 5 px abajo.

**Línea 184:** config\_frame = ttk.Frame(right\_frame)

- Crea un nuevo Frame dentro de right\_frame para agrupar controles de configuración.

**Línea 185:** config\_frame.pack(pady=5)

- Empaque el config\_frame con 5 px de padding vertical.

**Línea 186:** ttk.Label(config\_frame, text="  IP Local:").grid(row=0, column=0, sticky='e', pady=5)

- Etiqueta “IP Local:” en la fila 0, columna 0, alineada a la derecha (sticky='e').

**Línea 187:** self.ip\_entry = ttk.Entry(config\_frame, width=20, style="Custom.TEntry")

- Campo de texto para mostrar la IP local, 20 caracteres de ancho, con estilo personalizado.

**Línea 188:** self.ip\_entry.grid(row=0, column=1, pady=5, padx=(5, 0))

- Coloca el campo de IP en la fila 0, columna 1, con 5 px de margen a la izquierda.

**Línea 189:** self.ip\_entry.insert(0, obtener\_ip\_local())

- Inserta en el campo la IP obtenida por la función obtener\_ip\_local().

**Línea 191:** ttk.Label(config\_frame, text="👤 Usuario:").grid(row=1, column=0, sticky='e', pady=5)

- Etiqueta “Usuario:” en fila 1, columna 0, alineada a la derecha.

**Línea 192:** self.user\_display = ttk.Entry(config\_frame, width=20, style="Custom.TEntry")

- Campo de texto para mostrar el nombre de usuario, con el mismo ancho y estilo.

**Línea 193:** self.user\_display.insert(0, "Bowser")

- Inserta el texto fijo “Bowser” en el campo de usuario.

**Línea 194:** self.user\_display.config(state="readonly")

- Configura el campo para que sea de solo lectura, impidiendo ediciones.

**Línea 195:** self.user\_display.grid(row=1, column=1, pady=5, padx=(5, 0))

- Empaquea el campo de usuario en fila 1, columna 1, con márgenes similares al anterior.

**Línea 197:** ttk.Separator(right\_frame, orient='horizontal').pack(fill='x', pady=10)

- Añade un separador horizontal que cruza todo el ancho del panel derecho, con 10 px de espacio.

**Línea 199:** ttk.Label(right\_frame, text="Acciones", font=("Arial", 16, "bold"), foreground="#4A90E2").pack(pady=(5, 5))

- Subtítulo “Acciones” para la sección de botones, estilo similar al de “Configuración”.

**Línea 201:** acciones\_frame = ttk.Frame(right\_frame)

- Crea un marco para agrupar los botones de acción.

**Línea 202:** acciones\_frame.pack(pady=5)

- Empaquea acciones\_frame con 5 px de padding vertical.

**Línea 204:** self.boton\_live = ttk.Button(acciones\_frame, text="▶ Live", command=self.toggle\_live, style="Custom.TButton")

- Botón “Live” que llama a toggle\_live y usa estilo personalizado.

**Línea 205:** self.boton\_live.grid(row=0, column=0, padx=5, pady=5, sticky='ew')

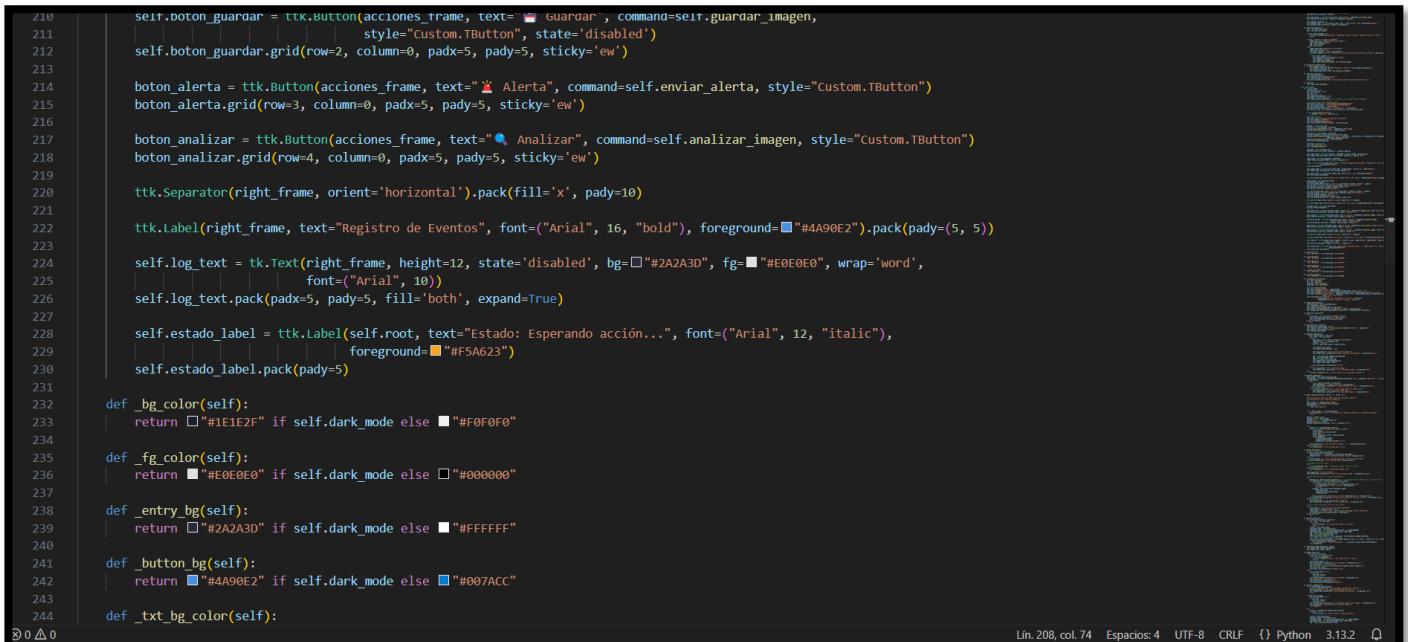
- Coloca el botón en fila 0, columna 0, expandiéndose en el ancho (sticky='ew').

**Línea 207:** boton\_capturar = ttk.Button(acciones\_frame, text="📸 Capturar", command=self.capturar\_imagen, style="Custom.TButton")

- Botón “Capturar” que invoca capturar\_imagen para sacar una foto del vídeo.

**Línea 208:** boton\_capturar.grid(row=1, column=0, padx=5, pady=5, sticky='ew')

- Ubica el botón de captura justo debajo de “Live”, con el mismo margen y expansión.



```
210 self.boton_guardar = ttk.Button(acciones_frame, text="💾 Guardar", command=self.guardar_imagen, style="Custom.TButton", state='disabled')
211 self.boton_guardar.grid(row=2, column=0, padx=5, pady=5, sticky='ew')
212
213 boton_alerta = ttk.Button(acciones_frame, text="⚠ Alerta", command=self.enviar_alerta, style="Custom.TButton")
214 boton_alerta.grid(row=3, column=0, padx=5, pady=5, sticky='ew')
215
216 boton_analizar = ttk.Button(acciones_frame, text="🔍 Analizar", command=self.analizar_imagen, style="Custom.TButton")
217 boton_analizar.grid(row=4, column=0, padx=5, pady=5, sticky='ew')
218
219 ttk.Separator(right_frame, orient='horizontal').pack(fill='x', pady=10)
220
221 ttk.Label(right_frame, text="Registro de Eventos", font=("Arial", 16, "bold"), foreground="#4A90E2").pack(pady=(5, 5))
222
223 self.log_text = tk.Text(right_frame, height=12, state='disabled', bg="#2A2A3D", fg="#E0E0E0", wrap='word',
224     font=("Arial", 10))
225 self.log_text.pack(padx=5, pady=5, fill='both', expand=True)
226
227 self.estado_label = ttk.Label(self.root, text="Estado: Esperando acción...", font=("Arial", 12, "italic"),
228     foreground="#F5A623")
229 self.estado_label.pack(pady=5)
230
231 def _bg_color(self):
232     return "#E1E1F" if self.dark_mode else "#F0F0F0"
233
234 def _fg_color(self):
235     return "#E0E0E0" if self.dark_mode else "#000000"
236
237 def _entry_bg(self):
238     return "#2A2A3D" if self.dark_mode else "#FFFFFF"
239
240 def _button_bg(self):
241     return "#4A90E2" if self.dark_mode else "#007ACC"
242
243 def _txt_bg_color(self):
```

Lín. 208, col. 74 Espacios: 4 UTF-8 CRLF {} Python 3.13.2

**Lineas 210-244:**

**Línea 210:** self.boton\_guardar = ttk.Button(acciones\_frame, text="💾 Guardar", command=self.guardar\_imagen,

- Crea el botón “Guardar” que invoca self.guardar\_imagen.

