Codigo mio

--init--.py : “” No tiene nada, no se si hay que rellenarlo con algo, si es asi, facilitame el codigo para añadirlo

config.py: “import os

```
# Rutas de las carpetas
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
IMAGE_DIR = os.path.join(BASE_DIR, 'images')
INPUT_DIR = os.path.join(IMAGE_DIR, 'input_images')
PROCESSING_DIR = os.path.join(IMAGE_DIR, 'processing_images')
PROCESSED_DIR = os.path.join(IMAGE_DIR, 'processed_images')
INFRACTIONS_DIR = os.path.join(IMAGE_DIR, 'infracciones_detectadas')
ERROR_DIR = os.path.join(IMAGE_DIR, 'error_images')

# Configuración del monitoreo de imágenes
MONITOR_INTERVAL_SECONDS = 5 # Intervalo de tiempo para revisar nuevas imágenes

# Configuración de la API de Gemini (Google AI)
GEMINI_API_KEY = "AIzaSyDBHci8NClFXNmfiNLArckpsoyRU9TGW4w" # API key proporcionada
GEMINI_MODEL_NAME = "gemini-1.5-flash"

# Umbrales de confianza para la detección de infracciones
CONFIDENCE_THRESHOLD_VIOLATION = 0.7 # Umbral mínimo de confianza para considerar una infracción
CONFIDENCE_THRESHOLD_PLATE = 0.5     # Umbral mínimo de confianza para detectar una patente (opcional, si el modelo lo permite)

# --- Configuración de MongoDB ---
MONGO_URI = "mongodb://localhost:27017/"  # URI de conexión a tu MongoDB local
MONGO_DB_NAME = "parking_violations_db"   # Nombre de la base de datos
MONGO_COLLECTION_NAME = "infractions"     # Nombre de la colección
```

”

data_logger.py : “

```
from pymongo import MongoClient
from pymongo.errors import ConnectionFailure, PyMongoError
from datetime import datetime
from src.config import MONGO_URI, MONGO_DB_NAME, MONGO_COLLECTION_NAME
import os

class DataLogger:
    def __init__(self):
        self.client = None
        self.db = None
        self.collection = None
        self._connect_to_mongodb()

    def _connect_to_mongodb(self):
        """Intenta conectar a la base de datos MongoDB."""
        try:
```

```
            self.client = MongoClient(MONGO_URI)
            # La siguiente línea forzará la conexión para verificarla
            self.client.admin.command('ping')
            self.db = self.client[MONGO_DB_NAME]
            self.collection = self.db[MONGO_COLLECTION_NAME]
            print(f"Conexión a MongoDB establecida: {MONGO_URI} | Base de datos:
{MONGO_DB_NAME} | Colección: {MONGO_COLLECTION_NAME}")
        except ConnectionFailure as e:
            print(f"Error de conexión a MongoDB: {e}")
            self.client = None
            self.db = None
            self.collection = None
        except PyMongoError as e:
            print(f"Error general de PyMongo al conectar: {e}")
            self.client = None
            self.db = None
            self.collection = None
        except Exception as e:
            print(f"Error inesperado al conectar a MongoDB: {e}")
            self.client = None
            self.db = None
            self.collection = None

    def log_infraction(self, image_name, infraction_data):
        """
        Registra una infracción en la colección de MongoDB.
        infraction_data debe ser un diccionario con los detalles de la infracción.
        """
        if self.collection is None:
            print("No hay conexión activa a MongoDB. No se pudo registrar la infracción.")
            return False

        # Asegurarse de que infraction_data sea un diccionario
        if not isinstance(infraction_data, dict):
            print(f"Datos de infracción inválidos para {image_name}: {infraction_data}")
            return False

        # Preparar el documento a insertar
        document = {
            "timestamp": datetime.now(),
            "image_name": image_name,
            "infraccion_detectada": infraction_data.get("infraccion_detectada", False),
            "tipo_infraccion": infraction_data.get("tipo_infraccion", "desconocido"),
            "patente": infraction_data.get("patente", "no_visible"),
            "confianza_infraccion": infraction_data.get("confianza_infraccion", 0.0),
            "confianza_patente": infraction_data.get("confianza_patente", 0.0)
        }

        try:
            result = self.collection.insert_one(document)
            print(f"Infracción para {image_name} registrada en MongoDB con ID: {result.inserted_id}")
            return True
        except PyMongoError as e:
```

```
            print(f"Error al insertar la infracción para {image_name} en MongoDB: {e}")
            return False
        except Exception as e:
            print(f"Error inesperado al registrar la infracción para {image_name}: {e}")
            return False

    def close_connection(self):
        """Cierra la conexión con MongoDB."""
        if self.client:
            self.client.close()
            print("Conexión a MongoDB cerrada.")
```

