Nome do Arquivo

main.py

Documentação

```
#1[
#1 TITULO: MAIN
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 07/10/2024
#1 VERSAO: 1
#1 FINALIDADE: INICIALIZAR O HANDLER DE DATAFRAMES E EXECUTAR SUAS
FUNÇÕES
#1 ENTRADAS: NENHUMA
#1 SAIDAS: EXECUÇÃO DAS FUNÇÕES DO DATAFRAMEHANDLER
#1 ROTINAS CHAMADAS: EXECUTE (DATAFRAMEHANDLER)
#1]

from dataframe_logic.dataframe_handler import DataframeHandler

#1[
#1 ROTINA: MAIN
#1 FINALIDADE: CRIA UMA INSTÂNCIA DO DATAFRAMEHANDLER E EXECUTA SUAS
FUNÇÕES
#1 ENTRADAS: NENHUMA
#1 DEPENDENCIAS: DATAFRAMEHANDLER
#1 CHAMADO POR: SCRIPT PRINCIPAL
#1 CHAMA: EXECUTE (DATAFRAMEHANDLER)
#1]
#2[
#2 PSEUDOCODIGO DE: MAIN
def main():
    #2 INICIALIZA O HANDLER DE DATAFRAMES
    df_handler = DataframeHandler()
    #2 EXECUTA A FUNÇÃO PRINCIPAL DO HANDLER
    df_handler.execute()
#2]
main()
```

Nome do Arquivo

toponym_handler.py

Documentação

```
#1[
#1 TITULO: TOPONYMHANDLER
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 07/10/2024
#1 VERSAO: 1
#1 FINALIDADE: LEMATIZAR TEXTOS E EXTRAIR TOPÔNIMOS (NOMES DE LOCAIS)
USANDO PADRÕES DEFINIDOS
#1 ENTRADAS: NOME DO DIRETÓRIO (NO CONSTRUTOR), TEXTO (NA FUNÇÃO
LEMMATIZE_AND_EXTRACT_TOPONYMS)
#1 SAIDAS: TEXTO LEMATIZADO E LISTA DE TOPÔNIMOS EXTRAÍDOS
#1 ROTINAS CHAMADAS: _ADD_PATTERNS, LEMMATIZE_AND_EXTRACT_TOPONYMS
#1]

import re
import spacy
from spacy.matcher import Matcher
from logging_logic.logging_handler import LoggingHandler

class ToponymHandler(LoggingHandler):
    #1[
    #1 ROTINA: __INIT__
    #1 FINALIDADE: INICIALIZA A CLASSE TOPONYMHANDLER E CONFIGURA O
PROCESSAMENTO DE LINGUAGEM NATURAL E PADRÕES DE TOPÔNIMOS
    #1 ENTRADAS: NOME DO DIRETÓRIO (STRING)
    #1 DEPENDENCIAS: SPACY, MATCHER, LOGGINGHANDLER
    #1 CHAMADO POR: TOPONYMHANDLER
    #1 CHAMA: LOGGINGHANDLER.__INIT__, SPACY.LOAD, MATCHER
    #1]
    #2[
    #2 PSEUDOCODIGO DE: __INIT__
    def __init__(self, directory_name):
        #2 CHAMA O CONSTRUTOR DA CLASSE PAI E INICIALIZA O DIRETÓRIO
        super().__init__(directory_name)
```

```
        #2 ARMAZENA O NOME DO DIRETÓRIO
        self.directory_name = directory_name
        #2 CARREGA O MODELO DE LINGUAGEM NATURAL EM PORTUGUÊS USANDO
SPACY
        self.nlp = spacy.load("pt_core_news_sm")
        #2 INICIALIZA O MATCHER (PARA DETECTAR PADRÕES DE TOPÔNIMOS)
        self.matcher = Matcher(self.nlp.vocab)
        #2 CHAMA A FUNÇÃO PARA ADICIONAR PADRÕES AO MATCHER
        self._add_patterns()
    #2]

    #1[
    #1 ROTINA: _ADD_PATTERNS
    #1 FINALIDADE: ADICIONA PADRÕES DE TOPÔNIMOS AO MATCHER USANDO NOMES
PRÓPRIOS E PALAVRAS-CHAVE RELACIONADAS A LOCAIS
    #1 ENTRADAS: NENHUMA
    #1 DEPENDENCIAS: SPACY, RE
    #1 CHAMADO POR: __INIT__
    #1 CHAMA: MATCHER.ADD
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _ADD_PATTERNS
    @LoggingHandler.log_method("ToponymHandler", "_add_patterns")
    def _add_patterns(self):
        #2 DEFINE OS TERMOS RELACIONADOS A LOCAIS E SUAS ABREVIAÇÕES
        location_keywords = [
            "AV", "AV.", "AVENIDA",
            "R", "R.", "RUA",
            "ESTRADA", "PONTE", "AEROPORTO", "CIDADE", "MUNICÍPIO",
            "PRAÇA", "RODOVIA", "TRAVESSA", "BAIRRO", "DISTRITO", "LAGO",
"RIO", "VILA"
        ]

        #2 DEFINE PADRÕES FIXOS QUE CORRESPONDEM A NOMES PRÓPRIOS E
ESTRUTURAS COMUNS EM NOMES DE LOCAIS
        base_patterns = [
            #2 EX: SÃO PAULO
            [{"POS": "PROPN"}],
            #2 EX: RIO DE JANEIRO
            [{"POS": "PROPN"}, {"POS": "ADP"}, {"POS": "PROPN"}],
            #2 EX: CIDADE DE SÃO PAULO
            [{"POS": "NOUN"}, {"POS": "ADP"}, {"POS": "PROPN"}],
            #2 EX: VILA NOVA DE GAIA
            [{"POS": "NOUN"}, {"POS": "ADJ"}, {"POS": "ADP"}, {"POS":
"PROPN"}],
            #2 EX: SÃO PEDRO E SÃO PAULO
            [{"POS": "PROPN"}, {"POS": "CCONJ"}, {"POS": "PROPN"}],
            #2 EX: O RIO DE JANEIRO
            [{"POS": "DET"}, {"POS": "PROPN"}, {"POS": "ADP"}, {"POS":
"PROPN"}],
            #2 EX: A CIDADE DE SÃO PAULO
            [{"POS": "DET"}, {"POS": "NOUN"}, {"POS": "ADP"}, {"POS":
"PROPN"}],
            #2 EX: A CIDADE DE SÃO PAULO E RIO DE JANEIRO
            [{"POS": "DET"}, {"POS": "NOUN"}, {"POS": "ADP"}, {"POS":
"PROPN"}, {"POS": "CCONJ"}, {"POS": "PROPN"}],
            #2 EX: ALOYSIO NUNES
            [{"POS": "PROPN"}, {"POS": "PROPN"}],
            #2 EX: PEDRO ÁLVARES CABRAL
            [{"POS": "PROPN"}, {"POS": "PROPN"}, {"POS": "PROPN"}],
            #2 EX: ALOYSIO NUNES FERREIRA FILHO
            [{"POS": "PROPN"}, {"POS": "PROPN"}, {"POS": "PROPN"},
{"POS": "PROPN"}],
        ]

        #2 DEFINE PADRÕES PARA NOMES PRÓPRIOS COM HÍFEN
        hyphenated_patterns = [
            #2 DOIS NOMES PRÓPRIOS COM HÍFEN EX: PEDRO ÁLVARES-CABRAL
            [{"POS": "PROPN"}, {"ORTH": "-"}, {"POS": "PROPN"}],
            #2 TRÊS NOMES PRÓPRIOS COM UM HÍFEN EX: PEDRO ÁLVARES-CABRAL
FILHO
            [{"POS": "PROPN"}, {"ORTH": "-"}, {"POS": "PROPN"}, {"POS":
"PROPN"}],
            #2 DOIS NOMES PRÓPRIOS SEGUIDOS DE UM HÍFEN E MAIS DOIS NOMES
PRÓPRIOS EX: JOÃO PAULO-SILVA
            [{"POS": "PROPN"}, {"POS": "PROPN"}, {"ORTH": "-"}, {"POS":
"PROPN"}],
            #2 TRÊS NOMES PRÓPRIOS COM TRES HÍFENS EX: JOÃO-PAULO-SILVA
            [{"POS": "PROPN"}, {"ORTH": "-"}, {"POS": "PROPN"}, {"ORTH":
"-"}, {"POS": "PROPN"}],
            #2 TRÊS NOMES PRÓPRIOS COM DOIS HÍFENS EX: JOÃO-PAULO-SILVA
FILHO
            [{"POS": "PROPN"}, {"ORTH": "-"}, {"POS": "PROPN"}, {"ORTH":
```

```
"-"}, {"POS": "PROPN"}, {"POS": "PROPN"}],
            #2 QUATRO NOMES PRÓPRIOS COM DOIS HÍFENS EX: JOÃO-PAULO-
SILVA-SANTOS
            [{"POS": "PROPN"}, {"ORTH": "-"}, {"POS": "PROPN"}, {"ORTH":
"-"}, {"POS": "PROPN"}, {"ORTH": "-"}, {"POS": "PROPN"}],
        ]

        patterns = []

        #2 ADICIONA OS PADRÕES FIXOS AO MATCHER
        patterns.extend(base_patterns)

        #2 ADICIONA VARIAÇÕES DOS PADRÕES USANDO PALAVRAS-CHAVE DE LOCAIS
        for keyword in location_keywords:
            for pattern in base_patterns:
                #2 CADA PADRÃO BASE É PRECEDIDO POR UMA PALAVRA-CHAVE DE
LOCAL
                patterns.append([{"TEXT": keyword}] + pattern)

            #2 ADICIONA VARIAÇÕES COM HÍFENS PARA CADA PALAVRA-CHAVE
            for hyphen_pattern in hyphenated_patterns:
                patterns.append([{"TEXT": keyword}] + hyphen_pattern)

        #2 ADICIONA OS PADRÕES COM HÍFEN AO MATCHER
        patterns.extend(hyphenated_patterns)

        #2 ADICIONA TODOS OS PADRÕES AO MATCHER
        for pattern in patterns:
            self.matcher.add("TOPONIMO", [pattern])
    #2]

    #1[
    #1 ROTINA: LEMMATIZE_AND_EXTRACT_TOPONYMS
    #1 FINALIDADE: LEMATIZA O TEXTO E EXTRAI OS TOPÔNIMOS USANDO O
MATCHER
    #1 ENTRADAS: TEXTO (STRING)
    #1 DEPENDENCIAS: SPACY, MATCHER
    #1 CHAMADO POR: USUÁRIO
    #1 CHAMA: MATCHER, NLP
    #1]
    #2[
    #2 PSEUDOCODIGO DE: LEMMATIZE_AND_EXTRACT_TOPONYMS
    @LoggingHandler.log_method("ToponymHandler",
"lemmatize_and_extract_toponyms")
    def lemmatize_and_extract_toponyms(self, text):
        try:
            #2 PROCESSA O TEXTO USANDO O MODELO NLP
            doc = self.nlp(text)
            #2 LEMATIZA CADA TOKEN DO TEXTO
            lemmatized_text = " ".join([token.lemma_ for token in doc])
            #2 ENCONTRA OS PADRÕES CORRESPONDENTES AOS TOPÔNIMOS NO TEXTO
            matches = self.matcher(doc)
            #2 EXTRAI OS TOPÔNIMOS IDENTIFICADOS
            toponyms = [doc[start:end].text for match_id, start, end in
matches]
            #2 RETORNA O TEXTO LEMATIZADO E OS TOPÔNIMOS COMO UMA STRING
SEPARADA POR VÍRGULAS
            return lemmatized_text, ",".join(toponyms)
        except Exception as e:
            #2 RETORNA STRINGS VAZIAS EM CASO DE ERRO
            return "", ""
    #2]
```

