Explicación Detallada del Código

Importaciones y Configuración

```
import asyncio
from fastapi import FastAPI, Response
from fastapi.responses import StreamingResponse, JSONResponse
import cv2
import threading
from typing import AsyncGenerator, Optional, Union
from contextlib import asynccontextmanager
import uvicorn
```

Esta sección importa las librerías necesarias: - `asyncio` para programación asincrónica. - `FastAPI` para crear el servidor web. - `cv2` (OpenCV) para captura y procesamiento de video. - `threading` para acceso seguro a la cámara desde múltiples hilos. - Tipos opcionales (`Optional`, `Union`) y generadores asincrónicos (`AsyncGenerator`) para anotaciones de tipo. - `uvicorn` para ejecutar el servidor ASGI.

Context Manager de Duración de la Aplicación

```
@asynccontextmanager
async def duracion_app(app: FastAPI):
    try:
        yield
    except asyncio.exceptions.CancelledError as error:
        print(error.args)
    finally:
        if camara:
            camara.liberar()
            print("Recurso de cámara liberado.")

app = FastAPI(lifespan=duracion_app)
```

Este context manager asegura la correcta gestión de recursos: - Maneja los eventos de inicio y cierre de la aplicación FastAPI. - Al cerrar la aplicación, libera la cámara para evitar fugas de recursos.

Clase Camara

```
class Camara:
    def __init__(self, fuente: Optional[Union[str, int]] = 0) -> None:
        self.cap = cv2.VideoCapture(fuente)
        self.lock = threading.Lock()

    def obtener_frame(self) -> bytes:
```

```
        with self.lock:
            ret, frame = self.cap.read()
            if not ret:
                return b''
            ret, jpeg = cv2.imencode('.jpg', frame)
            if not ret:
                return b''
            return jpeg.tobytes()

    def liberar(self) -> None:
        with self.lock:
            if self.cap.isOpened():
                self.cap.release()
```

La clase `Camara` encapsula la funcionalidad de captura de video: - Inicializa un objeto de captura de video usando un índice de cámara o URL RTSP. - `obtener_frame()` captura un frame, lo convierte a JPEG y devuelve los bytes. - `liberar()` libera de manera segura el recurso de la cámara. - El uso de locks garantiza acceso seguro a la cámara desde múltiples hilos.

Generador de Frames

```
async def generar_frames() -> AsyncGenerator[bytes, None]:
    try:
        while True:
            frame = camara.obtener_frame()
            if frame:
                yield (b'--frame\r\n'
                       b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
            else:
                await asyncio.sleep(0.1)
            await asyncio.sleep(0)
    except (asyncio.CancelledError, GeneratorExit):
        print("Generación de frames cancelada.")
    finally:
        print("Generador de frames finalizado.")
```

Este generador asincrónico: - Produce continuamente frames de la cámara. - Formatea cada frame como parte de una respuesta HTTP multipart. - Maneja la cancelación y limpieza de recursos correctamente.

Endpoints de la API

```
@app.get("/")
async def raiz():
    return JSONResponse({"mensaje": "Servidor de video funcionando. Accede
a /video para ver el streaming."})

@app.get("/video")
```

```
async def transmision_video() -> StreamingResponse:
    return StreamingResponse(
        generar_frames(),
        media_type='multipart/x-mixed-replace; boundary=frame'
    )

@app.get("/snapshot")
async def obtener_snapshot() -> Response:
    frame = camara.obtener_frame()
    if frame:
        return Response(content=frame, media_type="image/jpeg")
    else:
        return Response(status_code=404,
content="No se pudo obtener el frame de la cámara.")
```

Se definen tres endpoints: - `/`: Endpoint raíz para verificar que el servidor funciona. - `/video`: Transmite video continuo mediante `StreamingResponse`. - `/snapshot`: Devuelve un solo frame en formato JPEG.

Función Principal y Arranque del Servidor

```
async def ejecutar_servidor():
    config = uvicorn.Config(app, host='192.168.18.26', port=8007,
log_level="info")
    servidor = uvicorn.Server(config)
    await servidor.serve()

if __name__ == '__main__':
    camara = Camara('rtsp://admin:TIss9831@192.168.18.14:554/')
    print("Servidor iniciado. Abre tu navegador en http://localhost:8007 o
http://<tu_IP>:8007")
    try:
        asyncio.run(ejecutar_servidor())
    except KeyboardInterrupt:
        print("Servidor detenido por el usuario.")
```

La función principal: - Configura y arranca el servidor Uvicorn. - Inicializa la cámara con la URL RTSP especificada. - Ejecuta el servidor de manera asincrónica y maneja interrupciones por teclado para un cierre seguro.