”

gemini_analyzer.py : “

```
import google.generativeai as genai
import os
import json
from PIL import Image
from src.config import GEMINI_API_KEY, GEMINI_MODEL_NAME,
CONFIDENCE_THRESHOLD_VIOLATION, CONFIDENCE_THRESHOLD_PLATE

class GeminiAnalyzer:
    def __init__(self):
        if not GEMINI_API_KEY or GEMINI_API_KEY == "TU_API_KEY_DE_GEMINI":
            raise ValueError("La clave de API de Gemini no está configurada. Por favor, edita
src/config.py.")
        genai.configure(api_key=GEMINI_API_KEY)
        self.model = genai.GenerativeModel(GEMINI_MODEL_NAME)

    def _generate_prompt(self):
        """Genera el prompt para la detección de infracciones."""
        return """
        Analiza esta imagen para detectar vehículos mal estacionados en la vía pública o en lugares no
permitidos (como aceras, pasos peatonales, doble fila).
        Identifica si hay vehículos y, si es así, determina si están cometiendo alguna infracción de
estacionamiento.
        Si detectas una infracción, especifica el tipo de infracción (ej. "doble fila", "sobre acera", "paso
peatonal", "bloqueando entrada", "estacionado en zona prohibida").
        Además, intenta identificar la patente (placa) del vehículo infractor. Si no se puede leer o no es
visible, indica "no_visible".
        Para cada vehículo detectado en la imagen, proporciona la siguiente información en formato
JSON.
        Si no hay vehículos o no hay infracciones, la salida JSON debe ser un array vacío.

        Ejemplo de formato JSON esperado para cada infracción detectada:
        {
          "infraccion_detectada": true,
          "tipo_infraccion": "tipo_de_infraccion",
          "patente": "ABC123",
          "confianza_infraccion": 0.95,
          "confianza_patente": 0.8
        }
```

```
        Si no hay infracción, la respuesta debería ser:
        []

        Considera que la ubicación general es una ciudad en Chile (como Temuco) y las regulaciones de estacionamiento chilenas.
        """

    def analyze_image(self, image_path):
        """
        Analiza una imagen usando la API de Gemini para detectar infracciones de estacionamiento.
        Retorna una lista de diccionarios con los resultados de la detección,
        o una lista vacía si no hay infracciones o si el formato es inválido.
        """
        try:
            img = Image.open(image_path).convert('RGB')
            prompt = self._generate_prompt()
            response = self.model.generate_content([prompt, img])

            # Acceder al texto generado
            text_response = response.text.strip()

            # Intentar parsear la respuesta JSON
            try:
                # Limpiar la respuesta para asegurar que sea un JSON válido
                # Gemini a veces incluye el bloque de código, ej. ```json ... ```
                if text_response.startswith("```json"):
                    text_response = text_response.replace("```json", "").replace("```", "").strip()

                data = json.loads(text_response) # Usar json.loads para parsear JSON de forma segura

                if not isinstance(data, list):
                    print(f"[{os.path.basename(image_path)}] Respuesta inesperada del modelo (no es una lista): {data}")
                    return [] # Retornar lista vacía si no es una lista

                # Filtrar infracciones por umbral de confianza y agregar timestamp
                processed_results = []
                for item in data:
                    if isinstance(item, dict) and item.get("infraccion_detectada", False):
                        confianza = item.get("confianza_infraccion", 0.0)
                        if confianza >= CONFIDENCE_THRESHOLD_VIOLATION:
                            # Asegúrate de que los campos existan o asigna valores por defecto
                            processed_results.append({
                                "infraccion_detectada": True,
                                "tipo_infraccion": item.get("tipo_infraccion", "desconocido"),
                                "patente": item.get("patente", "no_visible"),
                                "confianza_infraccion": float(confianza),
                                "confianza_patente": float(item.get("confianza_patente", 0.0))
                            })
                        else:
                            print(f"[{os.path.basename(image_path)}] Infracción detectada pero con confianza ({confianza}) bajo el umbral.")
                    else:
```

```
                # Si no es una infracción o no cumple el formato esperado
                print(f"[{os.path.basename(image_path)}] Elemento no válido en la respuesta del modelo: {item}")
            return processed_results

        except (SyntaxError, ValueError) as json_error:
            print(f"[{os.path.basename(image_path)}] No se pudo parsear la respuesta JSON de Gemini: {json_error}")
            print(f"Respuesta cruda de Gemini: {text_response}")
            return [] # Retornar lista vacía si el JSON es inválido

    except Exception as e:
        print(f"Error al analizar la imagen {image_path} con Gemini: {e}")
        return []


”
```