Nome do Arquivo

logging_handler.py

Documentação

```
#1[
#1 TITULO: LOGGING HANDLER
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 07/10/2024
#1 VERSAO: 1
#1 FINALIDADE: GERENCIA O LOGGING PARA UMA CLASSE, PERMITINDO A CRIACAO
DE LOGS E FORMATO ESPECIFICO, E LOGAR EXECUCAO DE METODOS COM PARAMETROS
E RESULTADOS
#1 ENTRADAS: NOME DO DIRETORIO PARA ARMAZENAR OS LOGS, NOME DA CLASSE,
NOME DO METODO, OPCOES PARA LOGAR PARAMETROS E RESULTADOS
#1 SAIDAS: LOGS EM ARQUIVO DE TEXTO
```

```
#1 ROTINAS CHAMADAS: __INIT__, LOG_METHOD
#1 DEPENDENCIAS: LOGGING, OS
#1]

import logging
import os

#1[
#1 ROTINA: __INIT__
#1 FINALIDADE: INICIALIZA O MANIPULADOR DE LOGS E CONFIGURA O FORMATO DE
LOGS PARA A CLASSE LOGGINGHANDLER
#1 ENTRADAS: NOME DO DIRETORIO PARA ARMAZENAR OS LOGS
#1 DEPENDENCIAS: LOGGING, OS
#1 CHAMADO POR: LOGGINGHANDLER
#1 CHAMA: OS.MAKEDIRS (SE O DIRETORIO NAO EXISTIR), LOGGING.GETLOGGER,
LOGGING.FILEHANDLER, LOGGING.FORMATTER
#1]
#2[
#2 PSEUDOCODIGO DE: __INIT__
class LoggingHandler:
    def __init__(self, directory_name):
        #2 OBTEM UM LOGGER ASSOCIADO AO NOME DA CLASSE
        self.logger = logging.getLogger(self.__class__.__name__)
        #2 VERIFICA SE O DIRETORIO EXISTE, SE NAO, CRIA O DIRETORIO
        if not os.path.exists(directory_name):
            os.makedirs(directory_name)
        #2 DEFINE O CAMINHO DO ARQUIVO DE LOGS COMO LOGS.TXT
        file_path = os.path.join(directory_name, "logs.txt")
        #2 CRIA UM MANIPULADOR DE LOG PARA O ARQUIVO
        handler = logging.FileHandler(file_path)
        #2 DEFINE O FORMATO DO LOG COM O NOME DA CLASSE E O NOME DO
METODO
        formatter = logging.Formatter("[+] %(class_name)s.%(method_name)s
%(message)s")
        handler.setFormatter(formatter)
        #2 ADICIONA O MANIPULADOR AO LOGGER
        self.logger.addHandler(handler)
        #2 DEFINE O NIVEL DE LOG PARA INFO
        self.logger.setLevel(logging.INFO)
#2]

#1[
#1 ROTINA: LOG_METHOD
#1 FINALIDADE: DECORADOR PARA LOGAR A EXECUCAO DE METODOS, COM OPCOES
PARA EXIBIR PARAMETROS E RESULTADOS
#1 ENTRADAS: NOME DA CLASSE, NOME DO METODO, OPCOES DE EXIBIR PARAMETROS
E EXIBIR RESULTADO
#1 DEPENDENCIAS: LOGGING
#1 CHAMADO POR: LOGGINGHANDLER
#1 CHAMA: FUNC (METODO DECORADO)
#1]
#2[
#2 PSEUDOCODIGO DE: LOG_METHOD
    @staticmethod
    def log_method(class_name, method_name, show_output=False,
show_parameters=False):
        #2 DEFINE UM DECORADOR PARA O METODO
        def decorator(func):
            #2 CRIA O WRAPPER QUE ENVOLVE O METODO ORIGINAL
            def wrapper(self, *args, **kwargs):
                #2 SE EXIBIR PARAMETROS, GERA UMA STRING COM OS
PARAMETROS
                if show_parameters:
                    parameters = f"Parameters: args={args}, kwargs=
{kwargs}"
                else:
                    parameters = ""
                #2 LOGA OS PARAMETROS UTILIZANDO O LOGGER
                logging.getLogger(class_name).info(parameters, extra=
{'class_name': class_name, 'method_name': method_name})
                #2 EXECUTA O METODO ORIGINAL E OBTEM O RESULTADO
                result = func(self, *args, **kwargs)
                #2 SE EXIBIR O RESULTADO, GERA UMA MENSAGEM DE LOG COM O
RESULTADO
                if show_output:
                    output_message = f"Output: {result}"
                    logging.getLogger(class_name).info(output_message,
extra={'class_name': class_name, 'method_name': method_name})
                #2 RETORNA O RESULTADO DA EXECUCAO DO METODO ORIGINAL
                return result
            #2 RETORNA O WRAPPER COMO O NOVO METODO DECORADO
            return wrapper
        #2 RETORNA O DECORADOR
```

```
        return decorator
#2]
```

