```python
from pathlib import Path
import json
import pandas as pd
import numpy as np
import logging
# Define the log file path (same directory as the script or specify another path)
log_file_path = Path(__file__).parent / "app.log"
# Configure logging to log both to a file and the console
logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler(log_file_path), # Log to file
        logging.StreamHandler() # Log to console
    ]
)
TABLE_NAMES = {
    'tblIndividual': pd.DataFrame(),
    'tblRemarks': pd.DataFrame(),
    'tblAssociatedIndividual': pd.DataFrame(),
    'tblAssociatedCountry': pd.DataFrame(),
    'tblAssociatedDocument': pd.DataFrame(),
    'tblAssociatedOrganization': pd.DataFrame(),
}
def iterate_json_files(directory: str):
    """
    Iterate over all JSON files in the provided directory.
    Args:
    directory (str): Path to the directory containing JSON files.
    Yields:
    tuple: A tuple containing (file_name, file_path) for each JSON file.
    """
    directory = Path(directory)
    for file_path in directory.glob('*.json'):
        yield file_path.stem, file_path
def process_associations(assoc_type: str, assoc_list: list, fk_id: str, dataframes:
dict):
    """
    Process association data and append to the corresponding DataFrame.
    Args:
    assoc_type (str): The type of association (e.g., 'Individual', 'Country').
    assoc_list (list): List of associated items.
    fk_id (str): Foreign key ID.
    dataframes (dict): Dictionary containing the dataframes to append to.
    """
    df_name = f'tblAssociated{assoc_type.capitalize()}'
    if df_name in dataframes:
        df = pd.DataFrame({f'{assoc_type}_id': assoc_list, 'fkID': [fk_id] * len(assoc_list)})
        dataframes[df_name] = pd.concat([dataframes[df_name], df], ignore_index=True)
        logging.info(f"Processed association {assoc_type} for file with fkID {fk_id}.")
def process_json_files(directory: str) -> dict:
    """
    Process all JSON files in the directory and return a dictionary of DataFrames.
    Args:
    directory (str): Path to the directory containing JSON files.
    Returns:
    dict: A dictionary of dataframes with table names as keys.
    """
    dataframes = {key: pd.DataFrame() for key in TABLE_NAMES.keys()}
    for file_name, file_path in iterate_json_files(directory):
        fk_id = file_name # Use the file name without the extension as the foreign key ID
        try:
            with open(file_path, 'r') as file:
                data = json.load(file)
                logging.info(f"Processing file: {file_path}")
            if not isinstance(data, list):
                data = [data]
            for item in data:
                # Handle single value fields
                single_value_data = {k: [v if v is not None else np.nan] for k, v in item.items
                isinstance(v, (list, dict))}
                single_value_data['fkID'] = [fk_id]
                df_individual = pd.DataFrame(single_value_data)
                dataframes['tblIndividual'] = pd.concat([dataframes['tblIndividual'], df_indivi
                ignore_index=True)
                logging.info(f"Processed individual data for file with fkID {fk_id}.")
                # Handle remarks
                if 'remarks' in item and isinstance(item['remarks'], list):
                    df_remarks = pd.DataFrame({'remark': item['remarks'], 'fkID': [fk_id] *
                    len(item['remarks'])})
                    dataframes['tblRemarks'] = pd.concat([dataframes['tblRemarks'], df_remarks],
                    ignore_index=True)
                    logging.info(f"Processed remarks for file with fkID {fk_id}.")
                # Handle associations
                if 'associations' in item and isinstance(item['associations'], dict):
                    for assoc_type, assoc_list in item['associations'].items():
                        if isinstance(assoc_list, list):
                            process_associations(assoc_type, assoc_list, fk_id, dataframes)
        except (json.JSONDecodeError, IOError) as e:
            logging.error(f"Error processing file {file_path}: {e}")
            continue
    return dataframes
if __name__ == "__main__":
    # Define the directory containing the JSON files
    directory = r"C:\Users\kroy2\Documents\python\projects\json_processor\data"
    # Process the JSON files and get the resulting DataFrames
    result_dataframes = process_json_files(directory)
    # Ask the user for their choice: print DataFrames or create CSV files
    while True:
        choice = input("Would you like to print the dataframes or create CSV files? (print/csv)
        ").strip().lower()
        if choice in ['print', 'csv']:
            break
        logging.warning("Invalid choice. Please enter 'print' or 'csv'.")
    if choice == 'print':
        # If user chooses to print, iterate through the DataFrames and print them
        for table_name, df in result_dataframes.items():
            print(f"\n{table_name}:")
            print(df)
            print("\n" + "=" * 50)
            logging.info(f"Printed DataFrame for {table_name}.")
    elif choice == 'csv':
        # Create an output directory for CSV files if it doesn't exist
        output_dir = Path(directory) / 'outputs'
        output_dir.mkdir(parents=True, exist_ok=True)
        # Iterate through the DataFrames and save each as a CSV file
        for table_name, df in result_dataframes.items():
            csv_file_path = output_dir / f"{table_name}.csv"
            df.to_csv(csv_file_path, index=False)
            logging.info(f"CSV for {table_name} created at: {csv_file_path}")
```