**Línea 211:** style="Custom.TButton", state='disabled')

- Aplica estilo personalizado y lo inicia deshabilitado (state='disabled') hasta que haya imagen capturada.

**Línea 212:** self.boton\_guardar.grid(row=2, column=0, padx=5, pady=5, sticky='ew')

- Ubica el botón en fila 2, columna 0, con márgenes y expansión horizontal.

**Línea 214:** boton\_alerta = ttk.Button(acciones\_frame, text="⚠ Alerta", command=self.enviar\_alerta, style="Custom.TButton")

- Crea el botón “Alerta” que llama a self.enviar\_alerta al pulsar.

**Línea 215:** boton\_alerta.grid(row=3, column=0, padx=5, pady=5, sticky='ew')

- Coloca el botón de alerta en la siguiente fila con el mismo espaciado.

**Línea 217:** boton\_analizar = ttk.Button(acciones\_frame, text="🔍 Analizar", command=self.analizar\_imagen, style="Custom.TButton")

- Botón “Analizar” para ejecutar self.analizar\_imagen en la imagen capturada.

**Línea 218:** boton\_analizar.grid(row=4, column=0, padx=5, pady=5, sticky='ew')

- Ubica el botón de análisis en fila 4 del marco de acciones.

**Línea 220:** ttk.Separator(right\_frame, orient='horizontal').pack(fill='x', pady=10)

- Añade un separador horizontal en el panel derecho con 10 px de espacio.

**Línea 222:** ttk.Label(right\_frame, text="Registro de Eventos", font=("Arial", 16, "bold"), foreground="#4A90E2").pack(pady=(5, 5))

- Título “Registro de Eventos” para el área de log, con estilo consistente.

**Línea 224:** self.log\_text = tk.Text(right\_frame, height=12, state='disabled', bg="#2A2A3D", fg="#E0E0E0", wrap='word',

- Crea un widget Text para mostrar los eventos de la aplicación.
- height=12: 12 líneas de alto.
- state='disabled': Lectura solamente.
- bg, fg: colores de fondo y texto según modo oscuro.
- wrap='word': Ajuste por palabra.

**Línea 225:** font=("Arial", 10))

- Fuente Arial tamaño 10 para el texto de log.

**Línea 226:** self.log\_text.pack(padx=5, pady=5, fill='both', expand=True)

- Empaque la área de log con padding y permite que crezca.

**Línea 228:** self.estado\_label = ttk.Label(self.root, text="Estado: Esperando acción...", font=("Arial", 12, "italic"),

- Etiqueta de estado en la parte inferior de la ventana principal.
- Texto inicial “Estado: Esperando acción...”, fuente Arial 12 cursiva.

**Línea 229:** foreground="#F5A623")

- Texto en color naranja claro (#F5A623) para destacar el estado.

**Línea 230:** self.estado\_label.pack(pady=5)

- Empaque la etiqueta de estado con 5 px de espacio vertical.

**Línea 232:** def \_bg\_color(self):

- Método interno que retorna el color de fondo según el modo oscuro.

**Línea 233:** return "#1E1E2F" if self.dark\_mode else "#F0F0F0"

- Si self.dark\_mode es True, fondo oscuro; si no, claro.

**Línea 235:** def \_fg\_color(self):

- Método que retorna el color de primer plano (texto) según el modo.

**Línea 236:** return "#E0E0E0" if self.dark\_mode else "#000000"

- Texto claro en modo oscuro, negro en modo claro.

**Línea 238:** def \_entry\_bg(self):

- Método que retorna el color de fondo para widgets de entrada.

**Línea 239:** return "#2A2A3D" if self.dark\_mode else "#FFFFFF"

- Fondo oscuro o blanco según el modo.

**Línea 241:** def \_button\_bg(self):

- Método que retorna el color de fondo para botones.

**Línea 242:** return "#4A90E2" if self.dark\_mode else "#007ACC"

- Botón azul claro en modo oscuro, azul diferente en claro.

**Línea 244:** def \_txt\_bg\_color(self):

- Método que retorna el color de fondo para el área de texto (log) según el modo.

```

245     return "#2A2A3D" if self.dark_mode else "#FFFFFF"
246
247     def _txt_fg_color(self):
248         return "#E0E0E0" if self.dark_mode else "#000000"
249
250     def _configurar_estilos(self):
251         bg = self._bg_color()
252         fg = self._fg_color()
253         entry_bg = self._entry_bg()
254         button_bg = self._button_bg()
255
256         self.root.configure(bg=bg)
257         self.style.configure("TFrame", background=bg)
258         self.style.configure(" TLabel", background=bg, foreground=fg, font=("Arial", 11))
259         self.style.configure("custom.TEntry", fieldbackground=entry_bg, foreground=fg, font=("Arial", 11))
260         self.style.configure("custom.TButton", padding=10, relief="flat", background=button_bg, foreground="white",
261                             font=("Arial", 10, "bold"))
262         self.style.map("custom.TButton",
263                         background=[("active", "#357ABD"), ("!active", button_bg)],
264                         foreground=[("active", "white"), ("!active", "white")])
265
266     def toggle_dark_mode(self):
267         self.dark_mode = self.var_darkmode.get()
268         self._configurar_estilos()
269         self.imagen_label.config(bg=self._bg_color())
270         self.log_text.config(bg=self._txt_bg_color(), fg=self._txt_fg_color())
271         self.estado_label.config(background=self._bg_color(), foreground=self._fg_color())
272
273     def actualizar_reloj(self):
274         try:
275             hora_actual = datetime.now().strftime("%H:%M:%S")
276             self.reloj_label.config(text=f" {hora_actual}")
277             self.root.after(1000, self.actualizar_reloj)
278         except tk.TclError:
279             pass
280

```

Lín. 244, Col. 29 (24 seleccionada) Espacios: 4 UTF-8 CRLF {} Python 3.13.2 Q

Lineas 245-279:

**Línea 245:** return "#2A2A3D" if self.dark\_mode else "#FFFFFF"

- Retorna el color de fondo para el área de texto según self.dark\_mode.

**Línea 247:** def \_txt\_fg\_color(self):

- Define el método que retorna el color de primer plano (texto) para el área de texto.

**Línea 248:** return "#E0E0E0" if self.dark\_mode else "#000000"

- Si self.dark\_mode es True, devuelve gris claro; si no, negro.

**Línea 250:** def \_configurar\_estilos(self):

- Método interno que aplica todos los estilos (ttk.Style) basados en los colores calculados.

**Línea 251:** bg = self.\_bg\_color()

- Guarda en bg el color de fondo general.

**Línea 252:** fg = self.\_fg\_color()

- Guarda en fg el color de texto general.

**Línea 253:** entry\_bg = self.\_entry\_bg()

- Guarda en entry\_bg el color de fondo para campos de entrada.

**Línea 254:** button\_bg = self.\_button\_bg()

- Guarda en button\_bg el color de fondo para botones.

**Línea 256:** self.root.configure(bg=bg)

- Aplica el color de fondo a la ventana principal.

**Línea 257:** self.style.configure("TFrame", background=bg)

- Configura el estilo de todos los Frame con el fondo bg.

**Línea 258:** self.style.configure(" TLabel", background=bg, foreground=fg, font=("Arial", 11))

- Ajusta etiquetas con fondo bg, texto fg y fuente Arial 11.

**Línea 259:** self.style.configure("Custom.TEntry", fieldbackground=entry\_bg, foreground=fg, font=("Arial", 11))

- Estilo para campos de entrada con fondo entry\_bg, texto fg.

**Línea 260:** self.style.configure("Custom.TButton", padding=10, relief="flat", background=button\_bg, foreground="white",

**Línea 261:** font=("Arial", 10, "bold"))

- Define botones con 10 px de padding, borde plano, fondo button\_bg y fuente negrita.

**Línea 262:** self.style.map("Custom.TButton",

- Inicia el mapeo de estados para el estilo de botón personalizado.

**Línea 263:** background=[("active", "#357ABD"), ("!active", button\_bg)],

- Cambia el color de fondo a #357ABD cuando el botón esté activo.

**Línea 264:** foreground=[("active", "white"), ("!active", "white")])

- Mantiene el texto blanco tanto activo como inactivo.

**Línea 266:** def toggle\_dark\_mode(self):

- Método que alterna entre modo oscuro y claro.

**Línea 267:** self.dark\_mode = self.var\_darkmode.get()

- Actualiza self.dark\_mode según el valor de la casilla del menú.

**Línea 268:** self.\_configurar\_estilos()

- Vuelve a aplicar todos los estilos para reflejar el nuevo modo.

**Línea 269:** self.imagen\_label.configure(bg=self.\_bg\_color())

- Ajusta el fondo del label de la imagen según el modo actual.

**Línea 270:** self.log\_text.configure(bg=self.\_txt\_bg\_color(), fg=self.\_txt\_fg\_color())

- Actualiza los colores de fondo y texto del área de log.