Nome do Arquivo
alesp.py

Documentação

```
#1[
#1 TITULO: ALESP SCRAPPER
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 07/10/2024
#1 VERSAO: 1
#1 FINALIDADE: REALIZAR O SCRAPING DE NORMAS DA ASSEMBLEIA LEGISLATIVA DO
ESTADO DE SÃO PAULO (ALESP) COM BASE EM DETERMINADOS ASSUNTOS
#1 ENTRADAS: NOME DO DIRETÓRIO, ASSUNTO (NA FUNÇÃO SCRAPE)
#1 SAIDAS: DADOS DE RESPOSTA PROCESSADOS, URLS VISITADAS
#1 ROTINAS CHAMADAS: SCRAPE, BUILD_URL, PARSE_CONTENT, EXTRACT_LINKS,
SCRAPE_LINKS, SCRAPE_ALL_SUBJECTS
#1]

import time
import urllib.parse
from bs4 import BeautifulSoup
from concurrent.futures import ThreadPoolExecutor
from logging_logic.logging_handler import LoggingHandler
from scraping_logic.scraping_handler import ScrapingHandler

class AlespScrapper(ScrapingHandler):
    BASE_URL = "https://www.al.sp.gov.br/norma"
    NORMA_BASE_URL = "https://www.al.sp.gov.br"
    SUBJECTS = ["Alteração", "Mudança", "Modificação", "Reforma",
"Transformação", "Troca", "Reajuste", "Revisão", "Ajuste", "Correção"]
    DIRECTORY_NAME = "Alesp"
    SLEEP = 15
    MAX_ERROR = 3

    #1[
    #1 ROTINA: __INIT__
    #1 FINALIDADE: INICIALIZA A CLASSE ALESP SCRAPPER E CONFIGURA AS
VARIÁVEIS INICIAIS E O DIRETÓRIO DE SALVAMENTO
    #1 ENTRADAS: NENHUMA
    #1 DEPENDENCIAS: LOGGINGHANDLER, SCRAPINGHANDLER
    #1 CHAMADO POR: ALESP SCRAPPER
    #1 CHAMA: SCRAPINGHANDLER.__INIT__
    #1]
    #2[
    #2 PSEUDOCODIGO DE: __INIT__
    @LoggingHandler.log_method('AlespScrapper', '__init__',
show_parameters=False, show_output=False)
    def __init__(self):
        #2 INICIALIZA A CLASSE PAI E O DIRETÓRIO
        super().__init__(self.DIRECTORY_NAME)
        #2 INICIALIZA O CONJUNTO DE URLS VISITADAS
        self.visited_urls = set()
    #2]

    #1[
    #1 ROTINA: SCRAPE
    #1 FINALIDADE: REALIZAR O SCRAPING DE NORMAS COM BASE NO ASSUNTO
ESPECIFICADO
    #1 ENTRADAS: ASSUNTO (STRING)
    #1 DEPENDENCIAS: URLLIB, TIME, REQUESTINGHANDLER
    #1 CHAMADO POR: USUÁRIO, SCRAPE_ALL_SUBJECTS
    #1 CHAMA: BUILD_URL, MAKE_REQUEST (REQUESTINGHANDLER),
PROCESS_RESPONSE, PARSE_CONTENT, EXTRACT_LINKS, SCRAPE_LINKS
    #1]
    #2[
    #2 PSEUDOCODIGO DE: SCRAPE
    @LoggingHandler.log_method('AlespScrapper', 'scrape',
show_parameters=False, show_output=False)
    def scrape(self, subject):
        #2 ESCAPA O ASSUNTO PARA FORMATO DE URL E DEPOIS DECODIFICA
        escaped_subject = urllib.parse.quote(subject)
        decoded_subject = urllib.parse.unquote(escaped_subject)
        page_number = 1
        error_count = 0
        #2 REALIZA O LOOP PARA TENTAR ACESSAR PÁGINAS ENQUANTO O NÚMERO
MÁXIMO DE ERROS NÃO FOR ALCANÇADO
```

```
        while error_count < self.MAX_ERROR:
            #2 CONSTROI A URL COM BASE NO ASSUNTO E NÚMERO DA PÁGINA
            url = self.build_url(page_number, escaped_subject,
decoded_subject)
            #2 REALIZA A REQUISIÇÃO HTTP
            content = self.requesting_handler.make_request('GET', url)
            #2 PROCESSA A RESPOSTA
            self.process_response(content)
            if not content.get('error'):
                #2 ANALISA O CONTEÚDO HTML
                parsed_content =
self.parse_content(content['response_string'])
                #2 EXTRAI OS LINKS ENCONTRADOS NO CONTEÚDO HTML
                links = self.extract_links(parsed_content)
                #2 REALIZA O SCRAPING DOS LINKS ENCONTRADOS
                self.scrape_links(links)
                error_count = 0
            else:
                error_count += 1
            page_number += 1
            time.sleep(self.SLEEP)
    #2]

    #1[
    #1 ROTINA: BUILD_URL
    #1 FINALIDADE: CONSTRUIR A URL PARA REALIZAR A REQUISIÇÃO HTTP
BASEADO NO ASSUNTO E NO NÚMERO DA PÁGINA
    #1 ENTRADAS: NÚMERO DA PÁGINA (INT), ASSUNTO ESCAPADO (STRING),
ASSUNTO DECODIFICADO (STRING)
    #1 DEPENDENCIAS: NENHUMA
    #1 CHAMADO POR: SCRAPE
    #1 CHAMA: NENHUMA
    #1]
    #2[
    #2 PSEUDOCODIGO DE: BUILD_URL
    @LoggingHandler.log_method('AlespScrapper', 'build_url',
show_parameters=False, show_output=False)
    def build_url(self, page_number, escaped_subject, decoded_subject):
        #2 CONSTROI A URL PARA BUSCA DE NORMAS COM BASE NO NÚMERO DA
PÁGINA E NO ASSUNTO
        return (f"{self.BASE_URL}/resultados?page=
{page_number}&size=10&tipoPesquisa=E&"

f"buscaLivreEscape=&buscaLivreDecode=&_idsTipoNorma=1&idsTipoNorma=9&"

f"idsTipoNorma=2&idsTipoNorma=55&idsTipoNorma=3&idsTipoNorma=28&"

f"idsTipoNorma=25&idsTipoNorma=1&idsTipoNorma=19&idsTipoNorma=14&"

f"idsTipoNorma=12&idsTipoNorma=21&idsTipoNorma=22&idsTipoNorma=23&"

f"idsTipoNorma=59&nuNorma=&ano=&complemento=&dtNormaInicio=&dtNormaFim=&"
                f"idTipoSituacao=0&_idsTema=1&palavraChaveEscape=
{escaped_subject}&"
                f"palavraChaveDecode=
{decoded_subject}&_idsAutorProposicao=1&_temQuestionamentos=on")
    #2]

    #1[
    #1 ROTINA: PARSE_CONTENT
    #1 FINALIDADE: ANALISAR O CONTEÚDO HTML OBTIDO NA REQUISIÇÃO E
TRANSFORMÁ-LO EM UM OBJETO BEAUTIFULSOUP
    #1 ENTRADAS: CONTEÚDO HTML (STRING)
    #1 DEPENDENCIAS: BEAUTIFULSOUP
    #1 CHAMADO POR: SCRAPE
    #1 CHAMA: NENHUMA
    #1]
    #2[
    #2 PSEUDOCODIGO DE: PARSE_CONTENT
    @LoggingHandler.log_method('AlespScrapper', 'parse_content',
show_parameters=False, show_output=False)
    def parse_content(self, content):
        #2 TRANSFORMA O CONTEÚDO HTML EM UM OBJETO BEAUTIFULSOUP PARA
FACILITAR A EXTRAÇÃO DE DADOS
        return BeautifulSoup(content, 'html.parser')
    #2]

    #1[
    #1 ROTINA: EXTRACT_LINKS
    #1 FINALIDADE: EXTRAIR LINKS RELEVANTES DO CONTEÚDO HTML ANALISADO
    #1 ENTRADAS: CONTEÚDO HTML ANALISADO (OBJETO BEAUTIFULSOUP)
    #1 DEPENDENCIAS: BEAUTIFULSOUP
    #1 CHAMADO POR: SCRAPE
```

```
    #1 CHAMA: NENHUMA
    #1]
    #2[
    #2 PSEUDOCODIGO DE: EXTRACT_LINKS
    @LoggingHandler.log_method('AlespScrapper', 'extract_links',
show_parameters=False, show_output=False)
    def extract_links(self, parsed_content):
        #2 EXTRAI TODOS OS LINKS QUE CONTÊM 'NORMA' NO CAMINHO DO HREF
        return [a['href'] for a in parsed_content.find_all('a',
href=True) if 'norma' in a['href']]
    #2]

    #1[
    #1 ROTINA: SCRAPE_LINKS
    #1 FINALIDADE: REALIZAR O SCRAPING DE CADA LINK EXTRAÍDO, SEGUINDO OS
LINKS E PROCESSANDO AS RESPOSTAS
    #1 ENTRADAS: LISTA DE LINKS (LISTA DE STRINGS)
    #1 DEPENDENCIAS: TIME, REQUESTINGHANDLER
    #1 CHAMADO POR: SCRAPE
    #1 CHAMA: MAKE_REQUEST (REQUESTINGHANDLER), PROCESS_RESPONSE
    #1]
    #2[
    #2 PSEUDOCODIGO DE: SCRAPE_LINKS
    @LoggingHandler.log_method('AlespScrapper', 'scrape_links',
show_parameters=False, show_output=False)
    def scrape_links(self, links):
        #2 ITERA SOBRE CADA LINK EXTRAÍDO
        for link in links:
            #2 CRIA A URL COMPLETA CONCATENANDO O LINK COM A BASE_URL DA
NORMA
            full_url = f"{self.NORMA_BASE_URL}{link}"
            #2 VERIFICA SE A URL JÁ FOI VISITADA
            if full_url not in self.visited_urls:
                #2 ADICIONA A URL AO CONJUNTO DE URLs VISITADAS
                self.visited_urls.add(full_url)
                #2 FAZ A REQUISIÇÃO PARA A URL DO LINK
                content = self.requesting_handler.make_request('GET',
full_url)
                #2 PROCESSA A RESPOSTA RECEBIDA
                self.process_response(content)
                #2 AGUARDA O TEMPO CONFIGURADO ENTRE REQUISIÇÕES
                time.sleep(self.SLEEP)
    #2]

    #1[
    #1 ROTINA: SCRAPE_ALL_SUBJECTS
    #1 FINALIDADE: REALIZAR O SCRAPING DE TODOS OS ASSUNTOS DEFINIDOS NA
VARIÁVEL SUBJECTS UTILIZANDO MÚLTIPLAS THREADS
    #1 ENTRADAS: NENHUMA (UTILIZA OS ASSUNTOS DA VARIÁVEL DE CLASSE
SUBJECTS)
    #1 DEPENDENCIAS: THREADPOOLEXECUTOR
    #1 CHAMADO POR: USUÁRIO, FUNÇÃO MAIN
    #1 CHAMA: SCRAPE (PARA CADA ASSUNTO)
    #1]
    #2[
    #2 PSEUDOCODIGO DE: SCRAPE_ALL_SUBJECTS
    @LoggingHandler.log_method('AlespScrapper', 'scrape_all_subjects',
show_parameters=False, show_output=False)
    def scrape_all_subjects(self):
        #2 UTILIZA UM THREADPOOLEXECUTOR PARA PARALELIZAR O PROCESSO DE
SCRAPING PARA TODOS OS ASSUNTOS
        with ThreadPoolExecutor(max_workers=len(self.SUBJECTS)) as
executor:
            #2 EXECUTA A FUNÇÃO SCRAPE PARA CADA ASSUNTO EM SUBJECTS
SIMULTANEAMENTE
            executor.map(self.scrape, self.SUBJECTS)
    #2]

if __name__ == '__main__':
    scraper = AlespScrapper()
    scraper.scrape_all_subjects()
```

