Marcin Świątkowski

Dokumentacja modułu sejmowego

Temat: Moduł serwera MQTT pozwalający na uzyskiwanie informacji na temat aktualnego stanu członkowskiego sejmu.

SPIS TREŚCI:

Ogólny opis

Zaprezentowany moduł oferuje wgląd w aktualny stan członkowski i partyjny polskiego sejmu. Za pomocą zapytań można uzyskać informacje dotyczące nazwisk posłów oraz osób pełniących ważne państwowe funkcje, liczby posłów czy danych dotyczących partii politycznych. Moduł pozwala na wykonywanie zapytań na kilka różnych sposobów, uniewrażliwiając się do pewnego stopnia na niepoprawność składniową zapytań. Moduł korzysta z bazy danych składającej się z trzech plików csv i zawierających dane personalne posłów, partii politycznych oraz osób pełniących ważniejsze funkcje, które, choć niekoniecznie mieszczące się w ramach sejmu, mogą być przedmiotem zapytań ze strony użytkownika. Moduł przetwarza zapytania składając je w wysokopoziomowe odpowiedzi, przyjazne w odbiorze dla użytkownika. Moduł napisany jest w paradygmacie funkcyjnym. Z elementów obiektowości korzysta na etapie podłączenia do serwera MQTT. Serwer MQTT, z którego moduł korzysta to serwer “Voice Assistant” napisany przez Wojciecha Węgrzynka. Zarówno moduł jak i serwer powstały w ramach zajęć projektowych z “Języków i bibliotek analizy danych” prowadzonych przez dr. Marka Gajęckiego na Akademii Górniczo-Hutniczej w Krakowie.

Dane, z których korzysta moduł zostały pozyskane z rządowej strony sejmu Rzeczypospolitej Polskiej[1] [2]. Moduł korzysta z trzech plików:

- *sejm.csv*, który zawiera dane wszystkich polskich posłów i posłanek (nazwisko, imię, przynależność, stanowisko - jeśli jest, płeć),
- *politicians.csv*, który zawiera informacje o osobach pełniących ważne funkcje państwowe, nie wchodzące w skład sejmu, ale mogące być użyteczne dla użytkownika,
- *clubs.csv*, który zawiera informacje na temat ugrupowań, które znajdują się w sejmie.

Oprócz tego moduł zapisuje logi niezależnie do odrębnego pliku tekstowego o nazwie *logs.txt*.

[1] https://www.sejm.gov.pl/sejm9.nsf/poslowie.xsp?type=A
[2] https://www.sejm.gov.pl/sejm9.nsf/page.xsp/poslowie_obecnie

Przewodnik użytkownika

Po dodaniu modułu do serwera, użytkownik musi połączyć się z serwerem przez broker mqtt. Po wydaniu zapytania moduł odpowie na nie jeśli będzie ono skierowane do niego i poprawnie sformułowane. Jeśli chodzi o poprawność formułowania zapytań, to poniżej zaprezentowane są wszystkie słowa, które moduł potrafi zrozumieć, poprzedzone odpowiednimi **słowami kluczowymi**:

kto - kto, któż
ile - ile, iluż
ilu - ilu, iluż
w - w
jest - jest, jestże
prezydent - prezydent, prezydentem, prezydentów
posel - poseł, posłem, posłów
premier - premier, premierem, premierów
leader - przewodniczący, przewodniczącym, lider, liderem, szef, szefem
rpo - rpo, rzecznik, rzecznikiem
k - kobieta, kobiet, kobiety
m - mężczyzna, mężczyzn, mężczyźni
pis - pis, pisu, pisowi, prawa, prawo, prawie, prawu
ko - ko, koalicja, koalicji
lewica - lewica, lewicy
kp - polska, polskiej, kape
konfederacja - konfederacja, konfederacji, konfa, konfie, konfy
polska2050 - 2050, pięćdziesiąt
porozumienie - porozumienie, porozumieniu, porozumienia
kukiz15 - kukiz, piętnaście
ps - ps, pees, sprawy, spraw, sprawach, polskich
pps - pps, pepees
niezrzeszony - niezrzeszony, niezrzeszeni, niezrzeszonych
partia - partia, partii, ugrupowanie, ugrupowaniu, klub, klubie, klubu