**Línea 271:** self.estado\_label.configure(background=self.\_bg\_color(), foreground=self.\_fg\_color())

- Cambia los colores de la etiqueta de estado.

**Línea 273:** def actualizar\_reloj(self):

- Método que actualiza el reloj cada segundo.

**Línea 274:** try:

- Bloque try para capturar posibles errores de tkinter si la ventana ya está cerrada.

**Línea 275:** hora\_actual = datetime.now().strftime("%H:%M:%S")

- Obtiene la hora actual en formato HH:MM:SS.

**Línea 276:** self.reloj\_label.config(text=f"<img alt='clock icon' data-bbox='465 788 485 803' style='vertical-align: middle; height: 1em;"/> {hora\_actual}")

- Actualiza el texto de reloj\_label con la hora.

**Línea 277:** self.root.after(1000, self.actualizar\_reloj)

- Programa la siguiente actualización tras 1 000 ms (1 segundo).

**Línea 278:** except tk.TclError:

- Captura la excepción TclError si el widget ya no existe.

**Línea 279:** pass

- Ignora el error silenciosamente, deteniendo la actualización.

```
281     def log_evento(self, mensaje):
282         self.log_text.config(state='normal')
283         self.log_text.insert(tk.END, f'{datetime.now().strftime("%H:%M:%S")} - {mensaje}\n')
284         self.log_text.config(state='disabled')
285         self.log_text.see(tk.END)
286
287     def capturar_imagen(self):
288         if self.live_activado and self.cap:
289             ret, frame = self.cap.read()
290             if ret:
291                 timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
292                 nombre_archivo = f'{timestamp}.png'
293                 carpeta = "capturas"
294                 ruta = os.path.join(carpeta, nombre_archivo)
295
296                 cv2.imwrite(ruta, frame)
297                 self.imagen_capturada_path = ruta
298
299                 self.log_evento("✅ Imagen capturada correctamente.")
300                 self.estado_label.config(text="✅ Imagen capturada correctamente.", foreground="green")
301
302                 img = Image.open(self.imagen_capturada_path)
303                 img = img.resize((800, 500))
304                 img_tk = ImageTk.PhotoImage(img)
305                 self.imagen_label.config(image=img_tk)
306                 self.imagen_label.image = img_tk
307
308                 self.boton_guardar.config(state='normal')
309             else:
310                 self.log_evento("❌ Error capturando imagen.")
311                 self.estado_label.config(text="❌ Error capturando imagen.", foreground="red")
312         else:
313             messagebox.showinfo("Info", "No hay video en vivo activo para capturar.")
314
315     def guardar_imagen(self):
316         ruta_origen = self.imagen_capturada_path
```

Lin. 279, Col. 17 (4 seleccionada) Espacios: 4 UTF-8 CRLF {} Python 3.13.2 Q

**Líneas 281-316:**

**Línea 281:** def log\_evento(self, mensaje):

- Define el método log\_evento que añade entradas al widget de texto de registro.

**Línea 282:** self.log\_text.config(state='normal')

- Cambia el estado del Text de “disabled” a “normal” para permitir insertar texto.

**Línea 283:** self.log\_text.insert(tk.END, f'{datetime.now().strftime('%H:%M:%S')}- {mensaje}\n')

- Inserta en la última línea (tk.END) la hora actual y el mensaje recibido, seguido de salto de línea.

**Línea 284:** self.log\_text.config(state='disabled')

- Vuelve a poner el widget en modo “disabled” para prevenir edición manual.

**Línea 285:** self.log\_text.see(tk.END)

- Desplaza la vista hasta el final del texto, mostrando la última entrada agregada.

**Línea 287:** def capturar\_imagen(self):

- Define el método que captura un fotograma del vídeo en vivo cuando está activo.

**Línea 288:** if self.live\_activado and self.cap:

- Comprueba que el vídeo en vivo esté activado y que el objeto VideoCapture exista.

**Línea 289:** ret, frame = self.cap.read()

- Lee un fotograma de la cámara; ret indica éxito y frame contiene la imagen.

**Línea 290:** if ret:

- Si la lectura fue exitosa, ejecuta el bloque para guardar y mostrar la imagen.

**Línea 291:** timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S")

- Genera una cadena con fecha y hora para nombrar el archivo:  
YYYYMMDD\_HHMMSS.

**Línea 292:** nombre\_archivo = f"{timestamp}.png"

- Crea el nombre del archivo añadiendo la extensión .png.

**Línea 293:** carpeta = "capturas"

- Define la carpeta donde se almacenarán las capturas.

**Línea 294:** ruta = os.path.join(carpeta, nombre\_archivo)

- Construye la ruta completa (carpeta/nombre\_archivo) de forma portable.

**Línea 296:** cv2.imwrite(ruta, frame)

- Guarda la imagen (frame) en disco en la ruta especificada en formato PNG.

**Línea 297:** self.imagen\_capturada\_path = ruta

- Almacena la ruta de la imagen capturada en un atributo para uso posterior.

**Línea 299:** self.log\_evento("📸 Imagen capturada correctamente.")

- Registra en el log un mensaje indicando que la captura fue exitosa.

**Línea 300:** self.estado\_label.config(text="✅ Imagen capturada correctamente.", foreground="green")

- Actualiza la etiqueta de estado con un mensaje de éxito en color verde.

**Línea 302:** img = Image.open(self.imagen\_capturada\_path)

- Abre la imagen usando Pillow para poder manipularla.

**Línea 303:** img = img.resize((800, 500))

- Redimensiona la imagen a 800×500 píxeles para que encaje en el Label.

**Línea 304:** img\_tk = ImageTk.PhotoImage(img)

- Convierte la imagen a un objeto compatible con tkinter.

**Línea 305:** self.imagen\_label.config(image=img\_tk)

- Asigna la imagen convertida al Label de la interfaz.

**Línea 306:** self.imagen\_label.image = img\_tk

- Guarda referencia a img\_tk en el widget para evitar que el recolector de basura la destruya.

**Línea 308:** self.boton\_guardar.config(state='normal')

- Habilita el botón “Guardar” ahora que existe una imagen capturada.

**Línea 309:** else:

- Si ret es False, la lectura del fotograma falló, se ejecuta el bloque de error.

**Línea 310:** self.log\_evento("✖ Error capturando imagen.")

- Registra en el log el fallo de captura.

**Línea 311:** self.estado\_label.config(text="✖ Error capturando imagen.", foreground="red")

- Muestra mensaje de error en rojo en la etiqueta de estado.

**Línea 312:** else:

- Si no hay vídeo en vivo activo o self.cap es None, se muestra información.

**Línea 313:** messagebox.showinfo("Info", "No hay video en vivo activo para capturar.")

- Muestra un cuadro de información indicando que no puede capturar sin vídeo.

**Línea 314:** (*línea en blanco*)

**Línea 315:** def guardar\_imagen(self):

- Define el método que copia la imagen capturada a una ubicación elegida por el usuario.

**Línea 316:** ruta\_origen = self.imagen\_capturada\_path

- Asigna a ruta\_origen la ruta almacenada de la última captura para copiarla.

```
317 ruta_destino = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG files", "*.png")])
318 if ruta_destino:
319     try:
320         shutil.copy(ruta_origen, ruta_destino)
321         self.log_evento(f"🕒 Imagen guardada en {ruta_destino}")
322         self.estado_label.config(text="🕒 Imagen guardada con éxito.", foreground="green")
323     except Exception as e:
324         messagebox.showerror("Error", f"No se pudo guardar la imagen: {e}")
325         self.log_evento(f"✖ Error guardando imagen: {e}")
326         self.estado_label.config(text="✖ Error guardando imagen.", foreground="red")
327
328 def enviar_email_alerta(self, asunto: str, cuerpo: str):
329     """
330     Envía un correo de alerta vía SMTP usando TLS (smtp.gmail.com:587).
331     Los destinatarios fijos: angel.mejia.e@uni.pe.
332     """
333     email_sender = os.getenv("EMAIL_SENDER")
334     email_password = os.getenv("EMAIL_PASSWORD")
335     destinatarios = [
336         "angel.mejia.e@uni.pe"
337     ]
338
339     if not email_sender or not email_password:
340         self.log_evento("✖ Error: no se encontraron credenciales de email en variables de entorno.")
341         return
342
343     mensaje = MIMEMultipart()
344     mensaje["From"] = email_sender
345     mensaje["To"] = ", ".join(destinatarios)
346     mensaje["Subject"] = asunto
347     mensaje.attach(MIMEText(cuerpo, "plain", _charset="utf-8"))
348
349     try:
350         context = ssl.create_default_context()
351         with smtplib.SMTP("smtp.gmail.com", 587) as server:
```

**Lineas 317-351:**

**Línea 317:** ruta\_destino = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG files", "\*.\*")])

- Abre un diálogo para que el usuario elija dónde guardar la imagen.
- defaultextension: sugiere ".png"; filetypes: filtra a archivos PNG.