Nome do Arquivo
dataframe_handler.py
Documentação

```
#1[
#1 TITULO: DATAFRAMEHANDLER
```

```
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 07/10/2024
#1 VERSAO: 1
#1 FINALIDADE: MANIPULAR E PROCESSAR ARQUIVOS JSON, TRANSFORMANDO-OS EM
DATAFRAMES E EXTRAINDO INFORMAÇÕES ÚTEIS.
#1 ENTRADAS: NENHUMA
#1 SAIDAS: ARQUIVOS CSV GERADOS A PARTIR DE DATAFRAMES
#1 ROTINAS CHAMADAS: LOAD_JSON_FILES, _PROCESS_AND_SAVE_DATAFRAME,
EXTRACT_URL_INFORMATION, _EXTRACT_INSTITUTION_NAME, _EXTRACT_SUBJECT,
EXECUTE
#1]

import os
import json
import pandas as pd
import re
from logging_logic.logging_handler import LoggingHandler
from tqdm import tqdm

class DataframeHandler(LoggingHandler):
    DIRECTORY_PATH = os.getcwd()
    DIRECTORY_NAME = "DataframeHandler"

    #1[
    #1 ROTINA: __INIT__
    #1 FINALIDADE: INICIALIZA AS VARIÁVEIS DA CLASSE DATAFRAMEHANDLER E
CHAMA O CONSTRUTOR DA CLASSE PAI.
    #1 ENTRADAS: NENHUMA
    #1 DEPENDENCIAS: LOGGINGHANDLER, OS
    #1 CHAMADO POR: DATAFRAMEHANDLER
    #1 CHAMA: LOGGINGHANDLER.__INIT__
    #1]
    #2[
    #2 PSEUDOCODIGO DE: __init__
    def __init__(self):
        #2 CHAMA O CONSTRUTOR DA CLASSE PAI PASSANDO O NOME DO DIRETÓRIO
        super().__init__(self.DIRECTORY_NAME)
    #2]

    #1[
    #1 ROTINA: LOAD_JSON_FILES
    #1 FINALIDADE: CARREGA TODOS OS ARQUIVOS JSON DO DIRETÓRIO E OS
PROCESSA.
    #1 ENTRADAS: NENHUMA
    #1 DEPENDENCIAS: OS, JSON, TQDM, RE
    #1 CHAMADO POR: EXECUTE
    #1 CHAMA: _PROCESS_AND_SAVE_DATAFRAME
    #1]
    #2[
    #2 PSEUDOCODIGO DE: load_json_files
    @LoggingHandler.log_method("DataframeHandler", "load_json_files")
    def load_json_files(self):
        #2 ITERA SOBRE OS ARQUIVOS NO DIRETÓRIO ATUAL
        for root, _, files in os.walk(self.DIRECTORY_PATH):
            all_data = []
            #2 FILTRA ARQUIVOS COM EXTENSÃO .JSON
            json_files = [f for f in files if f.endswith('.json')]
            #2 CARREGA CADA ARQUIVO JSON E LIMPA OS DADOS
            for file_name in tqdm(json_files, desc=f'Procurando na pasta:
{root}', unit='file'):
                file_path = os.path.join(root, file_name)
                with open(file_path, 'r', encoding='utf-8') as file:
                    data = json.load(file)
                    cleaned_data = {k: re.sub(r'\s+', ' ', str(v)) for k,
v in data.items()}
                    all_data.append(cleaned_data)
            #2 SE EXISTIREM DADOS, PROCESSA E SALVA EM CSV
            if all_data:
                self._process_and_save_dataframe(all_data, root)
    #2]

    #1[
    #1 ROTINA: _PROCESS_AND_SAVE_DATAFRAME
    #1 FINALIDADE: PROCESSA OS DADOS E SALVA EM ARQUIVOS CSV NO DIRETÓRIO
CORRESPONDENTE.
    #1 ENTRADAS: LISTA DE DICIONÁRIOS COM OS DADOS JSON E O CAMINHO DO
DIRETÓRIO
    #1 DEPENDENCIAS: PANDAS, OS
    #1 CHAMADO POR: LOAD_JSON_FILES
    #1 CHAMA: EXTRACT_URL_INFORMATION
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _process_and_save_dataframe
```

```
    def _process_and_save_dataframe(self, all_data, root):
        #2 CRIA UM DATAFRAME A PARTIR DOS DADOS JSON
        df = pd.DataFrame(all_data)
        #2 EXTRAI INFORMAÇÕES DA URL E ADICIONA AO DATAFRAME
        self.extract_url_information(df)
        #2 OBTÉM O NOME DA INSTITUIÇÃO, OU DEFINE COMO 'DESCONHECIDO'
        institution_name = df['institution_name'].iloc[0] if
'institution_name' in df.columns else 'unknown'
        #2 DEFINE O CAMINHO DE SAÍDA PARA O CSV
        output_csv_path = os.path.join(root,
f'{institution_name}_dataframe')
        #2 DEFINE O TAMANHO DE CADA LOTE DE CSV A SER GERADO
        chunk_size = len(df) // 10 + 1
        #2 GERA E SALVA CADA PARTE DO CSV
        for i, chunk in enumerate(range(0, len(df), chunk_size)):
            chunk_df = df.iloc[chunk:chunk + chunk_size]
            chunk_df.to_csv(f'{output_csv_path}_part_{i + 1}.csv',
index=False, sep='|', quoting=1)
    #2]

    #1[
    #1 ROTINA: EXTRACT_URL_INFORMATION
    #1 FINALIDADE: EXTRAI INFORMAÇÕES RELEVANTES DAS URLS PRESENTES NO
DATAFRAME.
    #1 ENTRADAS: DATAFRAME COM A COLUNA 'URL'
    #1 DEPENDENCIAS: RE
    #1 CHAMADO POR: _PROCESS_AND_SAVE_DATAFRAME
    #1 CHAMA: _EXTRACT_INSTITUTION_NAME, _EXTRACT_SUBJECT
    #1]
    #2[
    #2 PSEUDOCODIGO DE: extract_url_information
    @LoggingHandler.log_method("DataframeHandler",
"extract_url_information")
    def extract_url_information(self, dataframe):
        #2 VERIFICA SE A COLUNA 'URL' EXISTE NO DATAFRAME
        if 'url' not in dataframe.columns:
            raise ValueError("A coluna 'url' não está presente no
DataFrame.")
        #2 EXTRAI O DOMÍNIO BASE DA URL
        dataframe['base_url'] = dataframe['url'].apply(lambda x:
x.split('/')[2] if x and isinstance(x, str) else None)
        #2 EXTRAI O NOME DA INSTITUIÇÃO A PARTIR DO DOMÍNIO
        dataframe['institution_name'] =
dataframe['base_url'].apply(self._extract_institution_name)
        #2 EXTRAI O ASSUNTO RELACIONADO COM A URL
        dataframe['subject'] = dataframe.apply(lambda row:
self._extract_subject(row['url'], row['institution_name']), axis=1)
    #2]

    #1[
    #1 ROTINA: _EXTRACT_INSTITUTION_NAME
    #1 FINALIDADE: IDENTIFICA O NOME DA INSTITUIÇÃO BASEADO NO DOMÍNIO DA
URL.
    #1 ENTRADAS: BASE_URL (STRING)
    #1 DEPENDENCIAS: RE
    #1 CHAMADO POR: EXTRACT_URL_INFORMATION
    #1 CHAMA: NENHUMA
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _extract_institution_name
    def _extract_institution_name(self, base_url):
        #2 VERIFICA SE O DOMÍNIO CONTÉM O TEXTO 'PREFEITURA.SP'
        if "prefeitura.sp" in base_url:
            return "Prefeitura SP"
        #2 VERIFICA SE O DOMÍNIO CONTÉM O TEXTO 'AL.SP'
        elif "al.sp" in base_url:
            return "Alesp"
        #2 VERIFICA SE O DOMÍNIO CONTÉM O TEXTO 'LEISMUNICIPAIS.COM.BR'
        elif "leismunicipais.com.br" in base_url:
            return "Camara SP"
        #2 RETORNA 'OUTROS' SE NENHUMA CONDIÇÃO FOR ATENDIDA
        else:
            return "Outros"
    #2]

    #1[
    #1 ROTINA: _EXTRACT_SUBJECT
    #1 FINALIDADE: EXTRAI O ASSUNTO DA URL COM BASE NO NOME DA
INSTITUIÇÃO.
    #1 ENTRADAS: URL (STRING), NOME DA INSTITUIÇÃO (STRING)
    #1 DEPENDENCIAS: RE
    #1 CHAMADO POR: EXTRACT_URL_INFORMATION
    #1 CHAMA: NENHUMA
```

```
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _extract_subject
    def _extract_subject(self, url, institution_name):
        #2 EXTRAI O ASSUNTO PARA INSTITUIÇÕES DA ALESP
        if institution_name == "Alesp":
            match = re.search(r'palavraChaveDecode=([^&]+)&?', url)
            return match.group(1) if match else ""
        #2 EXTRAI O ASSUNTO PARA INSTITUIÇÕES DA PREFEITURA SP
        elif institution_name == "Prefeitura SP":
            match = re.search(r'assunto=([^&]+)', url)
            return match.group(1) if match else ""
        #2 EXTRAI O ASSUNTO PARA INSTITUIÇÕES DA CÂMARA SP
        elif institution_name == "Camara SP":
            match = re.search(r'q=([^&]+)&?', url)
            return match.group(1) if match else ""
        #2 RETORNA STRING VAZIA SE NÃO HOUVER MATCH
        return ""
    #2]

    #1[
    #1 ROTINA: EXECUTE
    #1 FINALIDADE: EXECUTA A ROTINA PRINCIPAL DE CARREGAMENTO E
PROCESSAMENTO DOS ARQUIVOS JSON.
    #1 ENTRADAS: NENHUMA
    #1 DEPENDENCIAS: LOGGINGHANDLER
    #1 CHAMADO POR: USUÁRIO
    #1 CHAMA: LOAD_JSON_FILES
    #1]
    #2[
    #2 PSEUDOCODIGO DE: execute
    @LoggingHandler.log_method("DataframeHandler", "execute", True)
    def execute(self):
        #2 INICIA O PROCESSO DE CARREGAMENTO E PROCESSAMENTO DOS ARQUIVOS
JSON
        self.load_json_files()
    #2]
```