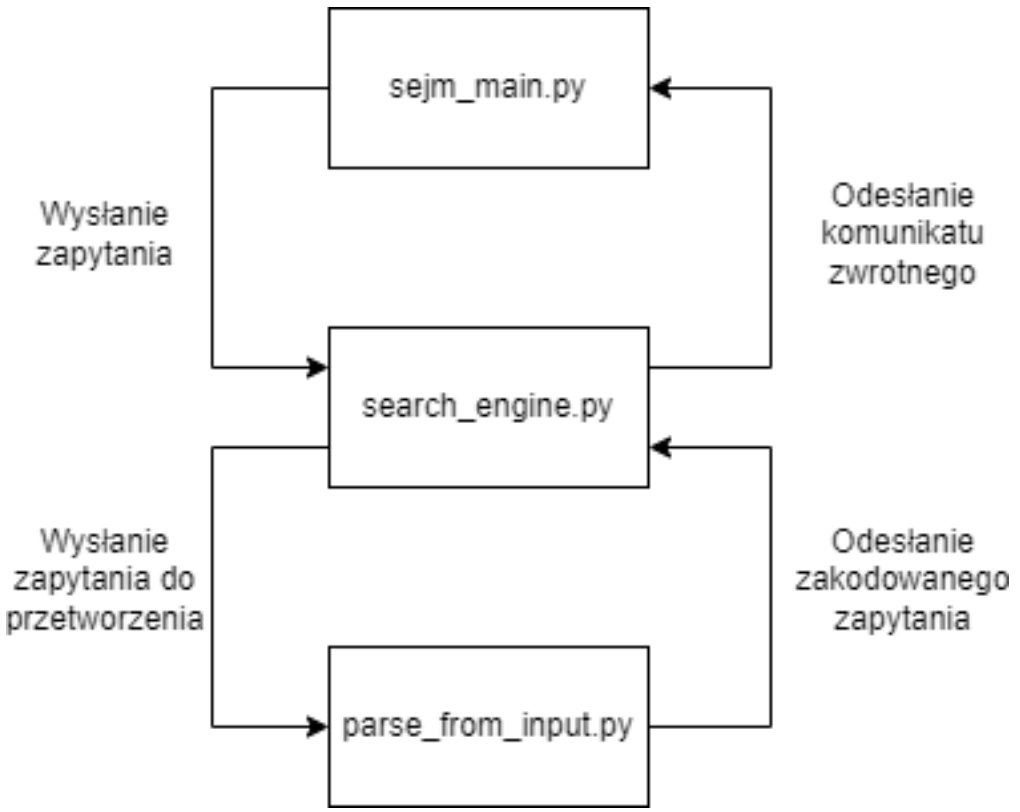
Na ich podstawie program decyduje jak przetworzyć zapytanie i których funkcji użyć.

Pełny opis systemu

Moduł wybiera sposób przetworzenia zapytania na podstawie słów kluczowych. Najprościej można opisać jego działanie w etapach:

1. Pobranie zapytania.
2. Przetworzenie zapytania na postać zakodowaną.
3. Wywołanie funkcji w oparciu o obecność słów kluczowych.
4. Przeszukiwanie bazy danych i formułowanie komunikatu.
5. Zwrócenie komunikatu.

Jeśli chodzi o relacje między trzema plikami, z których składa się program, to wyglądają one następująco:

