ChatGPT

Proyecto de Organización Automática de Archivos

Este proyecto de Python contiene varios módulos para escanear carpetas, extraer texto (incluyendo OCR para imágenes y PDF escaneados), detectar archivos duplicados y generar un índice JSON con metadatos. Se utiliza la librería **PyMuPDF** (fitz) junto con **Tesseract OCR** para extraer texto de documentos y imágenes [1]. La detección de duplicados se realiza calculando un hash (SHA-256 por defecto) de cada archivo y comparando esos valores [2]. Todos los scripts tienen comentarios en español para facilitar su comprensión.

A continuación se muestran los archivos del proyecto, cada uno en su propio bloque de código.

file_utils.py

```
# file_utils.py

import os
from datetime import datetime

# Mapa de extensiones a categorías de archivos
CATEGORIAS = {
    "txt": "texto",
    "md": "texto",
    "py": "script",
    "js": "script",
    "html": "script",
    "css": "script",
    "csv": "datos",
    "sql": "datos",
    "json": "datos",
    "xml": "datos",
    "jpg": "imagen",
    "jpeg": "imagen",
    "png": "imagen",
    "gif": "imagen",
    "bmp": "imagen",
    "pdf": "documento",
    "zip": "archivo_comprimido",
    # Agregar más extensiones y categorías según necesidad
}

def get_metadata(file_path):
    """
    Devuelve metadatos básicos del archivo: nombre, extensión, tipo, tamaño y
fecha de modificación.
```

```
    file_path puede ser ruta completa o solo nombre con extensión.
    """
    nombre = os.path.basename(file_path)
    extension = nombre.split('.')[-1].lower() if '.' in nombre else ''
    tipo = CATEGORIAS.get(extension, "otro")
    try:
        stats = os.stat(file_path)
        fecha = datetime.fromtimestamp(stats.st_mtime).isoformat()
        tamano = stats.st_size
    except (OSError, FileNotFoundError):
        # En caso de error (por ejemplo, archivos en zip), devolvemos None
        fecha = None
        tamano = None
    return {
        "name": nombre,
        "extension": extension,
        "type": tipo,
        "size": tamano,
        "modified": fecha
    }
```

zip_reader.py

```
# zip_reader.py

import zipfile
from file_utils import get_metadata

def process_zip_file(zip_path):
    """
    Procesa un archivo ZIP y devuelve una lista de metadatos de los archivos
internos.
    No extrae archivos al disco; solo lista su información básica.
    """
    contenidos = []
    try:
        with zipfile.ZipFile(zip_path, 'r') as z:
            for info in z.infolist():
                if info.is_dir():
                    continue  # Omitir directorios internos
                nombre = info.filename
                extension = nombre.split('.')[-1].lower() if '.' in nombre
else ''
                tipo = get_metadata(nombre)['type']  # Categoriza según
extensión
                # Construir diccionario con metadata del archivo dentro del
zip
                entry = {
                    "name": nombre,
                    "path": f"{zip_path}::{nombre}",
```

```
                    "extension": extension,
                    "type": tipo,
                    "size": info.file_size,
                    # No se incluye fecha de modificación interna ni
contenido de texto
                }
                contenidos.append(entry)
    except zipfile.BadZipFile:
        print(f"Error: {zip_path} no es un archivo ZIP válido.")
    return contenidos
```

utils_ocr.py

```
# utils_ocr.py

import pytesseract
from PIL import Image
import fitz  # PyMuPDF

def ocr_image(file_path):
    """
    Extrae texto de una imagen usando Tesseract OCR.
    Requiere que Tesseract esté instalado en el sistema.
    """
    try:
        imagen = Image.open(file_path)
        texto = pytesseract.image_to_string(imagen, lang='spa')
    except Exception as e:
        print(f"Error en OCR de imagen {file_path}: {e}")
        texto = ""
    return texto

def ocr_pdf(file_path):
    """
    Intenta extraer texto de un PDF.
    Primero intenta extracción de texto con PyMuPDF; si falla o la página
está vacía, usa OCR página por página.
    """
    texto_total = ""
    try:
        doc = fitz.open(file_path)
        for pagina in doc:
            texto = pagina.get_text().strip()
            if texto:
                # Si hay texto extraído directamente, lo agregamos
                texto_total += texto + "\n"
            else:
                # Si la página no tiene texto, renderizamos la página como
imagen y aplicamos OCR
                pix = pagina.get_pixmap()
```

```
                imagen = Image.frombytes("RGB", [pix.width, pix.height],
pix.samples)
                texto = pytesseract.image_to_string(imagen, lang='spa')
                texto_total += texto + "\n"
    except Exception as e:
        print(f"Error procesando PDF {file_path}: {e}")
    return texto_total
```