Nome do Arquivo

leis_municipais_camara_sp.py

Documentação

```
#1[
#1 TITULO: LEISMUNICIPAISCAMARASP SCRAPPER
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 07/10/2024
#1 VERSAO: 1
#1 FINALIDADE: REALIZAR O SCRAPING DE DECRETOS NO SITE
LEISMUNICIPAIS.COM.BR REFERENTE À CÂMARA DE SÃO PAULO, COM BASE EM
DIVERSOS ASSUNTOS
#1 ENTRADAS: NOME DO DIRETÓRIO (NO CONSTRUTOR), ASSUNTO (NA FUNÇÃO
SCRAPE)
#1 SAIDAS: DADOS PROCESSADOS, LINKS EXTRAÍDOS, DECRETOS VISITADOS
#1 ROTINAS CHAMADAS: SCRAPE, BUILD_URL, PARSE_CONTENT, EXTRACT_LINKS,
SCRAPE_LINKS, SCRAPE_ALL_SUBJECTS
#1]

import time
from bs4 import BeautifulSoup
from concurrent.futures import ThreadPoolExecutor
from logging_logic.logging_handler import LoggingHandler
from scraping_logic.scraping_handler import ScrapingHandler

class LeisMunicipaisCamaraSpScrapper(ScrapingHandler):
    BASE_URL = "https://leismunicipais.com.br/camara/sp/sao-paulo"
    DECRETO_BASE_URL = "https://leismunicipais.com.br"
    SUBJECTS = ["Alteração", "Mudança", "Modificação", "Reforma",
"Transformação", "Troca", "Reajuste", "Revisão", "Ajuste", "Correção"]
    DIRECTORY_NAME = "LeisMunicipaisCamaraSp"
    SLEEP = 15
    MAX_ERROR = 3

    #1[
    #1 ROTINA: __INIT__
    #1 FINALIDADE: INICIALIZA A CLASSE LEISMUNICIPAISCAMARASP SCRAPPER E
CONFIGURA AS VARIÁVEIS INICIAIS E O DIRETÓRIO DE SALVAMENTO
    #1 ENTRADAS: NENHUMA
    #1 DEPENDENCIAS: LOGGINGHANDLER, SCRAPINGHANDLER
```

```
    #1 CHAMADO POR: LEISMUNICIPAISCAMARASP SCRAPPER
    #1 CHAMA: SCRAPINGHANDLER.__INIT__
    #1]
    #2[
    #2 PSEUDOCODIGO DE: __INIT__
    @LoggingHandler.log_method('LeisMunicipaisCamaraSpScrapper',
'__init__', show_parameters=False, show_output=False)
    def __init__(self):
        #2 CHAMA O CONSTRUTOR DA CLASSE PAI E CONFIGURA O DIRETÓRIO
        super().__init__(self.DIRECTORY_NAME)
        #2 INICIALIZA UM CONJUNTO PARA ARMAZENAR AS URLS JÁ VISITADAS
        self.visited_urls = set()
    #2]

    #1[
    #1 ROTINA: SCRAPE
    #1 FINALIDADE: REALIZAR O SCRAPING DE DECRETOS COM BASE NO ASSUNTO
ESPECIFICADO
    #1 ENTRADAS: ASSUNTO (STRING)
    #1 DEPENDENCIAS: TIME, REQUESTINGHANDLER
    #1 CHAMADO POR: USUÁRIO, SCRAPE_ALL_SUBJECTS
    #1 CHAMA: BUILD_URL, MAKE_REQUEST (REQUESTINGHANDLER),
PROCESS_RESPONSE, PARSE_CONTENT, EXTRACT_LINKS, SCRAPE_LINKS
    #1]
    #2[
    #2 PSEUDOCODIGO DE: SCRAPE
    @LoggingHandler.log_method('LeisMunicipaisCamaraSpScrapper',
'scrape', show_parameters=False, show_output=False)
    def scrape(self, subject):
        #2 DEFINE O NÚMERO DA PÁGINA E O CONTADOR DE ERROS
        page_number = 1
        error_count = 0
        #2 ENQUANTO NÃO ATINGIR O LIMITE DE ERROS, CONTINUA FAZENDO O
SCRAPING
        while error_count < self.MAX_ERROR:
            #2 CONSTROI A URL BASEADA NO ASSUNTO E NÚMERO DA PÁGINA
            url = self.build_url(subject, page_number)
            #2 FAZ A REQUISIÇÃO HTTP USANDO O HANDLER DE REQUISIÇÃO
            content = self.requesting_handler.make_request('GET', url)
            #2 PROCESSA A RESPOSTA RECEBIDA
            self.process_response(content)
            #2 SE NÃO HOUVER ERRO NA REQUISIÇÃO, PROSSEGUE COM O PARSE E
EXTRAÇÃO DE LINKS
            if not content.get('error'):
                #2 ANALISA O CONTEÚDO HTML RECEBIDO
                parsed_content =
self.parse_content(content['response_string'])
                #2 EXTRAI OS LINKS RELEVANTES DO CONTEÚDO HTML
                links = self.extract_links(parsed_content)
                #2 REALIZA O SCRAPING DOS LINKS ENCONTRADOS
                self.scrape_links(links)
                #2 REINICIA O CONTADOR DE ERROS
                error_count = 0
            else:
                #2 INCREMENTA O CONTADOR DE ERROS SE OCORRER UM ERRO NA
REQUISIÇÃO
                error_count += 1
            #2 AUMENTA O NÚMERO DA PÁGINA E AGUARDA UM TEMPO ANTES DE
CONTINUAR
            page_number += 1
            time.sleep(self.SLEEP)
    #2]

    #1[
    #1 ROTINA: BUILD_URL
    #1 FINALIDADE: CONSTRUIR A URL PARA REALIZAR A REQUISIÇÃO HTTP
BASEADO NO ASSUNTO E NO NÚMERO DA PÁGINA
    #1 ENTRADAS: ASSUNTO (STRING), NÚMERO DA PÁGINA (INT)
    #1 DEPENDENCIAS: NENHUMA
    #1 CHAMADO POR: SCRAPE
    #1 CHAMA: NENHUMA
    #1]
    #2[
    #2 PSEUDOCODIGO DE: BUILD_URL
    @LoggingHandler.log_method('LeisMunicipaisCamaraSpScrapper',
'build_url', show_parameters=False, show_output=False)
    def build_url(self, subject, page_number):
        #2 RETORNA A URL CONSTRUÍDA COM O ASSUNTO E NÚMERO DA PÁGINA PARA
A BUSCA
        return f"{self.BASE_URL}?q={subject}&page={page_number}"
    #2]

    #1[
```

```
    #1 ROTINA: PARSE_CONTENT
    #1 FINALIDADE: ANALISAR O CONTEÚDO HTML OBTIDO NA REQUISIÇÃO E
TRANSFORMÁ-LO EM UM OBJETO BEAUTIFULSOUP
    #1 ENTRADAS: CONTEÚDO HTML (STRING)
    #1 DEPENDENCIAS: BEAUTIFULSOUP
    #1 CHAMADO POR: SCRAPE
    #1 CHAMA: NENHUMA
    #1]
    #2[
    #2 PSEUDOCODIGO DE: PARSE_CONTENT
    @LoggingHandler.log_method('LeisMunicipaisCamaraSpScrapper',
'parse_content', show_parameters=False, show_output=False)
    def parse_content(self, content):
        #2 TRANSFORMA O CONTEÚDO HTML EM UM OBJETO BEAUTIFULSOUP PARA
FACILITAR A EXTRAÇÃO DE DADOS
        return BeautifulSoup(content, 'html.parser')
    #2]

    #1[
    #1 ROTINA: EXTRACT_LINKS
    #1 FINALIDADE: EXTRAIR LINKS RELEVANTES DO CONTEÚDO HTML ANALISADO
    #1 ENTRADAS: CONTEÚDO HTML ANALISADO (OBJETO BEAUTIFULSOUP)
    #1 DEPENDENCIAS: BEAUTIFULSOUP
    #1 CHAMADO POR: SCRAPE
    #1 CHAMA: NENHUMA
    #1]
    #2[
    #2 PSEUDOCODIGO DE: EXTRACT_LINKS
    @LoggingHandler.log_method('LeisMunicipaisCamaraSpScrapper',
'extract_links', show_parameters=False, show_output=False)
    def extract_links(self, parsed_content):
        #2 EXTRAI TODOS OS LINKS QUE CONTÊM 'DECRETO' NO CAMINHO DO HREF
        return [a['href'] for a in parsed_content.find_all('a',
href=True) if 'decreto' in a['href']]
    #2]

    #1[
    #1 ROTINA: SCRAPE_LINKS
    #1 FINALIDADE: REALIZAR O SCRAPING DE CADA LINK EXTRAÍDO, SEGUINDO OS
LINKS E PROCESSANDO AS RESPOSTAS
    #1 ENTRADAS: LISTA DE LINKS (LISTA DE STRINGS)
    #1 DEPENDENCIAS: TIME, REQUESTINGHANDLER
    #1 CHAMADO POR: SCRAPE
    #1 CHAMA: MAKE_REQUEST (REQUESTINGHANDLER), PROCESS_RESPONSE
    #1]
    #2[
    #2 PSEUDOCODIGO DE: SCRAPE_LINKS
    @LoggingHandler.log_method('LeisMunicipaisCamaraSpScrapper',
'scrape_links', show_parameters=False, show_output=False)
    def scrape_links(self, links):
        #2 ITERA SOBRE CADA LINK EXTRAÍDO
        for link in links:
            #2 CRIA A URL COMPLETA CONCATENANDO O LINK COM A
DECRETO_BASE_URL
            full_url = f"{self.DECRETO_BASE_URL}{link}"
            #2 VERIFICA SE A URL JÁ FOI VISITADA
            if full_url not in self.visited_urls:
                #2 ADICIONA A URL AO CONJUNTO DE URLs VISITADAS
                self.visited_urls.add(full_url)
                #2 FAZ A REQUISIÇÃO PARA A URL DO LINK
                content = self.requesting_handler.make_request('GET',
full_url)
                #2 PROCESSA A RESPOSTA RECEBIDA
                self.process_response(content)
                #2 AGUARDA O TEMPO CONFIGURADO ENTRE REQUISIÇÕES
                time.sleep(self.SLEEP)
    #2]

    #1[
    #1 ROTINA: SCRAPE_ALL_SUBJECTS
    #1 FINALIDADE: REALIZAR O SCRAPING DE TODOS OS ASSUNTOS DEFINIDOS
SIMULTANEAMENTE UTILIZANDO MÚLTIPLAS THREADS
    #1 ENTRADAS: NENHUMA (UTILIZA OS ASSUNTOS DA VARIÁVEL DE CLASSE
SUBJECTS)
    #1 DEPENDENCIAS: THREADPOOLEXECUTOR
    #1 CHAMADO POR: USUÁRIO
    #1 CHAMA: SCRAPE (PARA CADA ASSUNTO)
    #1]
    #2[
    #2 PSEUDOCODIGO DE: SCRAPE_ALL_SUBJECTS
    @LoggingHandler.log_method('LeisMunicipaisCamaraSpScrapper',
'scrape_all_subjects', show_parameters=False, show_output=False)
    def scrape_all_subjects(self):
```

```
        #2 UTILIZA UM THREADPOOLEXECUTOR PARA PARALELIZAR O PROCESSO DE
SCRAPING PARA CADA ASSUNTO EM SUBJECTS
        with ThreadPoolExecutor(max_workers=len(self.SUBJECTS)) as
executor:
            #2 EXECUTA A FUNÇÃO SCRAPE PARA CADA ASSUNTO DA LISTA
            executor.map(self.scrape, self.SUBJECTS)
    #2]
if __name__ == '__main__':
    scraper = LeisMunicipaisCamaraSpScrapper()
    scraper.scrape_all_subjects()
```