**Línea 318:** if ruta\_destino:

- Comprueba si el usuario seleccionó una ruta (no canceló el diálogo).

**Línea 319:** try:

- Inicia un bloque para intentar copiar el archivo y capturar posibles errores.

**Línea 320:** shutil.copy(ruta\_origen, ruta\_destino)

- Copia el archivo desde la ruta original (ruta\_origen) a la ruta destino.

**Línea 321:** self.log\_evento(f"💾 Imagen guardada en {ruta\_destino}")

- Registra en el log el éxito indicando la ruta donde se guardó.

**Línea 322:** self.estado\_label.config(text="💾 Imagen guardada con éxito.", foreground="green")

- Muestra en la interfaz un mensaje de éxito en color verde.

**Línea 323:** except Exception as e:

- Captura cualquier excepción (e) que ocurra durante la copia.

**Línea 324:** messagebox.showerror("Error", f"No se pudo guardar la imagen: {e}")

- Muestra un cuadro de error con el mensaje de la excepción.

**Línea 325:** self.log\_evento(f"❌ Error guardando imagen: {e}")

- Registra en el log el detalle del error.

**Línea 326:** self.estado\_label.config(text="❌ Error guardando imagen.", foreground="red")

- Actualiza la etiqueta de estado con un mensaje de fallo en rojo.

**Línea 328:** def enviar\_email\_alerta(self, asunto: str, cuerpo: str):

- Define el método que envía un correo de alerta usando SMTP.
- Parámetros: asunto (subject) y cuerpo (body) del mensaje.

**Línea 329:** """

- Inicio de la cadena de documentación (docstring) del método.

**Línea 330:** Envía un correo de alerta vía SMTP usando TLS (smtp.gmail.com:587).

- Describe que emplea TLS por el puerto 587 de Gmail.

**Línea 331:** Los destinatarios fijos: [angel.mejia.e@uni.pe](mailto:angel.mejia.e@uni.pe).

- Indica que el correo siempre se envía a esa dirección fija.

**Línea 332:** """"

- Fin de la docstring.

**Línea 333:** email\_sender = os.getenv("EMAIL\_SENDER")

- Obtiene de las variables de entorno el correo remitente.

**Línea 344:** email\_password = os.getenv("EMAIL\_PASSWORD")

- Obtiene de las variables de entorno la contraseña o app-password.

**Línea 355:** destinatarios = [

- Inicia una lista de destinatarios a quienes se enviará el correo.

**Línea 366:** [angel.mejia.e@uni.pe](mailto:angel.mejia.e@uni.pe)

- Dirección fija incluida en la lista.

**Línea 377:** ]

- Cierre de la lista de destinatarios.

**Línea 399:** if not email\_sender or not email\_password:

- Verifica que existan ambas credenciales en las variables de entorno.

**Línea 400:** self.log\_evento("✖ Error: no se encontraron credenciales de email en variables de entorno.")

- Registra en el log la falta de credenciales.

**Línea 411:** return

- Sale del método si no están definidas las credenciales.

**Línea 433:** mensaje = MIMEMultipart()

- Crea el objeto multipart para componer el correo.

**Línea 444:** mensaje["From"] = email\_sender

- Establece el campo "From" con el correo remitente.

**Línea 455:** mensaje["To"] = ", ".join(destinatarios)

- Establece el campo “To” con la(s) dirección(es) de destinatario(s).

**Línea 346:** mensaje["Subject"] = asunto

- Asigna el asunto al correo.

**Línea 347:** mensaje.attach(MIMEText(cuerpo, "plain", \_charset="utf-8"))

- Adjunta el cuerpo como texto plano UTF-8 al mensaje.

## **Linea 349: try:**

- Inicia el bloque para la conexión SMTP y envío del correo.

**Línea 350:** context = ssl.create\_default\_context()

- Crea un contexto SSL/TLS predeterminado para cifrar la conexión.

**Línea 351:** con `smtplib.SMTP("smtp.gmail.com", 587)` como servidor:

- Abre la conexión con el servidor SMTP de Gmail en el puerto 587, gestionado automáticamente por el gestor de contexto.

```
352     server.ehlo()
353     server.starttls(context=context)
354     server.ehlo()
355     server.login(email_sender, email_password)
356     server.sendmail(
357         from_addr=email_sender,
358         to_addrs=destinatarios,
359         msg=mensaje.as_string().encode("utf-8")
360     )
361     self.log_evento(f"✉ Email de alerta enviado a: {', '.join(destinatarios)}")
362 except Exception as e:
363     self.log_evento(f"✗ Error enviando email: {e}")
364
365 def enviar_alerta(self):
366     # Mostrar mensaje de alerta emergente
367     if self.ultimos_objetos_detectados:
368         objetos_texto = ", ".join(self.ultimos_objetos_detectados)
369         mensaje_alerta = f"⚠ Alerta: Amenaza detectada. Objetos: {objetos_texto}"
370     else:
371         mensaje_alerta = "⚠ Alerta: Amenaza detectada. Sin objetos identificados."
372     messagebox.showwarning("⚠ Alerta de Seguridad", mensaje_alerta)
373
374     # Reproducir sonido de alerta
375     try:
376         winsound.Beep(1000, 500) # Frecuencia: 1000 Hz, Duración: 500 ms
377     except Exception as e:
378         self.log_evento(f"✗ Error reproduciendo sonido: {e}")
379
380     self.log_evento("⚠ Alerta enviada")
381     self.estado_label.config(text="⚠ Alerta de seguridad enviada.", foreground="orange")
382
383     # Enviar SMS con Twilio a múltiples destinatarios
384     try:
385         mensaje_sms = mensaje_alerta.replace("\n", " ") # Asegurar que el mensaje esté en una sola linea
386         for destinatario in self.destinatarios.phone_numbers:
```

### **Lineas 352-386:**

**Linea 352:** server.ehlo()

- Envía el comando EHLO al servidor SMTP para identificarse y obtener capacidades extendidas.

**Línea 353:** server.starttls(context=context)

- Inicia el cifrado TLS de la conexión usando el contexto SSL creado previamente.

**Línea 354:** server.ehlo()

- Vuelve a enviar EHLO tras activar TLS, para renegociar capacidades seguras.

**Línea 355:** server.login(email\_sender, email\_password)

- Autentica en el servidor SMTP con el correo remitente y su contraseña.

**Línea 356:** server.sendmail(

- Comienza la llamada para enviar el correo.

**Línea 357:** from\_addr=email\_sender,

- Parametra la dirección de origen del mensaje.

**Línea 358:** to\_addrs=destinatarios,

- Lista de destinatarios que recibirán el correo.

**Línea 359:** msg=mensaje.as\_string().encode("utf-8")

- Convierte el objeto MIME Multipart a cadena y luego a bytes UTF-8 para transmisión.

**Línea 360:** )

- Cierra la llamada a sendmail.

**Línea 361:** self.log\_evento(f"✉ Email de alerta enviado a: {', '.join(destinatarios)}")

- Registra en el log que el correo fue enviado exitosamente y a qué direcciones.

**Línea 362:** except Exception as e:

- Captura cualquier excepción que ocurra durante la conexión o envío de email.

**Línea 363:** self.log\_evento(f"✖ Error enviando email: {e}")

- Registra en el log el error ocurrido al tratar de enviar el correo.

**Línea 365:** def enviar\_alerta(self):

- Define el método que genera y envía las alertas (ventana, sonido, SMS y email).

**Línea 366:** # Mostrar mensaje de alerta emergente

- Comentario que indica que a continuación se mostrará un diálogo emergente.

**Línea 367:** if self.ultimos\_objetos\_detectados:

- Comprueba si hay objetos detectados para incluirlos en el mensaje de alerta.

**Línea 368:** objetos\_texto = ", ".join(self.ultimos\_objetos\_detectados)

- Une los nombres de los objetos detectados en una cadena separada por comas.

**Línea 369:** mensaje\_alerta = f"⚠ Alerta: Amenaza detectada. Objetos: {objetos\_texto}"

- Construye el texto de alerta incluyendo la lista de objetos.

**Línea 370:** else:

- Si no hubo detecciones, usa un mensaje genérico.

**Línea 371:** mensaje\_alerta = "⚠ Alerta: Amenaza detectada. Sin objetos identificados."

- Mensaje indicando que no se reconoció ningún objeto.

**Línea 372:** messagebox.showwarning("⚠ Alerta de Seguridad", mensaje\_alerta)

- Muestra un cuadro de advertencia con el título y el mensaje de alerta.

**Línea 374:** # Reproducir sonido de alerta

- Comentario que introduce la sección de reproducción de audio.

**Línea 375:** try:

- Inicia un bloque try para intentar emitir un sonido.