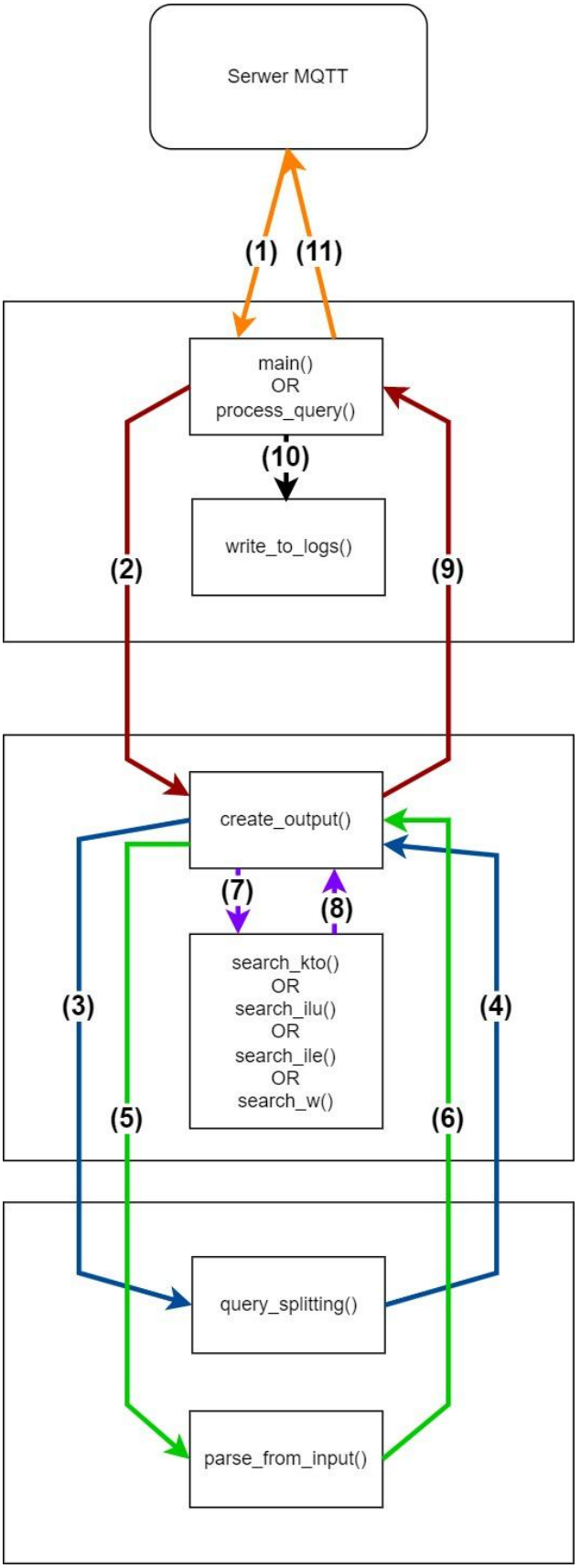


Jeśli chcielibyśmy rozpisać to w oparciu o wywoływane funkcje, to wyglądałoby następująco:

sejm_main.py
lub
sejm_main_mqtt.py

search_engine.py

parse_from_input.py



Szczegółowy opis etapów z diagramu

1. Pobranie zapytania z serwera.
2. Wywołanie funkcji create_output() z zapytaniem jako argumentem.
3. Wywołanie funkcji query_splitting() z zapytaniem jako argumentem.
4. Zwrócenie zapytania przekształconego z łańcucha znaków na listę.
5. Wywołanie funkcji parse_from_input() z otrzymaną listą jako argumentem.
6. Zwrócenie listy będącej zakodowanym zapytaniem.
7. Wywołanie odpowiedniej funkcji tworzącej odpowiedź na zapytanie. Wybór funkcji odbywa się na podstawie zawartości listy będącej zakodowanym zapytaniem.
8. Zwrócenie przez wybraną funkcję odpowiedzi na zapytanie.
9. Zwrócenie przez funkcję create_output() odpowiedzi na zapytanie.
10. Zapisanie zapytania i odpowiedzi do logów.
11. Zwrócenie przez moduł odpowiedzi do serwera.

Funkcje modułu

Moduł składa się z następujących funkcji:

1. *main()* lub *process_query():*
   Funkcja pozwalająca przetworzyć zapytanie.
   :return: <string> odpowiedź na zapytanie użytkownika.

2. *write_to_logs():*
   Funkcja służąca do zapisywania zapytań i odpowiedzi w pliku logs.txt.
   :param query: <string> zapytanie, które pobierane jest z serwera mqtt (lub, w przypadku tego programu testowego, z inputu użytkownika).
   :param output: <string> łańcuch znaków, który wysyłany jest do serwera. Stanowi odpowiedź na zapytanie.
   :return: <None>.

3. *create_output():*
   Funkcja, która wywołuje wszystkie powyższe funkcje w zależności od zapytania. Jest wywoływana w metodzie main.

:param user_query: <string> Niezakodowane i niepodzielone zapytanie użytkownika.
:return: <string> fraza będąca odpowiedzią na zapytanie.

4. *search_kto():*
   Funkcja odpowiadająca na pytanie o to, kto pełni daną funkcję.
   :param keywords: <list> zakodowane zapytanie użytkownika.
   :return: <string> wykonanie funkcji search_politicians().

5. *search_ilu():*
   Funkcja obsługująca zapytania o liczebność polityków.
   :param keywords: <list> zakodowane zapytanie użytkownika.
   :return: <string> fraza będąca odpowiedzią na zapytanie.