Nome do Arquivo
requesting_handler.py
Documentação

```
#1[
#1 TITULO: REQUESTINGHANDLER
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 07/10/2024
#1 VERSAO: 1
#1 FINALIDADE: REALIZAR REQUISIÇÕES HTTP (GET E POST) E LOGAR O TEMPO DE
RESPOSTA E OUTRAS INFORMAÇÕES RELEVANTES
#1 ENTRADAS: NOME DO DIRETÓRIO (NO CONSTRUTOR), MÉTODO HTTP, URL, KWARGS
(DADOS ADICIONAIS OPCIONAIS)
#1 SAIDAS: DICIONÁRIO CONTENDO DADOS DA REQUISIÇÃO E RESPOSTA, INCLUINDO
TEMPO, TAMANHO E ERROS (SE HOUVER)
#1 ROTINAS CHAMADAS: SET_METHOD_MAPPING, _REQUEST_WRAPPER, MAKE_REQUEST
#1]

import requests
import time
from datetime import datetime
from logging_logic.logging_handler import LoggingHandler

class RequestingHandler(LoggingHandler):
    #1[
    #1 ROTINA: __INIT__
    #1 FINALIDADE: INICIALIZA A CLASSE REQUESTINGHANDLER E CONFIGURA O
MAPEAMENTO DE MÉTODOS HTTP
    #1 ENTRADAS: NOME DO DIRETÓRIO (STRING)
    #1 DEPENDENCIAS: LOGGINGHANDLER
    #1 CHAMADO POR: REQUESTINGHANDLER
    #1 CHAMA: LOGGINGHANDLER.__INIT__, SET_METHOD_MAPPING
    #1]
    #2[
    #2 PSEUDOCODIGO DE: __init__
    def __init__(self, directory_name):
        #2 CHAMA O CONSTRUTOR DA CLASSE PAI PASSANDO O NOME DO DIRETÓRIO
        super().__init__(directory_name)
        #2 CONFIGURA O MAPEAMENTO DE MÉTODOS HTTP
        self.set_method_mapping()
    #2]

    #1[
    #1 ROTINA: SET_METHOD_MAPPING
    #1 FINALIDADE: DEFINE O MAPEAMENTO DOS MÉTODOS HTTP (GET E POST)
    #1 ENTRADAS: NENHUMA
    #1 DEPENDENCIAS: REQUESTS
    #1 CHAMADO POR: __INIT__
    #1 CHAMA: NENHUMA
    #1]
    #2[
    #2 PSEUDOCODIGO DE: set_method_mapping
    @LoggingHandler.log_method('RequestingHandler', 'set_method_mapping',
show_output=False, show_parameters=True)
    def set_method_mapping(self):
        #2 DEFINE O MAPEAMENTO DE MÉTODOS HTTP 'GET' E 'POST'
        self.method_mapping = {
            'GET': requests.get,
            'POST': requests.post
        }
    #2]

    #1[
    #1 ROTINA: _REQUEST_WRAPPER
    #1 FINALIDADE: ENVOLVE A EXECUÇÃO DA REQUISIÇÃO HTTP E COLETA DADOS
COMO TEMPO DE RESPOSTA E TAMANHO DA RESPOSTA
    #1 ENTRADAS: FUNÇÃO DO MÉTODO HTTP, URL, DICIONÁRIO REQUEST_INFO,
```

```
KWARGS OPCIONAIS
    #1 DEPENDENCIAS: TIME, REQUESTS
    #1 CHAMADO POR: MAKE_REQUEST
    #1 CHAMA: MÉTODOS HTTP (GET OU POST), RAISE_FOR_STATUS (REQUESTS)
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _request_wrapper
    def _request_wrapper(self, method_function, url, request_info,
**kwargs):
        try:
            #2 OBTÉM O TEMPO DE INÍCIO DA REQUISIÇÃO
            start_time = time.time()
            #2 EXECUTA O MÉTODO HTTP COM A URL E OS KWARGS
            response = method_function(url, **kwargs)
            #2 OBTÉM O TEMPO FINAL DA REQUISIÇÃO
            end_time = time.time()
            #2 CALCULA O TEMPO DE RESPOSTA
            request_info["response_time"] = end_time - start_time
            #2 OBTÉM O TAMANHO DO CONTEÚDO DA RESPOSTA
            request_info["response_length"] = len(response.content)
            #2 SALVA O TEXTO DA RESPOSTA
            request_info["response_string"] = response.text
            #2 VERIFICA SE HOUVE ALGUM ERRO NA REQUISIÇÃO
            response.raise_for_status()
        except requests.exceptions.RequestException as e:
            #2 EM CASO DE ERRO, ARMAZENA A MENSAGEM DE ERRO NO DICIONÁRIO
            request_info["error"] = str(e)
        #2 RETORNA AS INFORMAÇÕES DA REQUISIÇÃO
        return request_info
    #2]

    #1[
    #1 ROTINA: MAKE_REQUEST
    #1 FINALIDADE: FAZ A REQUISIÇÃO HTTP USANDO O MÉTODO ESPECIFICADO
(GET OU POST) E RETORNA AS INFORMAÇÕES DA REQUISIÇÃO
    #1 ENTRADAS: MÉTODO (GET OU POST), URL, KWARGS OPCIONAIS
    #1 DEPENDENCIAS: DATETIME, REQUESTS, TIME
    #1 CHAMADO POR: USUÁRIO
    #1 CHAMA: _REQUEST_WRAPPER
    #1]
    #2[
    #2 PSEUDOCODIGO DE: make_request
    @LoggingHandler.log_method('RequestingHandler', 'make_request',
show_output=False, show_parameters=True)
    def make_request(self, method, url, **kwargs):
        #2 VERIFICA SE O MÉTODO É SUPORTADO (GET OU POST)
        if method.upper() not in self.method_mapping:
            raise ValueError(f"Method {method} not supported. Use 'GET'
or 'POST'.")
        #2 CRIA UM DICIONÁRIO PARA ARMAZENAR AS INFORMAÇÕES DA REQUISIÇÃO
        request_info = {
            "url": url,
            "method": method.upper(),
            "date_time": datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
            "response_time": "",
            "response_length": "",
            "response_string": "",
            "error": ""
        }
        #2 CHAMA O MÉTODO _REQUEST_WRAPPER PARA REALIZAR A REQUISIÇÃO E
RETORNAR AS INFORMAÇÕES
        return self._request_wrapper(self.method_mapping[method.upper()],
url, request_info, **kwargs)
    #2]
```