**Línea 376:** winsound.Beep(1000, 500) # Frecuencia: 1000 Hz, Duración: 500 ms

- Emite un pitido de 1000 Hz durante 500 ms en Windows.

**Línea 377:** except Exception as e:

- Captura errores si no se puede reproducir sonido (p.ej., en otro SO).

**Línea 378:** self.log\_evento(f"🔴 Error reproduciendo sonido: {e}")

- Registra en el log el error de audio.

**Línea 380:** self.log\_evento("⚠ Alerta enviada")

- Registra en el log que se inició el proceso de alerta.

**Línea 381:** self.estado\_label.config(text="⚠ Alerta de seguridad enviada.", foreground="orange")

- Actualiza la etiqueta de estado indicando éxito de alerta en color naranja.

**Línea 383:** # Enviar SMS con Twilio a múltiples destinatarios

- Comentario que introduce la sección de envío de SMS.

**Línea 384:** try:

- Inicia un bloque try para el envío de mensajes de texto.

**Línea 385:** mensaje\_sms = mensaje\_alerta.replace("\n", " ") # Asegurar que el mensaje esté en una sola línea

- Sustituye saltos de línea por espacios para envío en una sola línea de SMS.

**Línea 386:** for destinatario in self.destinatarios\_phone\_numbers:

- Comienza un bucle para iterar sobre cada número de teléfono destinatario.

```
387 # Validación básica del formato del número
388 if not destinatario.startswith("+") or len(destinatario) < 10:
389     self.log_evento("X Número invalido: {destinatario}")
390     continue
391 
392     mensaje = self.twilio_client.messages.create(
393         body=mensaje_sms,
394         from_=self.twilio_phone_number,
395         to=destinatario
396     )
397     self.log_evento(f"SMS enviado al número {destinatario}. SID: {mensaje.sid}")
398     self.estado_label.config(text=f"SMS enviado a todos los números con éxito.", foreground="green")
399 except Exception as e:
400     self.log_evento("X Error enviando SMS: {str(e)}")
401     self.estado_label.config(text="X Error enviando SMS.", foreground="red")
402 
403 # Enviar correo electrónico adicional (nueva parte)
404 try:
405     asunto_email = "Alerta de Seguridad: Objeto Detectado"
406     cuerpo_email = mensaje_alerta + "\n\n"
407     cuerpo_email += f"Fecha y hora: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"
408     self.enviar_email_alerta(asunto_email, cuerpo_email)
409 except Exception:
410     pass
411 
412 def mostrar_video(self):
413     while self.live_activado and self.cap:
414         ret, frame = self.cap.read()
415         if not ret:
416             self.log_evento("X No se pudo leer frame de la cámara")
417             break
418         results = self.model(frame)
419         annotated_frame = results[0].plot()
420         annotated_frame = cv2.cvtColor(annotated_frame, cv2.COLOR_BGR2RGB)
421         annotated_frame = cv2.resize(annotated_frame, (800, 500))
422         img = Image.fromarray(annotated_frame)
```

### Lineas 387-421:

**Línea 387:** # Validación básica del formato del número

- Comentario que señala que se hará una validación de los números de teléfono antes de enviar SMS.

**Línea 388:** if not destinatario.startswith"+" or len(destinatario)<10:

- Verifica que el número comience con "+" y tenga al menos 10 caracteres.

Línea 389: self.log\_evento(f"X Número inválido: {destinatario}")

- Registra en el log si el número no es válido.

### Línea 390: continue

- Salta al siguiente número si el actual no es válido.

Línea 391: mensaje = self.twilio\_client.messages.create(

- Llama a la API de Twilio para crear y enviar un mensaje SMS

Línea 392: body=mensaje\_sms

- Contenido del mensaje que se enviará

Línea 393: from =>self.twilio\_phone\_number

- Número registrado en Twilio que se usará como remitente

Línea 394: to=destinatario

- Número de teléfono del destinatario

Línea 395: )

- Fin de la llamada a create.

**Línea 396:** self.log\_evento(f" 📡 SMS enviado al número {destinatario}. SID: {mensaje.sid}")

- Registra en el log que el SMS fue enviado correctamente, incluyendo su SID.

**Línea 397:** self.estado\_label.config(text=" 📡 SMS enviado a todos los números con éxito.", foreground="green")

- Actualiza la etiqueta de estado indicando que los SMS fueron enviados exitosamente.

**Línea 398:** except Exception as e:

- Captura cualquier excepción que ocurra durante el envío de SMS.

**Línea 399:** self.log\_evento(f" ✗ Error enviando SMS: {str(e)}")

- Registra en el log el mensaje de error.

**Línea 400:** self.estado\_label.config(text=" ✗ Error enviando SMS.", foreground="red")

- Actualiza la etiqueta de estado indicando que hubo un error en el envío de SMS.

**Línea 402:** # Enviar correo electrónico adicional (nueva parte)

- Comentario que indica que se va a enviar un correo como parte de la alerta.

**Línea 403:** try:

- Inicia un bloque try para enviar el email adicional.

**Línea 404:** asunto\_email = "Alerta de Seguridad: Objeto Detectado"

- Define el asunto del correo electrónico de alerta.

**Línea 405:** cuerpo\_email = mensaje\_alerta + "\n\n"

- Crea el cuerpo del email comenzando con el mensaje de alerta.

**Línea 406:** cuerpo\_email += f"Fecha y hora: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"

- Añade al cuerpo la fecha y hora actuales.

**Línea 407:** self.enviar\_email\_alerta(asunto\_email, cuerpo\_email)

- Llama a la función que envía el correo con el asunto y el cuerpo definidos.

**Línea 408:** except Exception:

- Captura cualquier error ocurrido durante el intento de enviar el email.

**Línea 409:** pass

- Ignora el error (no se realiza acción si falla el envío).

**Línea 411:** def mostrar\_video(self):

- Define la función que transmite video en tiempo real y realiza detección de objetos.

**Línea 412:** while self.live\_activado and self.cap:

- Mientras la transmisión esté activada y la cámara abierta, continúa ejecutando.

**Línea 413:** ret, frame = self.cap.read()

- Captura un nuevo fotograma (frame) desde la cámara.

**Línea 414:** if not ret:

- Verifica si hubo un error al capturar el fotograma.

**Línea 415:** self.log\_evento("🔴 No se pudo leer frame de la cámara")

- Registra en el log si falló la captura del fotograma.

**Línea 416:** break

- Sale del bucle si no se pudo capturar un fotograma.

**Línea 417:** results = self.model(frame)

- Pasa el fotograma por el modelo YOLO para detectar objetos.

**Línea 418:** annotated\_frame = results[0].plot()

- Obtiene el fotograma anotado con los resultados de la detección (cajas, etiquetas).

**Línea 419:** annotated\_frame = cv2.cvtColor(annotated\_frame, cv2.COLOR\_BGR2RGB)

- Convierte el formato de color de BGR (OpenCV) a RGB (PIL/Tkinter).

**Línea 420:** annotated\_frame = cv2.resize(annotated\_frame, (800, 500))

- Redimensiona el fotograma a 800x500 píxeles.

**Línea 421:** img = Image.fromarray(annotated\_frame)

- Convierte el arreglo de NumPy (fotograma) a objeto Image de PIL.

```

422     img_tk = ImageTk.PhotoImage(img)
423     self.imagen_label.after(1, lambda img=img_tk: self.actualizar_imagen_label(img))
424     # Actualizar los últimos objetos detectados
425     self.ultimos_objetos_detectados = [self.model.names[int(cls)] for result in results for cls in result.bboxes.cls]
426     if self.ultimos_objetos_detectados:
427         self.log_evento(f"🕒 Objetos detectados: {', '.join(self.ultimos_objetos_detectados)}")
428         time.sleep(0.03)
429
430     def actualizar_imagen_label(self, img_tk):
431         self.imagen_label.config(image=img_tk)
432         self.imagen_label.image = img_tk
433
434     def toggle_live(self):
435         if not self.live_activado:
436             self.cap = cv2.VideoCapture(0)
437             if not self.cap.isOpened():
438                 messagebox.showerror("Error", "No se pudo abrir la cámara.")
439                 return
440             self.live_activado = True
441             self.estado_label.config(text="🔴 Live activado", foreground="green")
442             self.log_evento("🔴 Live activado")
443             self.hilo_live = threading.Thread(target=self.mostrar_video, daemon=True)
444             self.hilo_live.start()
445             self.botón_live.config(text="〃 Detener Live")
446         else:
447             self.live_activado = False
448             if self.cap:
449                 self.cap.release()
450                 self.cap = None
451             self.estado_label.config(text="〃 Live detenido", foreground="red")
452             self.log_evento("〃 Live detenido")
453             self.botón_live.config(text="● Live")
454             self.botón_guardar.config(state='disabled')
455

```

Líneas 422-456:

**Línea 422:** img\_tk = ImageTk.PhotoImage(img)

- Convierte la imagen de PIL en un objeto compatible con widgets de Tkinter (PhotoImage).