6. *search_ile():*
   Rozszerzenie gramatyczne funkcji liczącej polityków. Ta funkcja liczy tylko kobiety.
   :param keywords: <list> zakodowane zapytanie użytkownika.
   :return: <string> fraza będąca odpowiedzią na zapytanie.

7. *search_w():*
   Funkcja obsługująca zapytania zaczynające się na "w". Głównie dotyczące przynależności kogoś do partii.
   :param keywords: <list> zakodowane zapytanie użytkownika.
   :param split_query: <list> niezakodowane, ale podzielone na odrębne słowa zapytanie użytkownika.
   :return: <string> fraza będąca odpowiedzią na zapytanie.

8. *search_politicians():*
   Funkcja przeszukująca polityków pod względem funkcji.
   :param keyword: <string> pojedynczy element zakodowanego zapytania.
   :return: <string> fraza będąca odpowiedzią na zapytanie.

9. *count_sejm():*

   Funkcja licząca członków sejmu. Oba argumenty opcjonalne.

   :param keyword: <string> pojedynczy element zakodowanego zapytania oznaczający płeć.

   :param party: <string> pojedynczy element zakodowanego zapytania oznaczający partię.

   :return: <int> liczba naturalna wyrażająca liczbę osób w sejmie.

10. *search_clubs():*

    Funkcja przeszukująca kluby i koła, jakie znajdują się w sejmie.

    :param keywords: <list> zakodowane zapytanie użytkownika.

    :return: <list> wiersz z pliku clubs.csv.

11. *query_splitting():*

    Funkcja rozbijająca zapytanie na listę wchodzących w jego skład słów.

    :param user_query: <string> zapytanie użytkownika.

    :return: <list> zapytanie użytkownika w formie listy poszczególnych słów.

12. *parse_from_input():*

    Funkcja kodująca rozbite zapytanie użytkownika na listę kodów odpowiadających słowom z zapytania. Tworzy listę, która będzie użyta w dalszym przetwarzaniu.

    :param split_query: <list> zapytanie użytkownika w formie listy poszczególnych słów.

    :return: <list> zakodowane zapytanie użytkownika.

Instalacja

Modułu można używać na dwa sposoby. Plik sejm_main.py jest plikiem pozwalającym przetestować program bez używania brokera MQTT. W celu jego uruchomienia wystarczy uruchomić właśnie program z pliku sejm_main.py w terminalu z poziomu katalogu, w którym się on znajduje. W ten sposób możliwe będzie wprowadzanie zapytań tekstowo.

W przypadku uruchomienia wraz z serwerem MQTT należy:

1. Pobrać serwer Voice Assistant z tej strony: https://github.com/TETRX/VoiceAssistant
2. Przekleić katalog sejm_module do katalogu voice_assistant_modules w plikach serwera Voice Assistant.
3. Dalej należy postępować zgodnie z instrukcją obsługi serwera Voice Assistant.

Podsumowanie

Największym wyzwaniem przy tworzeniu modułu było takie przełożenie zapytania użytkownika, by program był w stanie je zrozumieć i na nie odpowiedzieć. Pula możliwych zapytań jest zapewne większa niż słowa, które zostały uwzględnione, niemniej program radzi sobie ze zrozumieniem użytkownika. Trudność polegającą na tym rozumieniu można było rozwiązać na dwa sposoby: rozszerzając pulę możliwych zapytań, szykując się na każdą ewentualność albo spróbować zaprojektować szkielet rozumienia zapytania. W mojej pracy nad modułem postarałem się pójść tą drugą drogą, nawet jeśli nie wykorzystując całego potencjału tej metody, to eksplorując możliwe rozwiązania w zakresie jej praktycznego zastosowania.