Nome do Arquivo
legislacao_prefeitura_sp_gov_br.py
Documentação

```
#1[
#1 TITULO: LEGISLACAOPREFEITURASPGOVBR SCRAPPER
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 07/10/2024
#1 VERSAO: 1
#1 FINALIDADE: REALIZAR O SCRAPING DE LEGISLAÇÕES NO SITE DA PREFEITURA
DE SÃO PAULO COM BASE EM DIVERSOS ASSUNTOS
#1 ENTRADAS: NOME DO DIRETÓRIO (NO CONSTRUTOR), ASSUNTO (NA FUNÇÃO
SCRAPE)
```

```
#1 SAIDAS: DADOS PROCESSADOS, LINKS EXTRAÍDOS, LEGISLAÇÕES VISITADAS
#1 ROTINAS CHAMADAS: SCRAPE, BUILD_URL, PARSE_CONTENT, EXTRACT_LINKS,
SCRAPE_LINKS, SCRAPE_ALL_SUBJECTS
#1]

import time
from bs4 import BeautifulSoup
from concurrent.futures import ThreadPoolExecutor
from logging_logic.logging_handler import LoggingHandler
from scraping_logic.scraping_handler import ScrapingHandler

class LegislacaoPrefeituraSpGovBrScrapper(ScrapingHandler):
    BASE_URL = "https://legislacao.prefeitura.sp.gov.br"
    SUBJECTS = ["Alteração", "Mudança", "Modificação", "Reforma",
"Transformação", "Troca", "Reajuste", "Revisão", "Ajuste", "Correção"]
    DIRECTORY_NAME = "LegislacaoprefeituraSpGovBr"
    SLEEP = 15
    MAX_ERROR = 3

    #1[
    #1 ROTINA: __INIT__
    #1 FINALIDADE: INICIALIZA A CLASSE LEGISLACAOPREFEITURASPGOVBR
SCRAPPER E CONFIGURA AS VARIÁVEIS INICIAIS E O DIRETÓRIO DE SALVAMENTO
    #1 ENTRADAS: NENHUMA
    #1 DEPENDENCIAS: LOGGINGHANDLER, SCRAPINGHANDLER
    #1 CHAMADO POR: LEGISLACAOPREFEITURASPGOVBR SCRAPPER
    #1 CHAMA: SCRAPINGHANDLER.__INIT__
    #1]
    #2[
    #2 PSEUDOCODIGO DE: __INIT__
    @LoggingHandler.log_method('LegislacaoPrefeituraSpGovBrScrapper',
'__init__', show_parameters=False, show_output=False)
    def __init__(self):
        #2 CHAMA O CONSTRUTOR DA CLASSE PAI E CONFIGURA O DIRETÓRIO
        super().__init__(self.DIRECTORY_NAME)
        #2 INICIALIZA UM CONJUNTO PARA ARMAZENAR AS URLS JÁ VISITADAS
        self.visited_urls = set()
    #2]

    #1[
    #1 ROTINA: SCRAPE
    #1 FINALIDADE: REALIZAR O SCRAPING DE LEGISLAÇÕES COM BASE NO ASSUNTO
ESPECIFICADO
    #1 ENTRADAS: ASSUNTO (STRING)
    #1 DEPENDENCIAS: TIME, REQUESTINGHANDLER
    #1 CHAMADO POR: USUÁRIO, SCRAPE_ALL_SUBJECTS
    #1 CHAMA: BUILD_URL, MAKE_REQUEST (REQUESTINGHANDLER),
PROCESS_RESPONSE, PARSE_CONTENT, EXTRACT_LINKS, SCRAPE_LINKS
    #1]
    #2[
    #2 PSEUDOCODIGO DE: SCRAPE
    @LoggingHandler.log_method('LegislacaoPrefeituraSpGovBrScrapper',
'scrape', show_parameters=False, show_output=False)
    def scrape(self, subject):
        #2 DEFINE O NÚMERO DA PÁGINA E O CONTADOR DE ERROS
        page_number = 2
        error_count = 0
        #2 ENQUANTO NÃO ATINGIR O LIMITE DE ERROS, CONTINUA FAZENDO O
SCRAPING
        while error_count < self.MAX_ERROR:
            #2 CONSTROI A URL BASEADA NO ASSUNTO E NÚMERO DA PÁGINA
            url = self.build_url(subject, page_number)
            #2 FAZ A REQUISIÇÃO HTTP USANDO O HANDLER DE REQUISIÇÃO
            content = self.requesting_handler.make_request('GET', url)
            #2 PROCESSA A RESPOSTA RECEBIDA
            self.process_response(content)
            #2 SE NÃO HOUVER ERRO NA REQUISIÇÃO, PROSSEGUE COM O PARSE E
EXTRAÇÃO DE LINKS
            if not content.get('error'):
                #2 ANALISA O CONTEÚDO HTML RECEBIDO
                parsed_content =
self.parse_content(content['response_string'])
                #2 EXTRAI OS LINKS RELEVANTES DO CONTEÚDO HTML
                links = self.extract_links(parsed_content)
                #2 REALIZA O SCRAPING DOS LINKS ENCONTRADOS
                self.scrape_links(links)
                #2 REINICIA O CONTADOR DE ERROS
                error_count = 0
            else:
                #2 INCREMENTA O CONTADOR DE ERROS SE OCORRER UM ERRO NA
REQUISIÇÃO
                error_count += 1
            #2 AUMENTA O NÚMERO DA PÁGINA E AGUARDA UM TEMPO ANTES DE
```

```
CONTINUAR
            page_number += 1
            time.sleep(self.SLEEP)
    #2]

    #1[
    #1 ROTINA: BUILD_URL
    #1 FINALIDADE: CONSTRUIR A URL PARA REALIZAR A REQUISIÇÃO HTTP
BASEADO NO ASSUNTO E NO NÚMERO DA PÁGINA
    #1 ENTRADAS: ASSUNTO (STRING), NÚMERO DA PÁGINA (INT)
    #1 DEPENDENCIAS: NENHUMA
    #1 CHAMADO POR: SCRAPE
    #1 CHAMA: NENHUMA
    #1]
    #2[
    #2 PSEUDOCODIGO DE: BUILD_URL
    @LoggingHandler.log_method('LegislacaoPrefeituraSpGovBrScrapper',
'build_url', show_parameters=False, show_output=False)
    def build_url(self, subject, page_number):
        #2 RETORNA A URL CONSTRUÍDA COM O ASSUNTO E NÚMERO DA PÁGINA PARA
A BUSCA
        return f"{self.BASE_URL}/busca/pg/{page_number}?assunto=
{subject}"
    #2]

    #1[
    #1 ROTINA: PARSE_CONTENT
    #1 FINALIDADE: ANALISAR O CONTEÚDO HTML OBTIDO NA REQUISIÇÃO E
TRANSFORMÁ-LO EM UM OBJETO BEAUTIFULSOUP
    #1 ENTRADAS: CONTEÚDO HTML (STRING)
    #1 DEPENDENCIAS: BEAUTIFULSOUP
    #1 CHAMADO POR: SCRAPE
    #1 CHAMA: NENHUMA
    #1]
    #2[
    #2 PSEUDOCODIGO DE: PARSE_CONTENT
    @LoggingHandler.log_method('LegislacaoPrefeituraSpGovBrScrapper',
'parse_content', show_parameters=False, show_output=False)
    def parse_content(self, content):
        #2 TRANSFORMA O CONTEÚDO HTML EM UM OBJETO BEAUTIFULSOUP PARA
FACILITAR A EXTRAÇÃO DE DADOS
        return BeautifulSoup(content, 'html.parser')
    #2]

    #1[
    #1 ROTINA: EXTRACT_LINKS
    #1 FINALIDADE: EXTRAIR LINKS RELEVANTES DO CONTEÚDO HTML ANALISADO
    #1 ENTRADAS: CONTEÚDO HTML ANALISADO (OBJETO BEAUTIFULSOUP)
    #1 DEPENDENCIAS: BEAUTIFULSOUP
    #1 CHAMADO POR: SCRAPE
    #1 CHAMA: NENHUMA
    #1]
    #2[
    #2 PSEUDOCODIGO DE: EXTRACT_LINKS
    @LoggingHandler.log_method('LegislacaoPrefeituraSpGovBrScrapper',
'extract_links', show_parameters=False, show_output=False)
    def extract_links(self, parsed_content):
        #2 EXTRAI TODOS OS LINKS QUE CONTÊM 'LEIS' NO CAMINHO DO HREF
        return [a['href'] for a in parsed_content.find_all('a',
href=True) if 'leis' in a['href']]
    #2]

    #1[
    #1 ROTINA: SCRAPE_LINKS
    #1 FINALIDADE: REALIZAR O SCRAPING DE CADA LINK EXTRAÍDO, SEGUINDO OS
LINKS E PROCESSANDO AS RESPOSTAS
    #1 ENTRADAS: LISTA DE LINKS (LISTA DE STRINGS)
    #1 DEPENDENCIAS: TIME, REQUESTINGHANDLER
    #1 CHAMADO POR: SCRAPE
    #1 CHAMA: MAKE_REQUEST (REQUESTINGHANDLER), PROCESS_RESPONSE
    #1]
    #2[
    #2 PSEUDOCODIGO DE: SCRAPE_LINKS
    @LoggingHandler.log_method('LegislacaoPrefeituraSpGovBrScrapper',
'scrape_links', show_parameters=False, show_output=False)
    def scrape_links(self, links):
        #2 ITERA SOBRE CADA LINK EXTRAÍDO
        for link in links:
            #2 CRIA A URL COMPLETA CONCATENANDO O LINK COM A BASE_URL
            full_url = f"{self.BASE_URL}{link}"
            #2 VERIFICA SE A URL JÁ FOI VISITADA
            if full_url not in self.visited_urls:
                #2 ADICIONA A URL AO CONJUNTO DE URLs VISITADAS
```

```
                self.visited_urls.add(full_url)
                #2 FAZ A REQUISIÇÃO PARA A URL DO LINK
                content = self.requesting_handler.make_request('GET', 
full_url)
                #2 PROCESSA A RESPOSTA RECEBIDA
                self.process_response(content)
                #2 AGUARDA O TEMPO CONFIGURADO ENTRE REQUISIÇÕES
                time.sleep(self.SLEEP)
    #2]

    #1[
    #1 ROTINA: SCRAPE_ALL_SUBJECTS
    #1 FINALIDADE: REALIZAR O SCRAPING DE TODOS OS ASSUNTOS DEFINIDOS 
SIMULTANEAMENTE UTILIZANDO MÚLTIPLAS THREADS
    #1 ENTRADAS: NENHUMA (UTILIZA OS ASSUNTOS DA VARIÁVEL DE CLASSE 
SUBJECTS)
    #1 DEPENDENCIAS: THREADPOOLEXECUTOR
    #1 CHAMADO POR: USUÁRIO
    #1 CHAMA: SCRAPE (PARA CADA ASSUNTO)
    #1]
    #2[
    #2 PSEUDOCODIGO DE: SCRAPE_ALL_SUBJECTS
    @LoggingHandler.log_method('LegislacaoPrefeituraSpGovBrScrapper', 
'scrape_all_subjects', show_parameters=False, show_output=False)
    def scrape_all_subjects(self):
        #2 UTILIZA UM THREADPOOLEXECUTOR PARA PARALELIZAR O PROCESSO DE 
SCRAPING PARA CADA ASSUNTO EM SUBJECTS
        with ThreadPoolExecutor(max_workers=len(self.SUBJECTS)) as 
executor:
            #2 EXECUTA A FUNÇÃO SCRAPE PARA CADA ASSUNTO DA LISTA
            executor.map(self.scrape, self.SUBJECTS)
    #2]
if __name__ == '__main__':
    scraper = LegislacaoPrefeituraSpGovBrScrapper()
    scraper.scrape_all_subjects()
```

Nome do Arquivo
logs.txt
Documentação

```
[+] DataframeHandler.execute
[+] DataframeHandler.load_json_files
[+] DataframeHandler.execute
[+] DataframeHandler.load_json_files
[+] DataframeHandler.extract_url_information
[+] DataframeHandler.extract_url_information
[+] DataframeHandler.extract_url_information
[+] DataframeHandler.execute Output: None
[+] DataframeHandler.execute
[+] DataframeHandler.load_json_files
[+] DataframeHandler.extract_url_information
[+] DataframeHandler.extract_url_information
[+] DataframeHandler.extract_url_information
[+] DataframeHandler.execute Output: None
[+] DataframeHandler.execute
[+] DataframeHandler.load_json_files
[+] DataframeHandler.extract_url_information
[+] DataframeHandler.extract_url_information
[+] DataframeHandler.extract_url_information
[+] DataframeHandler.execute Output: None
[+] DataframeHandler.execute
[+] DataframeHandler.load_json_files
[+] DataframeHandler.execute
[+] DataframeHandler.load_json_files
[+] DataframeHandler.extract_url_information
[+] DataframeHandler.extract_url_information
[+] DataframeHandler.execute
[+] DataframeHandler.load_json_files
[+] DataframeHandler.extract_url_information
[+] DataframeHandler.extract_url_information
[+] DataframeHandler.extract_url_information
[+] DataframeHandler.execute Output: None
```

Nome do Arquivo

text_handler.py

Documentação

```
#1[
#1 TITULO: TEXTHANDLER
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 07/10/2024
#1 VERSAO: 1
#1 FINALIDADE: REALIZAR EXTRAÇÃO E FILTRAGEM DE TEXTO BRUTO A PARTIR DE
CONTEÚDO HTML
#1 ENTRADAS: NOME DO DIRETÓRIO (NO CONSTRUTOR), CONTEÚDO HTML (NA FUNÇÃO
EXTRACT_RAW_TEXT)
#1 SAIDAS: TEXTO LIMPO EM MINÚSCULAS EXTRAÍDO DO CONTEÚDO HTML
#1 ROTINAS CHAMADAS: EXTRACT_RAW_TEXT, _FILTER_PORTUGUESE_TEXT
#1]

import re
from bs4 import BeautifulSoup
from logging_logic.logging_handler import LoggingHandler

class TextHandler(LoggingHandler):
    #1[
    #1 ROTINA: __INIT__
    #1 FINALIDADE: INICIALIZA A CLASSE TEXTHANDLER E CONFIGURA O
DIRETÓRIO
    #1 ENTRADAS: NOME DO DIRETÓRIO (STRING)
    #1 DEPENDENCIAS: LOGGINGHANDLER
    #1 CHAMADO POR: TEXTHANDLER
    #1 CHAMA: LOGGINGHANDLER.__INIT__
    #1]
    #2[
    #2 PSEUDOCODIGO DE: __init__
    def __init__(self, directory_name):
        #2 CHAMA O CONSTRUTOR DA CLASSE PAI E INICIALIZA O DIRETÓRIO
        super().__init__(directory_name)
        #2 ARMAZENA O NOME DO DIRETÓRIO
        self.directory_name = directory_name
    #2]

    #1[
    #1 ROTINA: EXTRACT_RAW_TEXT
    #1 FINALIDADE: EXTRAI O TEXTO BRUTO DO CONTEÚDO HTML E O FILTRA PARA
REMOVER CARACTERES INDESEJADOS
    #1 ENTRADAS: CONTEÚDO HTML (STRING)
    #1 DEPENDENCIAS: BEAUTIFULSOUP, RE
    #1 CHAMADO POR: USUÁRIO
    #1 CHAMA: _FILTER_PORTUGUESE_TEXT
    #1]
    #2[
    #2 PSEUDOCODIGO DE: extract_raw_text
    def extract_raw_text(self, html_content):
        try:
            #2 ANALISA O CONTEÚDO HTML USANDO BEAUTIFULSOUP
            soup = BeautifulSoup(html_content, 'html.parser')
            #2 EXTRAI O TEXTO BRUTO DO CONTEÚDO HTML
            raw_text = soup.get_text(separator=' ', strip=True)
            #2 FILTRA O TEXTO PARA REMOVER CARACTERES NÃO PERTENCENTES AO
PORTUGUÊS
            clean_text = self._filter_portuguese_text(raw_text)
            #2 RETORNA O TEXTO FILTRADO EM MINÚSCULAS
            return clean_text.lower()
        except Exception as e:
            #2 RETORNA UMA STRING VAZIA EM CASO DE FALHA
            return ""
    #2]

    #1[
    #1 ROTINA: _FILTER_PORTUGUESE_TEXT
    #1 FINALIDADE: FILTRA O TEXTO PARA REMOVER CARACTERES QUE NÃO SÃO DO
PORTUGUÊS
    #1 ENTRADAS: TEXTO BRUTO (STRING)
    #1 DEPENDENCIAS: RE
    #1 CHAMADO POR: EXTRACT_RAW_TEXT
    #1 CHAMA: NENHUMA
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _filter_portuguese_text
    def _filter_portuguese_text(self, text):
        #2 DEFINE UM PADRÃO PARA MANTER APENAS CARACTERES PORTUGUESES,
NÚMEROS E PONTUAÇÃO
        portuguese_pattern = re.compile(r"[^a-zA-ZÀ-ÿ0-9.,!?;:\-\(\)\
[\]\s]")
```

```
        #2 APLICA O PADRÃO PARA REMOVER CARACTERES INDESEJADOS
        filtered_text = portuguese_pattern.sub('', text)
        #2 RETORNA O TEXTO FILTRADO
        return filtered_text
    #2]
```