image_manager.py : “

```
import os
import shutil
from src.config import INPUT_DIR, PROCESSING_DIR, PROCESSED_DIR, INFRACTIONS_DIR, ERROR_DIR

class ImageManager:
    def __init__(self):
        self._ensure_directories_exist()

    def _ensure_directories_exist(self):
        """Asegura que todos los directorios necesarios existan."""
        for d in [INPUT_DIR, PROCESSING_DIR, PROCESSED_DIR, INFRACTIONS_DIR, ERROR_DIR]:
            os.makedirs(d, exist_ok=True)

    def get_new_images(self):
        """Obtiene una lista de imágenes en el directorio de entrada."""
        return [f for f in os.listdir(INPUT_DIR) if os.path.isfile(os.path.join(INPUT_DIR, f))]

    def move_to_processing(self, image_name):
        """Mueve una imagen del directorio de entrada al de procesamiento."""
        src_path = os.path.join(INPUT_DIR, image_name)
        dest_path = os.path.join(PROCESSING_DIR, image_name)
        try:
            shutil.move(src_path, dest_path)
            print(f"[{image_name}] Movida a procesamiento.")
            return dest_path
        except Exception as e:
            print(f"Error al mover {image_name} a procesamiento: {e}")
            return None

    def move_to_processed(self, image_name):
        """Mueve una imagen del directorio de procesamiento al de procesados."""
        src_path = os.path.join(PROCESSING_DIR, image_name)
        dest_path = os.path.join(PROCESSED_DIR, image_name)
        try:
```

```
            shutil.move(src_path, dest_path)
            print(f"[{image_name}] Movida a procesados.")
        except Exception as e:
            print(f"Error al mover {image_name} a procesados: {e}")

    def move_to_infractions(self, image_name):
        """Mueve una imagen del directorio de procesamiento al de infracciones."""
        src_path = os.path.join(PROCESSING_DIR, image_name)
        dest_path = os.path.join(INFRACTIONS_DIR, image_name)
        try:
            shutil.move(src_path, dest_path)
            print(f"[{image_name}] ¡Infracción detectada! Movida a infracciones.")
        except Exception as e:
            print(f"Error al mover {image_name} a infracciones: {e}")

    def move_to_error(self, image_name):
        """Mueve una imagen del directorio de procesamiento al de errores."""
        src_path = os.path.join(PROCESSING_DIR, image_name)
        dest_path = os.path.join(ERROR_DIR, image_name)
        try:
            shutil.move(src_path, dest_path)
            print(f"[{image_name}] Movida a errores (fallo en procesamiento).")
        except Exception as e:
            print(f"Error al mover {image_name} a errores: {e}")
```

”

main_agent.py: “
```
import time
import os
from src.config import MONITOR_INTERVAL_SECONDS, INPUT_DIR, PROCESSING_DIR,
INFRACTIONS_DIR
from src.image_manager import ImageManager
from src.gemini_analyzer import GeminiAnalyzer
from src.data_logger import DataLogger # Importa DataLogger

def start_monitoring():
    image_manager = ImageManager()
    gemini_analyzer = GeminiAnalyzer()
    data_logger = DataLogger() # Instancia el DataLogger

    print(f"Monitoreando la carpeta de entrada: {INPUT_DIR}")
    print("Presiona Ctrl+C para detener el monitoreo.")

    try:
        while True:
            new_images = image_manager.get_new_images()
            if not new_images:
                # print(f"No se encontraron nuevas imágenes en {INPUT_DIR}. Esperando...")
                time.sleep(MONITOR_INTERVAL_SECONDS)
                continue

            for image_name in new_images:
```

```
            image_path_in_processing = None
            try:
                print(f"[{image_name}] Nueva imagen detectada. Moviendo a procesamiento...")
                image_path_in_processing = image_manager.move_to_processing(image_name)

                if image_path_in_processing:
                    print(f"[{image_name}] Analizando con Gemini...")
                    results = gemini_analyzer.analyze_image(image_path_in_processing)

                    if results:
                        print(f"[{image_name}] Infracción(es) detectada(s):")
                        for infraction in results:
                            print(f"  - Tipo: {infraction.get('tipo_infraccion')}, Patente: {infraction.get('patente')},
Confianza: {infraction.get('confianza_infraccion'):.2f}")
                            data_logger.log_infraction(image_name, infraction) # Registra en MongoDB
                        image_manager.move_to_infractions(image_name)
                    else:
                        print(f"[{image_name}] No se detectaron infracciones o no cumplen el umbral.")
                        image_manager.move_to_processed(image_name)
                else:
                    print(f"[{image_name}] No se pudo mover la imagen a procesamiento. Saltando.")

            except Exception as e:
                print(f"Ocurrió un error inesperado al procesar {image_name}: {e}")
                if image_path_in_processing and os.path.exists(image_path_in_processing):
                    image_manager.move_to_error(image_name)
                else:
                    # Si la imagen ya no está en processing, pudo haberse borrado o movido por otro
proceso.
                    print(f"[{image_name}] No se pudo mover a errores, el archivo no existe o ya fue
movido.")
        time.sleep(MONITOR_INTERVAL_SECONDS) # Esperar antes de revisar de nuevo

    except KeyboardInterrupt:
        print("\nMonitoreo detenido por el usuario.")
    except Exception as e:
        print(f"Ocurrió un error inesperado en el bucle principal:\n{e}")
        # Intenta mover la imagen actual a la carpeta de errores si es posible
        if 'image_name' in locals() and os.path.exists(os.path.join(PROCESSING_DIR, image_name)):
            image_manager.move_to_error(image_name)
        time.sleep(MONITOR_INTERVAL_SECONDS) # Esperar antes de reintentar en caso de error
    finally:
        data_logger.close_connection() # Asegura que la conexión a MongoDB se cierre al finalizar

if __name__ == "__main__":
    start_monitoring()
```