Przykładowe wpisy z logów

Zapytanie: kto jest prezydentem
Odpowiedź: prezydent polski to andrzej duda

Zapytanie: kto jest rzecznikiem
Odpowiedź: rzecznik praw obywatelskich polski to marcin wiącek

Zapytanie: ilu posłów ma konfederacja
Odpowiedź: konfederacja ma w sejmie 11 posłów

Zapytanie: ilu jest posłów niezrzeszonych
Odpowiedź: Posłów niezrzeszonych jest w sejmie 3

Zapytanie: ile jest kobiet w sejmie
Odpowiedź: kobiet jest w sejmie 130

Zapytanie: ilu jest mężczyzn w sejmie
Odpowiedź: mężczyzn jest w sejmie 329

Zapytanie: ilu jest posłów
Odpowiedź: posłów w sejmie jest 459

Zapytanie: ilu jest posłów prawa i sprawiedliwości
Odpowiedź: prawo i sprawiedliwość ma w sejmie 227 posłów

Zapytanie: ilu jest mężczyzn w partii prawo i sprawiedliwość
Odpowiedź: mężczyzn w ugrupowaniu prawo i sprawiedliwość jest 173

Zapytanie: w jakiej partii jest jarosław
Odpowiedź: Więcej niż jedna osoba ma tak na imię

Zapytanie: w której partii jest jarosław kaczyński
Odpowiedź: jarosław kaczyński jest w ugrupowaniu prawo i sprawiedliwość

Zapytanie: w jakiejże partii jest tomasz
Odpowiedź: Więcej niż jedna osoba ma tak na imię

Zapytanie: w którejże partii jest schetyna grzegorz
Odpowiedź: grzegorz schetyna jest w ugrupowaniu koalicja obywatelska

Zapytanie: w jakim ugrupowaniu jest gonciarz
Odpowiedź: gonciarz jest w partii prawo i sprawiedliwość

Zapytanie: kto jest liderem prawa i sprawiedliwości
Odpowiedź: przewodniczącym ugrupowania sejmowego prawo i sprawiedliwość jest ryszard terlecki

Zapytanie: kto jest szefem koalicji obywatelskiej
Odpowiedź: przewodniczącym ugrupowania sejmowego koalicja obywatelska jest borys budka

Zapytanie: ilu posłów jest w lewicy
Odpowiedź: lewica ma w sejmie 44 posłów

Zapytanie: ile kobiet jest w lewicy
Odpowiedź: kobiet w ugrupowaniu lewica jest 18

Zapytanie: w którym ugrupowaniu jest gawkowski
Odpowiedź: gawkowski jest w partii lewica

Zapytanie: kto jest szefem
Odpowiedź: Nie rozumiem

Kod źródłowy

========== plik sejm_main.py ==========

```
# coding=utf8
import search_engine as seng


"""
##################################################################
###################################
Ten plik zawiera funkcję main, która działa bez serwera mqtt.
Funkcja istnieje dla potrzeb
testowania i może być uruchamian w przypadku problemów z
podłączeniem modułu do serwera.
Plik zawiera dwie funkcje. Funkcja write_to_logs() służy do
zapisywania zapytań oraz
odpowiedzi wraz z datą i godziną w logach. Funkcja main() to
funkcja uruchamiająca cały program.

##################################################################
###################################
"""



def write_to_logs(query, output):
    """
      Funkjca służąca do zapisywania zapytań i odpowiedzi w
pliku logs.txt.
     :param query: zapytanie, które pobierane jest z serwera
mqtt (lub, w przypadku tego programy testowego,
    z inputu użytkownika.
     :param output: string, który wysyłany jest do serwera.
Stanowi odpowiedź na zapytanie.
    :return: None.
    """
    from datetime import datetime, date
    now = datetime.now()
    current_time = now.strftime("%H:%M:%S")
    today = date.today()
    with open("logs.txt", "a") as log:
                log.write(f"{str(today)}  {str(current_time)}
\nZapytanie: {query} \nOdpowiedź: {output}\n \n \n")


def main():
    """
    Funkcja uruchamiająca program.
```

```
    :return: odpowiedź na zapytanie użytkownika.
    """
      user_query = input("usr_query>>> ").lower()   # Pobranie
zapytania od użytkownika
    try:
         output_message = seng.create_output(user_query)      #
Wysłanie zapytania do funkcji create_output
                                                                 #
i stworzenie odpowiedzi
                                                                 #
Poniżej, w zależności od wymagań serwera,
                                                                 #
odpowiedź może zostać zastąpiona wartością None
        if output_message is None:
            output_message = "Nie rozumiem"
    except IndexError:
        output_message = "Nie rozumiem"
    print(output_message)
       write_to_logs(user_query, output_message)   # Zapis do
logów
    return output_message


if __name__ == '__main__':
    main()
```