**Línea 423:** self.imagen\_label.after(1, lambda img=img\_tk:  
self.actualizar\_imagen\_label(img))

- Programa una llamada asíncrona para actualizar la etiqueta de imagen (Label) con la imagen capturada.

**Línea 424:** # Actualizar los últimos objetos detectados

- Comentario que indica que se va a actualizar la lista de objetos reconocidos.

**Línea 425:** self.ultimos\_objetos\_detectados = [self.model.names[int(cls)]] for result in  
results for cls in result.boxes.cls]

- Extrae las clases detectadas del modelo YOLO y las traduce a nombres legibles.

**Línea 426:** if self.ultimos\_objetos\_detectados:

- Comprueba si se detectó al menos un objeto en el fotograma.

**Línea 427:** self.log\_evento(f"🔍 Objetos detectados: {',  
'join(self.ultimos\_objetos\_detectados)}")

- Registra los nombres de los objetos detectados en el log.

**Línea 428:** time.sleep(0.03)

- Pausa brevemente la ejecución para no sobrecargar el hilo de procesamiento (30 ms por fotograma).

**Línea 430:** def actualizar\_imagen\_label(self, img\_tk):

- Define el método que actualiza visualmente la etiqueta de imagen con una nueva imagen.

**Línea 431:** self.imagen\_label.config(image=img\_tk)

- Asigna la imagen a la propiedad image del Label.

**Línea 432:** self.imagen\_label.image = img\_tk

- Se guarda una referencia para evitar que la imagen sea recolectada por el recolector de basura.

**Línea 434:** def toggle\_live(self):

- Define el método que alterna el estado del video en vivo (iniciar/detener).

**Línea 435:** if not self.live\_activado:

- Si el video en vivo está desactivado, se procederá a activarlo.

**Línea 436:** self.cap = cv2.VideoCapture(0)

- Se crea un objeto de captura de video para usar la cámara por defecto (0).

**Línea 437:** if not self.cap.isOpened():

- Verifica si la cámara se pudo abrir correctamente.

**Línea 438:** messagebox.showerror("Error", "No se pudo abrir la cámara.")

- Muestra un mensaje de error si la cámara no está disponible.

**Línea 439:** return

- Sale del método si la cámara no se abrió.

**Línea 440:** self.live\_activado = True

- Cambia el estado de live\_activado a verdadero.

**Línea 441:** self.estado\_label.config(text="  Live activado", foreground="green")

- Actualiza el texto del estado para indicar que el live está activo.

**Línea 442:** self.log\_evento("  Live activado")

- Agrega una entrada al log registrando que el video en vivo fue activado.

**Línea 443:** self.hilo\_live = threading.Thread(target=self.mostrar\_video, daemon=True)

- Crea un hilo nuevo que ejecutará la función mostrar\_video.

**Línea 444:** self.hilo\_live.start()

- Inicia el hilo de video en vivo.

**Línea 445:** self.boton\_live.config(text="  Detener Live")

- Cambia el texto del botón para reflejar que ahora se puede detener el live.

**Línea 446:** else:

- Si ya estaba activado, se procederá a desactivarlo.

**Línea 447:** self.live\_activado = False

- Cambia el estado a falso para detener el live.

**Línea 448:** if self.cap:

- Verifica que exista un objeto de captura.

**Línea 449:** self.cap.release()

- Libera la cámara.

**Línea 450:** self.cap = None

- Elimina la referencia al objeto de cámara.

**Línea 451:** self.estado\_label.config(text=" || Live detenido", foreground="red")

- Actualiza el estado para indicar que el live ha sido detenido.

**Línea 452:** self.log\_evento(" || Live detenido")

- Agrega una entrada en el log indicando que se detuvo el video en vivo.

**Línea 453:** self.boton\_live.config(text="► Live")

- Cambia el texto del botón para mostrar que se puede volver a activar.

**Línea 454:** self.boton\_guardar.config(state='disabled')

- Desactiva el botón de guardar imagen cuando el live está apagado.

**Línea 456:** def analizar\_imagen(self):

- Define el método que analiza una imagen previamente capturada con YOLO.

```
C:\> Users > ASUS > OneDrive > Escritorio > Seguridad (renovado).py > SeguridadApp > analizar_imagen
121 class SeguridadApp:
122     def analizar_imagen(self):
123         if not self.imagen_capturada_path:
124             messagebox.showinfo("Info", "No hay imagen capturada para analizar.")
125             self.log_evento("X No hay imagen capturada para analizar.")
126             self.estado_label.config(text="X No hay imagen para analizar.", foreground="red")
127             return
128
129         if self.live_activado:
130             self.live_activado = False
131             if self.cap:
132                 self.cap.release()
133                 self.cap = None
134             self.estado_label.config(text="|| Live detenido para analizar imagen.", foreground="blue")
135             self.log_evento("|| Live detenido para analizar imagen.")
136             time.sleep(0.1)
137
138         try:
139             frame = cv2.imread(self.imagen_capturada_path)
140             if frame is None:
141                 raise Exception("No se pudo cargar la imagen capturada.")
142
143             results = self.model(frame)
144             annotated_frame = results[0].plot()
145             annotated_frame = cv2.cvtColor(annotated_frame, cv2.COLOR_BGR2RGB)
146             annotated_frame = cv2.resize(annotated_frame, (800, 500))
147             img = Image.fromarray(annotated_frame)
148             img_tk = ImageTk.PhotoImage(img)
149             self.imagen_label.config(image=img_tk)
150             self.imagen_label.image = img_tk
151
152             self.ultimos_objetos_detectados = [self.model.names[int(cls)] for result in results for cls in result.bboxes.cls]
153             if self.ultimos_objetos_detectados:
154                 self.log_evento(f"🕒 Objetos detectados: {', '.join(self.ultimos_objetos_detectados)}")
155                 self.estado_label.config(text="🕒 Análisis completado: Objetos detectados.", foreground="blue")
156             else:
157                 self.log_evento("🕒 No se detectaron objetos.")
```

Lín. 491, col. 63 Espacios:4 UTF-8 CRLF {} Python 3.13.2

**Líneas 457-491:**

**Línea 457:** if not self.imagen\_capturada\_path:

- Verifica si existe una imagen previamente capturada. Si no hay, se detiene el análisis.

**Línea 458:** messagebox.showinfo("Info", "No hay imagen capturada para analizar.")

- Muestra una ventana informativa indicando que no hay imagen disponible.

**Línea 459:** self.log\_evento("✖ No hay imagen capturada para analizar.")

- Registra el evento en el log, marcándolo como un error.

**Línea 460:** self.estado\_label.config(text="✖ No hay imagen para analizar.", foreground="red")

- Cambia la etiqueta de estado para indicar que no hay imagen que analizar.

**Línea 461:** return

- Termina la ejecución del método si no hay imagen capturada.

**Línea 463:** if self.live\_activado:

- Si el video en vivo está activado, se detiene para evitar conflicto durante el análisis.

**Línea 464:** self.live\_activado = False

- Cambia el estado a False para detener el live.

**Línea 465:** if self.cap:

- Verifica que el objeto de captura de cámara exista.

**Línea 466:** self.cap.release()

- Libera los recursos de la cámara.

**Línea 467:** self.cap = None

- Elimina la referencia al objeto de captura.

**Línea 468:** self.estado\_label.config(text="⏸ Live detenido para analizar imagen.", foreground="blue")

- Actualiza el texto de estado indicando que el live fue detenido para análisis.

**Línea 469:** self.log\_evento("⏸ Live detenido para analizar imagen.")

- Agrega el evento al log.

**Línea 470:** time.sleep(0.1)

- Espera una fracción de segundo para asegurar la liberación de la cámara.

**Línea 472:** try:

- Inicia un bloque try para capturar posibles errores durante el análisis.

**Línea 473:** frame = cv2.imread(self.imagen\_capturada\_path)

- Carga la imagen capturada desde el disco usando OpenCV.

**Línea 474:** if frame is None:

- Verifica si la imagen no se cargó correctamente.

**Línea 475:** raise Exception("No se pudo cargar la imagen capturada.")

- Lanza una excepción si la imagen no existe o no pudo leerse.

**Línea 477:** results = self.model(frame)

- Ejecuta el modelo YOLO sobre la imagen cargada para obtener predicciones.

**Línea 478:** annotated\_frame = results[0].plot()

- Dibuja las cajas y etiquetas sobre la imagen con los resultados de detección.

**Línea 479:** annotated\_frame = cv2.cvtColor(annotated\_frame, cv2.COLOR\_BGR2RGB)

- Convierte la imagen de BGR (formato OpenCV) a RGB para su visualización en Tkinter.

**Línea 480:** annotated\_frame = cv2.resize(annotated\_frame, (800, 500))

- Cambia el tamaño de la imagen anotada para adaptarla al espacio de la interfaz.