”

Así esta el orden de mis ficheros : “tree -a

```
.
├── images
│   ├── error_images
│   │   ├── 35e981bb-be19-4599-8dcb-f419575c60ef_PLATE-CYTV95_2025-06-04_15-37-22.jpeg
│   │   ├── 43251a0c-12ec-4cc3-92fa-6565f6987752_PLATE-CHWH83_2025-06-04_15-37-19.jpeg
│   │   ├── a4f97de5-e458-498b-bf9d-2ea627e697d4_PLATE-PCDB92_2025-06-04_15-37-26.jpeg
│   │   └── images.jpeg
│   ├── infracciones_detectadas
│   │   ├── 35e981bb-be19-4599-8dcb-f419575c60ef_PLATE-CYTV95_2025-06-04_15-37-22.jpeg
│   │   ├── 43251a0c-12ec-4cc3-92fa-6565f6987752_PLATE-CHWH83_2025-06-04_15-37-19.jpeg
│   │   ├── a4f97de5-e458-498b-bf9d-2ea627e697d4_PLATE-PCDB92_2025-06-04_15-37-26.jpeg
│   │   └── images.jpeg
│   ├── input_images
│   ├── processed_images
│   │   ├── 35e981bb-be19-4599-8dcb-f419575c60ef_PLATE-CYTV95_2025-06-04_15-37-22.jpeg
│   │   ├── 43251a0c-12ec-4cc3-92fa-6565f6987752_PLATE-CHWH83_2025-06-04_15-37-19.jpeg
│   │   ├── a4f97de5-e458-498b-bf9d-2ea627e697d4_PLATE-PCDB92_2025-06-04_15-37-26.jpeg
│   │   ├── donde-puedo-estacionar.jpg
│   │   └── images.jpeg
│   └── processing_images
├── src
│   ├── config.py
│   ├── data_logger.py
│   ├── gemini_analyzer.py
│   ├── image_manager.py
│   ├── __init__.py
│   ├── main_agent.py
│   └── __pycache__
│       ├── config.cpython-312.pyc
│       ├── data_logger.cpython-312.pyc
│       ├── gemini_analyzer.cpython-312.pyc
│       ├── image_manager.cpython-312.pyc
│       └── __init__.cpython-312.pyc
└── venv
    ├── bin
    │   ├── activate
    │   ├── activate.csh
    │   ├── activate.fish
    │   ├── Activate.ps1
    │   ├── normalizer
    │   ├── pip
```

```
│   ├── pip3
│   ├── pip3.12
│   ├── pyrsa-decrypt
│   ├── pyrsa-encrypt
│   ├── pyrsa-keygen
│   ├── pyrsa-priv2pub
│   ├── pyrsa-sign
│   ├── pyrsa-verify
│   ├── python -> python3.12
│   ├── python3 -> python3.12
│   ├── python3.12 -> /usr/bin/python3.12
│   └── tqdm
├── include
│   └── python3.12
├── lib
│   └── python3.12".
```