========== plik sejm_main_mqtt.py ==========

```
# coding=utf8
from src.voice_assistant_modules.va_module import VAModule
import search_engine as seng

"""
#####################################################################
#####################################
Ten plik zawiera metodę politModule() i służy do uruchomienia
programu przy użyciu serwera mqtt.
Dostosowany został do podłączenia do serwera Voice Assistant
napisanego przez Wojciecha Węgrzynka.

#####################################################################
#####################################
"""


def write_to_logs(query, output):
    """
      Funkcja służąca do zapisywania zapytań i odpowiedzi w
pliku logs.txt.
     :param query: zapytanie, które pobierane jest z serwera
mqtt (lub, w przypadku tego programy testowego,
    z inputu użytkownika.
     :param output: string, który wysyłany jest do serwera.
Stanowi odpowiedź na zapytanie.
    :return: None.
    """
    from datetime import datetime, date
    now = datetime.now()
    current_time = now.strftime("%H:%M:%S")
    today = date.today()
    with open("logs.txt", "a") as log:
                log.write(f"{str(today)}  {str(current_time)}
\nZapytanie: {query} \nOdpowiedź: {output}\n \n \n")


class politModule(VAModule):
    @classmethod
    def get_id(cls):
        return "politics"

    def process_query(self, query: str) -> str:
        """
        Funkcja pozwalająca przetworzyć zapytanie.
        :return: odpowiedź na zapytanie użytkownika.
        """
              user_query = query   # Pobranie zapytania od
użytkownika
```

```
        try:
             output_message = seng.create_output(user_query)  #
Wysłanie zapytania do funkcji create_output
            # i stworzenie odpowiedzi
                 # Poniżej, w zależności od wymagań serwera,
odpowiedź może zostać zastąpiona wartością None
            if output_message is None:
                output_message = None
        except IndexError:
            output_message = None
        print(output_message)
         write_to_logs(user_query, output_message)  # Zapis do
logów
        return output_message


if __name__ == '__main__':
    politModule.main()
```