**Línea 481:** img = Image.fromarray(annotated\_frame)

- Convierte la imagen en un objeto Image de PIL para poder manipularla visualmente.

**Línea 482:** img\_tk = ImageTk.PhotoImage(img)

- Convierte la imagen PIL en un formato compatible con Tkinter (PhotoImage).

**Línea 483:** self.imagen\_label.config(image=img\_tk)

- Establece la imagen procesada como contenido del Label.

**Línea 484:** self.imagen\_label.image = img\_tk

- Guarda una referencia a la imagen para evitar que sea recolectada por el recolector de basura.

**Línea 486:** self.ultimos\_objetos\_detectados = [self.model.names[int(cls)]] for result in results for cls in result.boxes.cls]

- Extrae los nombres de los objetos detectados y los guarda en la lista correspondiente.

**Línea 487:** if self.ultimos\_objetos\_detectados:

- Verifica si se han detectado objetos en la imagen.

**Línea 488:** self.log\_evento(f"🔍 Objetos detectados: {', '.join(self.ultimos\_objetos\_detectados)}")

- Registra los objetos encontrados en el log.

**Línea 489:** self.estado\_label.config(text="🔍 Análisis completado: Objetos detectados.", foreground="blue")

- Actualiza la etiqueta de estado indicando éxito en el análisis con detección.

**Línea 490:** else:

- Si no se detectaron objetos, se ejecuta la alternativa.

**Línea 491:** self.log\_evento("🔍 No se detectaron objetos.")

- Registra en el log que no hubo detección de objetos.

```
C:\Users>ASUS>OneDrive>Escritorio>Seguridad (renovado).py>SeguridadApp>analizar_imagen
121 class SeguridadApp:
122     def analizar_imagen(self):
123         self.log_evento("🔍 No se detectaron objetos.")
124         self.estado_label.config(text="🔍 Análisis completado: No se detectaron objetos.", foreground="blue")
125
126     except Exception as e:
127         messagebox.showerror("Error", f"No se pudo analizar la imagen: {e}")
128         self.log_evento(f"🔴 Error analizando imagen: {e}")
129         self.estado_label.config(text="🔴 Error analizando imagen.", foreground="red")
130
131     def iniciar_app(self):
132         self.root.mainloop()
133
134     def cerrar_app(self):
135         if self.live_activado and self.cap:
136             self.cap.release()
137         self.root.destroy()
138
139 if __name__ == "__main__":
140     login = LoginApp()
141     login.run()
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
```

**Líneas 492-509:**

**Línea 492:** self.estado\_label.config(text="🔍 Análisis completado: No se detectaron objetos.", foreground="blue")

- Actualiza el estado en pantalla para informar que el análisis concluyó sin encontrar objetos.

**Línea 494:** except Exception as e:

- Captura cualquier error que ocurra durante el análisis de la imagen.

**Línea 495:** messagebox.showerror("Error", f"No se pudo analizar la imagen: {e}")

- Muestra una ventana emergente indicando el error encontrado durante el análisis.

**Línea 496:** self.log\_evento(f"🔴 Error analizando imagen: {e}")

- Registra el error en el historial de eventos de la aplicación.

**Línea 497:** self.estado\_label.config(text="✖ Error analizando imagen.", foreground="red")

- Actualiza el estado visible en la GUI para reflejar que ocurrió un fallo en el análisis.

**Línea 499:** def iniciar\_app(self):

- Define el método para iniciar la interfaz principal del sistema de seguridad.

**Línea 500:** self.root.mainloop()

- Inicia el bucle principal de eventos de la ventana Tkinter (self.root).

**Línea 502:** def cerrar\_app(self):

- Define el método para cerrar la aplicación de forma segura.

**Línea 503:** if self.live\_activado and self.cap:

- Verifica si el modo en vivo está activo y hay una cámara abierta.

**Línea 504:** self.cap.release()

- Libera la cámara si está en uso, evitando que quede bloqueada.

**Línea 505:** self.root.destroy()

- Cierra completamente la ventana principal del sistema de seguridad.

**Línea 507:** if \_\_name\_\_ == "\_\_main\_\_":

- Verifica si el script se está ejecutando directamente (y no como módulo importado).

**Línea 508:** login = LoginApp()

- Crea una instancia de la clase LoginApp, que muestra la interfaz de inicio de sesión.

**Línea 509:** login.run()

- Ejecuta el método run() para lanzar el ciclo de eventos de la ventana de login.

# Explicación del código de la placa Raspberry en Python:

A continuación, se explicara el código empleado en Python para la placa Raspberry Pi Pico W, haciendo uso de conjuntos de lineas para una mejor vizualización. Las lineas en blanco han sido omitidas.

```
C:\> Users > ASUS > OneDrive > Escritorio > main.py
 1 # En Raspberry Pi: Transmisión MJPEG por Flask
 2 from flask import Flask, Response
 3 from picamera2 import Picamera2
 4 import cv2, time
 5
 6 app = Flask(__name__)
 7 picam2 = Picamera2()
 8 config = picam2.create_video_configuration(main={"format": "YUV420", "size": (640, 360)})
 9 picam2.configure(config)
10 picam2.start()
11 time.sleep(1)
12
13 def generate():
14     while True:
15         frame = picam2.capture_array("main")
16         frame = cv2.cvtColor(frame, cv2.COLOR_YUV2BGR_I420)
17         _, jpeg = cv2.imencode('.jpg', frame)
18         yield (b'--frame\r\nContent-Type: image/jpeg\r\n\r\n' + jpeg.tobytes() + b'\r\n')
19
20 @app.route('/video')
21 def video_feed():
22     return Response(generate(), mimetype='multipart/x-mixed-replace; boundary=frame')
23
24 app.run(host="0.0.0.0", port=8080)
25
```

Líneas 1-24:

**Línea 1:** # En Raspberry Pi: Transmisión MJPEG por Flask

- Comentario descriptivo: indica que el script transmite video MJPEG usando Flask en una Raspberry Pi.

**Línea 2:** from flask import Flask, Response

- Flask: Microframework web de Python para crear servidores HTTP.
- Response: Clase de Flask que permite construir respuestas HTTP personalizadas, como la de video en vivo.

**Línea 3:** from picamera2 import Picamera2

- Picamera2: Módulo para controlar la cámara de la Raspberry Pi de forma avanzada.
- Picamera2: Clase principal para capturar imágenes o video.

**Línea 4:** import cv2, time

- cv2: Módulo de OpenCV para procesamiento de imágenes.
- time: Módulo estándar de Python para trabajar con temporizadores o pausas.

**Línea 6:** app = Flask(\_\_name\_\_)

- Crea una instancia de la aplicación Flask.

- `__name__` se usa para determinar si el script se está ejecutando directamente o es importado.

**Línea 7:** `picam2 = Picamera2()`

- Crea una instancia del controlador de la cámara Raspberry Pi usando la clase `Picamera2`.

**Línea 8:** `config = picam2.create_video_configuration(main={"format": "YUV420", "size": (640, 360)})`

- Configura la resolución y el formato de salida del video en el modo principal.
- YUV420: Formato de color eficiente en ancho de banda.
- (640, 360): Resolución del video.

**Línea 9:** `picam2.configure(config)`

- Aplica la configuración previamente creada a la cámara.

**Línea 10:** `picam2.start()`

- Inicia la cámara con la configuración actual.

**Línea 11:** `time.sleep(1)`

- Espera 1 segundo para asegurarse de que la cámara esté completamente inicializada antes de capturar imágenes.

**Línea 13:** `def generate():`

- Define una función generadora que produce un flujo continuo de imágenes JPEG para transmitir como MJPEG.

**Línea 14:** `while True:`

- Bucle infinito que permite transmitir video en tiempo real.

**Línea 15:** `frame = picam2.capture_array("main")`

- Captura un cuadro de video desde el stream principal de la cámara en formato YUV.

**Línea 16:** `frame = cv2.cvtColor(frame, cv2.COLOR_YUV2BGR_I420)`

- Convierte el cuadro capturado de YUV a BGR, que es el formato estándar en OpenCV.

**Línea 17:** `_, jpeg = cv2.imencode('.jpg', frame)`

- Codifica la imagen BGR en formato JPEG, que será enviado como parte del stream MJPEG.

**Línea 18:** `yield (b'--frame\r\nContent-Type: image/jpeg\r\n\r\n' + jpeg.tobytes() + b'\r\n')`

- `yield`: Retorna cada imagen codificada como parte de un stream HTTP multipart para clientes web.

**Línea 20:** `@app.route('/video')`

- Define una ruta en Flask (`/video`) que se activará cuando se acceda a esa URL.

**Línea 21:** `def video_feed():`

- Función asociada a la ruta `/video`, que será llamada por Flask para manejar la solicitud.