Nome do Arquivo

saving_handler.py

Documentação

```
#1[
#1 TITULO: SAVINGHANDLER
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 07/10/2024
#1 VERSAO: 1
#1 FINALIDADE: GERENCIAR A CRIAÇÃO DE DIRETÓRIOS E O SALVAMENTO DE
INFORMAÇÕES DE REQUISIÇÕES EM ARQUIVOS JSON
#1 ENTRADAS: NOME DO DIRETÓRIO (NO CONSTRUTOR), DICIONÁRIO REQUEST_INFO
(NA FUNÇÃO SAVE_REQUEST_INFO)
#1 SAIDAS: CAMINHO DO ARQUIVO JSON GERADO COM AS INFORMAÇÕES DA
REQUISIÇÃO
#1 ROTINAS CHAMADAS: _CREATE_DIRECTORY, SAVE_REQUEST_INFO
#1]

import os
import json
from datetime import datetime
from logging_logic.logging_handler import LoggingHandler

class SavingHandler(LoggingHandler):
    #1[
    #1 ROTINA: __INIT__
    #1 FINALIDADE: INICIALIZA A CLASSE SAVINGHANDLER E CRIA O DIRETÓRIO
DE SALVAMENTO SE NÃO EXISTIR
    #1 ENTRADAS: NOME DO DIRETÓRIO (STRING)
    #1 DEPENDENCIAS: LOGGINGHANDLER, OS
    #1 CHAMADO POR: SAVINGHANDLER
    #1 CHAMA: LOGGINGHANDLER.__INIT__, _CREATE_DIRECTORY
    #1]
    #2[
    #2 PSEUDOCODIGO DE: __init__
    @LoggingHandler.log_method('SavingHandler', '__init__')
    def __init__(self, directory_name):
        #2 CHAMA O CONSTRUTOR DA CLASSE PAI E INICIALIZA O DIRETÓRIO
        super().__init__(directory_name)
        #2 ARMAZENA O NOME DO DIRETÓRIO
        self.directory_name = directory_name
        #2 CHAMA O MÉTODO PARA CRIAR O DIRETÓRIO SE ELE NÃO EXISTIR
        self._create_directory()
    #2]

    #1[
    #1 ROTINA: _CREATE_DIRECTORY
    #1 FINALIDADE: VERIFICA SE O DIRETÓRIO EXISTE E O CRIA SE NECESSÁRIO
    #1 ENTRADAS: NENHUMA
    #1 DEPENDENCIAS: OS
    #1 CHAMADO POR: __INIT__
    #1 CHAMA: OS.MAKEDIRS
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _create_directory
    def _create_directory(self):
        #2 VERIFICA SE O DIRETÓRIO EXISTE
        if not os.path.exists(self.directory_name):
            #2 CRIA O DIRETÓRIO SE ELE NÃO EXISTIR
            os.makedirs(self.directory_name)
    #2]

    #1[
    #1 ROTINA: SAVE_REQUEST_INFO
    #1 FINALIDADE: SALVA AS INFORMAÇÕES DA REQUISIÇÃO EM UM ARQUIVO JSON
DENTRO DO DIRETÓRIO ESPECIFICADO
    #1 ENTRADAS: DICIONÁRIO REQUEST_INFO (CONTENDO DADOS DA REQUISIÇÃO)
    #1 DEPENDENCIAS: JSON, OS, DATETIME
    #1 CHAMADO POR: USUÁRIO
    #1 CHAMA: NENHUMA
    #1]
```

```
    #2[
    #2 PSEUDOCODIGO DE: save_request_info
    @LoggingHandler.log_method('SavingHandler', 'save_request_info',
show_output=True)
    def save_request_info(self, request_info):
        #2 GERA UM TIMESTAMP PARA O NOME DO ARQUIVO
        timestamp = datetime.now().strftime('%Y%m%d%H%M%S%f')
        #2 DEFINE O NOME DO ARQUIVO JSON COM BASE NO TIMESTAMP
        file_name = f"request_{timestamp}.json"
        #2 DEFINE O CAMINHO COMPLETO DO ARQUIVO A SER SALVO
        file_path = os.path.join(self.directory_name, file_name)
        #2 ABRE O ARQUIVO JSON EM MODO DE ESCRITA E SALVA O DICIONÁRIO
REQUEST_INFO
        with open(file_path, 'w') as json_file:
            json.dump(request_info, json_file, indent=4)
        #2 RETORNA O CAMINHO DO ARQUIVO SALVO
        return file_path
    #2]
```

Nome do Arquivo

scraping_handler.py

Documentação

```
#1[
#1 TITULO: SCRAPINGHANDLER
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 07/10/2024
#1 VERSAO: 1
#1 FINALIDADE: REALIZAR O PROCESSO DE SCRAPING, EXTRAÇÃO DE TEXTO,
LEMATIZAÇÃO E SALVAMENTO DOS DADOS EM ARQUIVOS JSON
#1 ENTRADAS: NOME DO DIRETÓRIO (NO CONSTRUTOR), DICIONÁRIO DATA_DICT (NA
FUNÇÃO PROCESS_RESPONSE)
#1 SAIDAS: DICIONÁRIO PROCESSADO COM CAMPOS EXTRAÍDOS E ARQUIVO JSON COM
AS INFORMAÇÕES SALVAS
#1 ROTINAS CHAMADAS: PROCESS_RESPONSE, SAVE_REQUEST_INFO (SAVINGHANDLER),
EXTRACT_RAW_TEXT (TEXTHANDLER), LEMMATIZE_AND_EXTRACT_TOPONYMS
(TOPONYMHANDLER)
#1]

from logging_logic.logging_handler import LoggingHandler
from requesting_logic.requesting_handler import RequestingHandler
from saving_logic.saving_handler import SavingHandler
from text_logic.text_handler import TextHandler
from toponym_logic.toponym_handler import ToponymHandler

class ScrapingHandler(LoggingHandler):
    #1[
    #1 ROTINA: __INIT__
    #1 FINALIDADE: INICIALIZA A CLASSE SCRAPINGHANDLER E CONFIGURA AS
DEPENDÊNCIAS NECESSÁRIAS
    #1 ENTRADAS: NOME DO DIRETÓRIO (STRING)
    #1 DEPENDENCIAS: REQUESTINGHANDLER, SAVINGHANDLER, TEXTHANDLER,
TOPONYMHANDLER, LOGGINGHANDLER
    #1 CHAMADO POR: SCRAPINGHANDLER
    #1 CHAMA: LOGGINGHANDLER.__INIT__, REQUESTINGHANDLER, SAVINGHANDLER,
TEXTHANDLER, TOPONYMHANDLER
    #1]
    #2[
    #2 PSEUDOCODIGO DE: __init__
    def __init__(self, directory_name):
        #2 CHAMA O CONSTRUTOR DA CLASSE PAI E INICIALIZA O DIRETÓRIO
        super().__init__(directory_name)
        #2 INICIALIZA O HANDLER DE REQUISIÇÕES
        self.requesting_handler = RequestingHandler(directory_name)
        #2 INICIALIZA O HANDLER DE SALVAMENTO
        self.saving_handler = SavingHandler(directory_name)
        #2 INICIALIZA O HANDLER DE TEXTO
        self.text_handler = TextHandler(directory_name)
        #2 INICIALIZA O HANDLER DE TOPÔNIMOS
        self.toponym_handler = ToponymHandler(directory_name)
    #2]

    #1[
    #1 ROTINA: PROCESS_RESPONSE
    #1 FINALIDADE: PROCESSA O DICIONÁRIO DE RESPOSTA, EXTRAINDO TEXTO
BRUTO, LEMATIZANDO E EXTRAINDO TOPÔNIMOS, E SALVA AS INFORMAÇÕES
    #1 ENTRADAS: DICIONÁRIO DATA_DICT (COM CAMPOS COMO RESPONSE_STRING)
    #1 DEPENDENCIAS: TEXTHANDLER, TOPONYMHANDLER, SAVINGHANDLER
```

```
    #1 CHAMADO POR: USUÁRIO
    #1 CHAMA: EXTRACT_RAW_TEXT (TEXTHANDLER),
LEMMATIZE_AND_EXTRACT_TOPONYMS (TOPONYMHANDLER), SAVE_REQUEST_INFO
(SAVINGHANDLER)
    #1]
    #2[
    #2 PSEUDOCODIGO DE: process_response
    @LoggingHandler.log_method("ScrapingHandler", "process_response")
    def process_response(self, data_dict):
        #2 EXTRAI O TEXTO BRUTO A PARTIR DA RESPOSTA
        extracted_text =
self.text_handler.extract_raw_text(data_dict['response_string'])
        #2 ADICIONA O TEXTO EXTRAÍDO AO DICIONÁRIO
        data_dict['extracted_text'] = extracted_text
        #2 LEMATIZA O TEXTO E EXTRAI OS TOPÔNIMOS
        lemmatized_text, extracted_toponyms =
self.toponym_handler.lemmatize_and_extract_toponyms(extracted_text)
        #2 ADICIONA O TEXTO LEMATIZADO E OS TOPÔNIMOS AO DICIONÁRIO
        data_dict['lemmatized_text'] = lemmatized_text
        data_dict['extracted_toponyms'] = extracted_toponyms
        #2 SALVA O DICIONÁRIO PROCESSADO USANDO O SAVINGHANDLER
        self.saving_handler.save_request_info(data_dict)
    #2]
```