========== plik search_engine.py ==========

```
import csv
import parse_from_input as pfi    # Zaimportowanie programu parsującego

"""
##################################################################
###################################
Plik search_engine.py jest zasadniczo "mózgiem" modułu. To w nim bazy polityków są przeszukiwane,
a odpowiedzi na zapytania formułowane. Funkcje korzystają z pliku parse_from_input.py, który
zawiera słownik słów kluczowych oraz z plików csv:
- sejm.csv - zawierającego dane na temat posłów i posłanek;
- politicians.csv - zawierającego dane na temat polityków pełniących ważniejsze lub wyróżniające
się funkcje
- clubs.csv - zawierającego dane na temat klubów i kół w polskim sejmie (traktowane są one jednak
dla uproszczenia jak partie)

##################################################################
###################################
"""

"""Poniżej znajduje się blok kodu potrzebnego do zapytań o członkostwo w partiach.
Pozwala on przeanalizować imiona i nazwiska oddzielnie"""
firstnames = []
lastnames = []
fullnames = []
with open("sejm.csv", "r", encoding="utf-8") as sejm:
    sejm = csv.reader(sejm)
    for row in sejm:
        firstnames.append(row[1])
        lastnames.append(row[0])
        fullnames.append(row[1] + ' ' + row[0])

lastnames = [y for x in lastnames for y in x.split(' ')]  # Rozbicie dwuczłonowych nazwisk na pojedyncze elementy listy


##################################################################
##################

def search_politicians(keyword):
    """
    Funkcja przeszukująca polityków pod względem funkcji.
    :param keyword: pojedynczy element sparsowanego zapytania.
    :return: fraza będąca odpowiedzią na zapytanie.
```

```
    """
       with open('politicians.csv', 'r', encoding="utf-8") as
polits:
        reader = csv.reader(polits)
        for row in reader:
            if keyword in row:
                        return f'{keyword} polski to {row[1]}
{row[0]}'


def count_sejm(keyword="", party=""):
    """
    Funkcja licząca członków sejmu. Oba argumenty opcjonalne.
     :param keyword: pojedynczy element sparsowanego zapytania
oznaczający płeć.
      :param party: pojedynczy element sparsowanego zapytania
oznaczający partię.
    :return: integer wyrażający liczbę osób w sejmie.
    """

    """Jeśli zapytanie dotyczy wszystkich posłów:"""
    if keyword == "" and party == "":
        return 459

    count = 0
    with open('sejm.csv', 'r', encoding="utf-8") as sejm:
        reader = csv.reader(sejm)
        for row in reader:
            if keyword in row and party == "":
                count += 1
            elif keyword in row and party in row:
                count += 1
    # print(count)
    return count


def search_clubs(keywords):
    """
     Funkcja przeszukująca kluby i koła, jakie znajdują się w
sejmie.
    :param keywords: sparsowane zapytanie użytkownika.
    :return: wiersz z pliku clubs.csv.
    """
    with open("clubs.csv", "r", encoding="utf-8") as clubs:
        clubs = csv.reader(clubs)
          for item in keywords:                              # dla
każdego elementu w zapytaniu
                if item in pfi.clubs_keys.keys():            # dla
każdego klucza w słowniku kluczy club_keys
                    for line in clubs:                       # dla
każdego wiersza w pliku clubs.csv
```

```
                    if item == line[1]:
                        # print(line)
                          return f"przewodniczącym ugrupowania
sejmowego {pfi.clubs_keys.get(line[1])[1]} jest {line[2]}"


def search_kto(keywords):
    """
      Funkcja odpowiadająca na pytanie o to, kto pełni daną
funkcję.
    :param keywords: sparsowane zapytanie użytkownika.
    :return: wykonanie funkcji search_politicians().
    """
    if "kto" in keywords:
        if "prezydent" in keywords:
            return search_politicians("prezydent")
        if "rpo" in keywords:
                    return  search_politicians("rzecznik  praw
obywatelskich")
        if "leader" in keywords:
            return search_clubs(keywords)


def search_ilu(keywords):
    """
    Funkcja obsługująca zapytania o liczebność polityków.
    :param keywords: sparsowane zapytanie użytkownika.
    :return: fraza będąca odpowiedzią na zapytanie.
    """
    if "ilu" in keywords:
        if "posel" in keywords:
            for word in keywords:
                if word in pfi.clubs_keys.keys():
                                if word == "niezrzeszony":   #
wyszczególnienie  posłów  niezrzeszonych  wynika  z  przyczyn
stylistycznych
                        return f"Posłów niezrzeszonych jest w
sejmie {count_sejm(word)}"
                    else:
                         return f"{pfi.clubs_keys.get(word)[1]}
ma w sejmie {count_sejm(word)} posłów"
            return f"posłów w sejmie jest {count_sejm()}"
        if "prezydent" in keywords:
            return "prezydent jest tylko jeden"
        if "premier" in keywords:
            return "premier jest tylko jeden"
            if "m" in keywords:   # z przyczyn gramatycznych
rozróżnienie ze względu na płeć znajduje się w tej funkcji
            for word in keywords:
                  if word in pfi.clubs_keys.keys() and word !=
"partia":
```

```
                                return f"mężczyzn w ugrupowaniu
{pfi.clubs_keys.get(word)[1]} jest {count_sejm('m', word)}"
            return f"mężczyzn jest w sejmie {count_sejm('m')}"


def search_ile(keywords):
    """
      Rozszerzenie gramatyczne funkcji liczącej polityków. Ta
funkcja liczy tylko kobiety.
    :param keywords: sparsowane zapytanie użytkownika.
    :return: fraza będąca odpowiedzią na zapytanie.
    """
    if "ile" in keywords:
