```
import os
import json
import pandas as pd
import logging
from pathlib import Path
# Configure logging to track script progress and errors
log_file_path = Path(__file__).parent / "json_processor.log"
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler(log_file_path),
        logging.StreamHandler()
    ]
)
def sanitize_field_name(name):
    # Replace special characters in field names to ensure valid DataFrame column names
    return name.replace('.', '_').replace(' ', '_').replace('-', '_').replace(':', '_')
def iterate_json_files(directory):
    # Recursively iterate through all JSON files in the given directory and its
    subdirectories
    for root, _, files in os.walk(directory):
        for file in files:
            if file.endswith('.json'):
                yield os.path.splitext(file)[0], os.path.join(root, file)
def analyze_json_structure(directory):
    # Analyze all JSON files to determine the overall structure
    main_fields = set()
    array_fields = set()
    for _, file_path in iterate_json_files(directory):
        try:
            with open(file_path, 'r') as file:
                data = json.load(file)
                analyze_structure(data, main_fields, array_fields)
        except json.JSONDecodeError:
            logging.error(f"Error decoding JSON in file: {file_path}")
        except Exception as e:
            logging.error(f"Error processing file {file_path}: {str(e)}")
    return main_fields, array_fields
def analyze_structure(data, main_fields, array_fields, prefix=''):
    # Recursively analyze the structure of a JSON object
    if isinstance(data, dict):
        for key, value in data.items():
            new_prefix = f"{prefix}{key}." if prefix else f"{key}."
            if isinstance(value, (list, dict)):
                array_fields.add(sanitize_field_name(new_prefix.rstrip('.')))
                analyze_structure(value, main_fields, array_fields, new_prefix)
            else:
                main_fields.add(sanitize_field_name(new_prefix.rstrip('.')))
    elif isinstance(data, list):
        for item in data:
            analyze_structure(item, main_fields, array_fields, prefix)
def create_dataframes(main_fields, array_fields):
    # Create DataFrames based on the analyzed structure
    dataframes = {
        'main': pd.DataFrame(columns=list(main_fields) + ['fkID', 'classification'])
    }
    for field in array_fields:
        dataframes[field] = pd.DataFrame(columns=['fkID', 'value', 'classification'])
    return dataframes
```

```
def process_json_files(directory, dataframes):
    # Process all JSON files and populate the DataFrames
    for file_name, file_path in iterate_json_files(directory):
        try:
            with open(file_path, 'r') as file:
                data = json.load(file)
            process_json_data(data, file_name, dataframes)
            logging.info(f"Processed file: {file_path}")
        except json.JSONDecodeError:
            logging.error(f"Error decoding JSON in file: {file_path}")
        except Exception as e:
            logging.error(f"Error processing file {file_path}: {str(e)}")
def process_json_data(data, fk_id, dataframes):
    # Process a single JSON object and add its data to the appropriate DataFrames
    main_data = {'fkID': fk_id, 'classification': ''}
    for key, value in flatten_dict(data).items():
        sanitized_key = sanitize_field_name(key)
        if sanitized_key in dataframes['main'].columns:
            main_data[sanitized_key] = value
        elif sanitized_key in dataframes:
            if isinstance(value, list):
                for item in value:
                    dataframes[sanitized_key] = dataframes[sanitized_key].append(
                        {'fkID': fk_id, 'value': item, 'classification': ''},
                        ignore_index=True
                    )
            else:
                dataframes[sanitized_key] = dataframes[sanitized_key].append(
                    {'fkID': fk_id, 'value': value, 'classification': ''},
                    ignore_index=True
                )
    dataframes['main'] = dataframes['main'].append(main_data, ignore_index=True)
def flatten_dict(d, parent_key='', sep='.'):
    # Flatten a nested dictionary
    items = []
    for k, v in d.items():
        new_key = f"{parent_key}{sep}{k}" if parent_key else k
        if isinstance(v, dict):
            items.extend(flatten_dict(v, new_key, sep=sep).items())
        else:
            items.append((new_key, v))
    return dict(items)
def save_dataframes_as_csv(dataframes, output_dir):
    # Save all DataFrames as CSV files
    output_dir.mkdir(parents=True, exist_ok=True)
    for table_name, df in 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}")
# Main execution block
if __name__ == "__main__":
    try:
        # Specify the directory containing JSON files
        directory = r"C:\path\to\your\json\files" # Replace this with your actual directory patl
        # Validate the directory path
        if not os.path.isdir(directory):
            raise ValueError(f"Invalid directory path: {directory}")
        logging.info("Starting JSON analysis...")
        main_fields, array_fields = analyze_json_structure(directory)
        logging.info(f"Found {len(main_fields)} main fields and {len(array_fields)} array/dict
        fields")
```

```
    logging.info("Creating DataFrames...")
    dataframes = create_dataframes(main_fields, array_fields)
    logging.info("Processing JSON files...")
    process_json_files(directory, dataframes)
    # Save all DataFrames as CSV files
    output_dir = Path(directory) / 'outputs'
    logging.info("Saving DataFrames as CSV files...")
    save_dataframes_as_csv(dataframes, output_dir)
    logging.info("Processing complete. CSV files have been saved in the 'outputs' folder.")
except Exception as e:
    logging.error(f"An error occurred: {str(e)}")
```