duplicate_detector.py

```
# duplicate_detector.py

import hashlib

def get_file_hash(file_path, algorithm='sha256', block_size=65536):
    """
    Calcula el hash del archivo usando el algoritmo especificado (sha256 por
defecto).
    Lee el archivo en bloques para no cargarlo todo en memoria.
    """
    h = hashlib.new(algorithm)
    try:
        with open(file_path, 'rb') as f:
            while True:
                bloque = f.read(block_size)
                if not bloque:
                    break
                h.update(bloque)
    except Exception as e:
        print(f"Error al calcular hash de {file_path}: {e}")
        return None
    return h.hexdigest()
```

organizer_indexer.py

```
# organizer_indexer.py

import os
import json
from file_utils import get_metadata
from utils_ocr import ocr_image, ocr_pdf
from zip_reader import process_zip_file
from duplicate_detector import get_file_hash

def index_directory(ruta):
    """
    Recorre recursivamente la carpeta dada y genera un índice JSON con
metadatos de los archivos.
```

```
    El índice incluye nombre, ruta, tipo, tamaño, fecha, si tiene texto,
fragmento de texto, hash y si es duplicado.
    """
    indice = []
    hash_map = {}  # Para detectar duplicados usando hash de archivos
    for raiz, dirs, archivos in os.walk(ruta):
        for nombre in archivos:
            # Omitir el propio índice si existe
            if nombre.lower() == 'index.json':
                continue
            ruta_completa = os.path.join(raiz, nombre)
            # Obtener metadatos básicos
            datos = get_metadata(ruta_completa)
            datos["path"] = ruta_completa
            extension = datos["extension"]
            texto = ""
            # Extraer texto según el tipo de archivo
            if extension in ["txt", "md", "csv", "sql", "json", "html"]:
                try:
                    with open(ruta_completa, 'r', encoding='utf-8',
errors='ignore') as f:
                        texto = f.read(1000)  # lee hasta 1000 caracteres
                except Exception:
                    texto = ""
            elif extension == "pdf":
                texto = ocr_pdf(ruta_completa)
            elif extension in ["png", "jpg", "jpeg", "bmp", "gif"]:
                texto = ocr_image(ruta_completa)
            # Indicar si se detectó texto en el archivo
            datos["has_text"] = bool(texto.strip())
            datos["text_snippet"] = texto.strip()[:100]  # Fragmento corto
del texto
            # Calcular hash para detectar duplicados
            archivo_hash = get_file_hash(ruta_completa)
            datos["hash"] = archivo_hash
            # Marcar duplicados si el hash ya fue visto
            if archivo_hash:
                if archivo_hash in hash_map:
                    datos["duplicate"] = True
                    hash_map[archivo_hash].append(ruta_completa)
                else:
                    datos["duplicate"] = False
                    hash_map[archivo_hash] = [ruta_completa]
            else:
                datos["duplicate"] = False
            indice.append(datos)
            # Si es un archivo ZIP, también indexar sus archivos internos
            if extension == "zip":
                internos = process_zip_file(ruta_completa)
                for interno in internos:
                    # Los archivos internos del ZIP se añaden al índice como
```

```
elementos separados
                    interno["has_text"] = False
                    interno["text_snippet"] = ""
                    interno["hash"] = None
                    interno["duplicate"] = False
                    indice.append(interno)
    # Guardar el índice completo en un archivo JSON
    try:
        with open("index.json", "w", encoding="utf-8") as f:
            json.dump(indice, f, ensure_ascii=False, indent=2)
        print("Índice guardado en index.json")
    except Exception as e:
        print(f"Error al guardar index.json: {e}")

if __name__ == "__main__":
    carpeta = "."  # Directorio actual (cambiar según sea necesario)
    index_directory(carpeta)
```

Cada uno de estos scripts puede colocarse en un proyecto de Python independiente. Primero se ejecutarían los módulos `organizer_indexer.py` (por ejemplo desde la línea de comandos con `python organizer_indexer.py`) apuntando a la carpeta deseada. El resultado será un archivo `index.json` con todos los archivos indexados, sus metadatos y detectores de duplicados. Las funciones de OCR requieren tener instalado Tesseract y la librería PyMuPDF [1] para procesar PDFs y **pytesseract/Pillow** para imágenes. La detección de duplicados usa hashes (como se describe en recursos comunes [2]). Los comentarios en español en el código explican cada bloque para que sea fácil de estudiar.

Referencias: Uso de PyMuPDF para extracción de texto y OCR [1]; uso de hashing para detectar archivos duplicados [2].

---

[1] OCR - Optical Character Recognition - PyMuPDF documentation
https://pymupdf.readthedocs.io/en/latest/recipes-ocr.html

[2] Finding Duplicate Files with Python - GeeksforGeeks
https://www.geeksforgeeks.org/python/finding-duplicate-files-with-python/