**Línea 22:** `return Response(generate(), mimetype='multipart/x-mixed-replace; boundary=frame')`

- Devuelve una respuesta HTTP con el stream MJPEG generado por la función `generate()`.

**Línea 24:** `app.run(host="0.0.0.0", port=8080)`

- Inicia el servidor Flask accesible desde cualquier IP (`0.0.0.0`) en el puerto 8080. Ideal para acceso en red local.

# REQUISITOS:

## 1. Requisitos de Hardware

Para el correcto funcionamiento del sistema de seguridad inteligente basado en detección visual remota con inteligencia artificial, se requiere el siguiente equipamiento mínimo:

- Raspberry Pi 5 de 8 GB de RAM, que actuará como nodo de captura y transmisión de video.
- Módulo de cámara compatible, preferentemente la Raspberry Pi Camera Module 5, con interfaz CSI para una mayor fluidez y compatibilidad.
- Tarjeta microSD de 32 GB, clase 10, con sistema operativo Raspberry Pi OS.
- Fuente de alimentación oficial para Raspberry Pi (5.1V/3A USB-C).
- MacBook o Laptop con sistema operativo macOS, Windows o Linux, que actuará como estación cliente para la visualización, análisis y gestión del sistema de seguridad.
- Conexión a Internet estable en ambos extremos (Raspberry y cliente), requerida para la transmisión del video mediante el servicio ngrok.

## 2. Requisitos de Software

A nivel de software, el sistema requiere la instalación y configuración de los siguientes componentes:

- Python 3.8 o superior, instalado tanto en la Raspberry Pi como en la estación cliente.
- Librerías de Python necesarias:
  - opencv-python
  - ultralytics (para YOLOv8)
  - Pillow
  - tkinter (incluido por defecto en la mayoría de instalaciones de Python)
  - twilio (para envío de alertas SMS)
  - python-dotenv (para manejo de credenciales)
- Ngrok para la exposición segura del stream de video capturado por la Raspberry Pi a través de un túnel HTTP público.
- Archivo .env configurado adecuadamente con las credenciales necesarias para:
  - Acceso SMTP del correo electrónico de envío de alertas
  - Credenciales de Twilio (SID, token, número)
- Archivo de configuración ngrok.yml, debidamente autenticado con el token de la cuenta y apuntando al puerto de transmisión del servidor de video.

- Modelo YOLOv8 (yolov8n.pt) descargado y disponible localmente.

# Uso del Sistema

El sistema de seguridad inteligente consta de dos componentes principales: un servidor de captura y transmisión de video alojado en una Raspberry Pi, y una interfaz gráfica de monitoreo y análisis que se ejecuta en una computadora personal (cliente). A continuación, se describen los pasos de uso para la correcta operación del sistema.

## 1. Inicio del servidor de video (Raspberry Pi)

La Raspberry Pi utiliza el módulo Picamera2 y la biblioteca Flask para capturar imágenes en tiempo real desde su cámara integrada y transmitirlas como un flujo MJPEG a través de la red. El archivo principal main.py contiene el siguiente proceso:

- Se inicializa la cámara usando Picamera2.
- Se configura la salida de video en formato YUV420 a una resolución de 640x360 píxeles.
- Se lanza un servidor Flask local que expone una ruta /video, donde se emite continuamente el flujo de imágenes capturadas, convertido al formato JPEG.
- El servidor queda escuchando en el puerto 8080, permitiendo el acceso desde cualquier dispositivo conectado a la red, o desde el exterior usando herramientas como ngrok.

## 2. Acceso al flujo de video desde el cliente

En la computadora del usuario (MacBook, PC u otro dispositivo), se ejecuta una interfaz gráfica basada en Tkinter, la cual accede al flujo de video emitido por la Raspberry Pi mediante su URL pública (obtenida con ngrok o una IP fija).

- El usuario inicia sesión en la interfaz.
- Se activa el monitoreo en vivo mediante el botón “Live”, que carga el stream remoto usando cv2.VideoCapture(URL).
- Se aplica detección de objetos en tiempo real mediante un modelo YOLOv8 cargado localmente. En este caso, se identifican personas y armas (knife, gun), clasificando visualmente a los individuos como:
  - Persona Agresiva: Si se encuentra cercana a un arma.
  - Víctima: Si está próxima a una persona agresiva pero no porta armas.
  - Persona: Si no hay amenazas en la escena.

Este procesamiento se muestra sobre la interfaz gráfica, junto a funciones adicionales como:

- Captura de imágenes.
- Análisis posterior de imágenes capturadas.
- Envío de alertas por sonido, SMS (Twilio) y correo electrónico (SMTP) ante la detección de situaciones potencialmente peligrosas.

### 3. Flujo resumido

- I. La Raspberry transmite video en vivo mediante /video.
- II. La PC del cliente accede a dicha URL.
- III. Se detectan objetos con YOLOv8 en tiempo real.
- IV. Se visualiza el estado de cada persona en pantalla.
- V. Si se detectan amenazas, se activa el sistema de alerta múltiple.

## Configuración de Transmisión con Ngrok

Para permitir el acceso remoto al flujo de video generado por la Raspberry Pi, incluso desde redes externas o dinámicas, se ha integrado Ngrok como solución de túnel seguro. Ngrok permite exponer el puerto local donde se ejecuta el servidor Flask a una URL pública accesible desde cualquier lugar del mundo, sin necesidad de configurar el router ni abrir puertos manualmente.

### 1. Requisitos Previos

Antes de proceder con la configuración, asegúrese de que:

- Tiene una cuenta activa en Ngrok.
- Ha instalado ngrok en la Raspberry Pi. Si no lo ha hecho, puede instalarlo ejecutando:

```
wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-arm.zip  
unzip ngrok-stable-linux-arm.zip  
sudo mv ngrok /usr/local/bin
```

### 2. Autenticación con su Cuenta Ngrok

Una vez instalado, es necesario vincular ngrok con su cuenta para habilitar túneles personalizados y persistentes. Inicie sesión en su cuenta de Ngrok y copie el Auth Token que aparece en el panel de control. Luego, configúrelo en la Raspberry ejecutando:

```
ngrok config add-authtoken TOKEN
```

Esto almacenará su token de autenticación en `~/.ngrok2/ngrok.yml`.

### 3. Creación del Túnel para el Servidor Flask

El servidor Flask que ejecuta `main.py` se lanza por defecto en el puerto 8080. Para exponer este puerto, basta con ejecutar:

```
ngrok http 8080
```

Ngrok generará una URL pública como:

```
https://raspberrybowser.ngrok.app
```

### 4. Automatización al Iniciar la Raspberry Pi

Para facilitar la operación sin intervención manual, se recomienda automatizar el lanzamiento tanto de ngrok como del script `main.py` al iniciar la Raspberry Pi. Esto puede lograrse añadiendo las siguientes líneas al archivo `~/.bashrc`, `~/.profile` o como servicio `systemd`:

```
# Inicia el servidor Flask
python3 /home/pi/proyecto/main.py &

# Espera unos segundos y luego lanza Ngrok
sleep 5
ngrok http 8080 --log=stdout > /home/pi/ngrok.log &
```

También puede usarse un archivo `ngrok.yml` personalizado en `~/.ngrok2/` para definir dominios reservados o configuraciones más avanzadas.

### 5. Integración con la Interfaz Cliente

Una vez lanzado ngrok, la URL proporcionada debe ser utilizada como origen del flujo de video en la interfaz gráfica del cliente, por ejemplo:

```
url = 'https://raspberrybowser.ngrok.app/video'
```

```
cap = cv2.VideoCapture(url)
```

Esta URL puede cambiar cada vez que se reinicia ngrok, a menos que utilice una cuenta Ngrok Pro con subdominios personalizados. Para entornos de producción, se recomienda usar un subdominio fijo con el siguiente comando:

```
ngrok http --domain=raspberrybowser.ngrok.app 8080
```

Esto requiere tener configurado dicho subdominio previamente en la cuenta de Ngrok.

## Referencias bibliograficas:

Alpha2lucifer. (s.f.). *sentinalsecure*. GitHub.

<https://github.com/alpha2lucifer/sentinalsecure>

Alexisvalentino. (s.f.). *Threat-Detection-System*. GitHub.

<https://github.com/alexisvalentino/Threat-Detection-System>

Ciscoblockchain. (s.f.). *YOLO-Pi*. GitHub. <https://github.com/CiscoBlockChain/YOLO-Pi>

Shruthika-s. (s.f.). *VisionGuard: Anomaly Detection in CCTV footage*. GitHub.

<http://github.com/Shruthika-s/VisionGuard-Anomaly-Detection-in-CCTV-footage>

Yigitekin. (s.f.). *Human Detection using YOLOv5 and Raspberry Pi*. GitHub.

<https://github.com/YigitEkin/Human-Detection-using-YOLOv5-and-Raspberry-Pi>