        if "k" in keywords:
            for word in keywords:
                  if word in pfi.clubs_keys.keys() and word !=
"partia":
                                  return f"kobiet w ugrupowaniu
{pfi.clubs_keys.get(word)[1]} jest {count_sejm('k', word)}"
            return f"kobiet jest w sejmie {count_sejm('k')}"


def search_w(keywords, split_query):
    """
       Funkcja obsługująca zapytania zaczynające się na "w".
Głównie dotyczące przynależności kogoś do partii.
    :param keywords: sparsowane zapytanie użytkownika.
       :param split_query: niesparsowane, ale podzielone na
odrębne słowa zapytanie użytkownika.
    :return: fraza będąca odpowiedzią na zapytanie.
    """
    if "w" in keywords[0]:
          """Obie listy poniżej różnią się od tych z początku
kody tym,
            że te zawierają tylko elementy wyszukane w bazie
polityków"""
        list_of_lastnames = []
        list_of_firstnames = []
        club = []
          ask = set()      # utworzenie zbioru potrzebnego do
określenia czy użytkownik podał imię, nazwisko czy oba
        # print(split_query)

        for name in split_query[1:]:
              with open('sejm.csv', 'r', encoding="utf-8") as
sejm:
                sejm = csv.reader(sejm)

                for row in sejm:
                        """Oddzielnie przetwarzane są imiona i
nazwiska"""
```

```
                    if name in row[0] and name in lastnames:
                        list_of_lastnames.append(name)
                        club.append(row[2])
                        ask.add("lastname")
                    if name in row[1] and name in firstnames:
                        club.append(row[2])
                        list_of_firstnames.append(name)
                        ask.add("firstname")

        ask = list(ask)
         if len(ask) == 2:                              # podanie
imienia i nazwiska
                  firstlast = list_of_firstnames[0] + ' ' +
list_of_lastnames[0]
                   lastfirst = list_of_lastnames[0] + ' ' +
list_of_firstnames[0]
            if firstlast in fullnames:
                # print(firstlast)
                  with open("sejm.csv", "r", encoding="utf-8")
as sejm:
                    sejm = csv.reader(sejm)
                    for line in sejm:
                           if list_of_firstnames[0] == line[1]
and list_of_lastnames[0] == line[0]:
                            # print(firstlast, line[2])
                                  return f"{firstlast} jest w
ugrupowaniu {pfi.clubs_keys.get(line[2])[1]}"
            elif lastfirst in fullnames:
                # print(lastfirst)
                  with open("sejm.csv", "r", encoding="utf-8")
as sejm:
                    sejm = csv.reader(sejm)
                    for line in sejm:
                           if list_of_firstnames[0] == line[0]
and list_of_lastnames[0] == line[1]:
                            # print(lastfirst, line[2])
                                  return f"{lastfirst} jest w
ugrupowaniu {pfi.clubs_keys.get(line[2])[1]}"
             elif firstlast not in fullnames and lastfirst not
in fullnames:
                return "Nie ma takiej osoby"
         elif len(ask) == 1:                            # podanie
tylko imienia lub nazwiska
            if ask[0] == "firstname":
                          if len(list_of_lastnames) == 0 and
len(list_of_firstnames) > 1:  # powtórzenie imienia
                     return "Więcej niż jedna osoba ma tak na
imię"
            elif ask[0] == "lastname":
                          if len(list_of_lastnames) > 1 and
len(list_of_firstnames) == 0:  # powtórzenie nazwiska
```

```
                        return "Więcej niż jedna osoba się tak
nazywa"

        # print(list_of_firstnames)

           """W poniższym bloku kodu sprawdzane są przypadki
podania samego imienia lub nazwiska, a także sytuacja,
        w której jakiejś osoby nie ma w bazie danych"""
                    if len(list_of_lastnames) == 1 and
len(list_of_firstnames) == 0:   # podanie w zapytaniu samego
nazwiska
                return f"{list_of_lastnames[0]} jest w partii
{pfi.clubs_keys.get(club[0])[1]}"
                    elif len(list_of_firstnames) == 1 and
len(list_of_lastnames) == 0:   # podanie w zapytaniu samego
imienia
                return f"{list_of_firstnames[0]} jest w partii
{pfi.clubs_keys.get(club[0])[1]}"
                    elif len(list_of_firstnames) == 0 and
len(list_of_lastnames) == 0:   # podanie w zapytaniu osoby
spoza listy
            return "w żadnej partii nie ma takiej osoby"


def create_output(user_query):
    """
       Funkcja, która wywołuje wszystkie powyższe funkcje w
zależności od zapytania. Jest wywoływana w metodzie main.
    :param user_query: Niesparsowane i niepodzielone zapytanie
użytkownika.
    :return: fraza będąca odpowiedzią na zapytanie.
    """
         split_query = pfi.query_splitting(user_query)     #
podzielenie zapytania na listę słów
         keywords = pfi.parse_from_input(split_query)     #
zakodowanie listy słów według słownika słów kluczowych
    if "kto" in keywords:
        return search_kto(keywords)
    if "ilu" in keywords:
        return search_ilu(keywords)
    if "ile" in keywords:
        return search_ile(keywords)
    if "w" in keywords[0] and "partia" in keywords:
        return search_w(keywords, split_query)
    return None
```

========== plik parse_from_input.py ==========

```
"""
#####################################################################
###################################

Plik parse_from_input.py zawiera funkcje pozwalające
przetworzyć zapytanie użytkownika na listę słów
kluczowych, które następnie może wykorzystać silnik
wyszukiwania. Oprócz dwóch parsujących funkcji
plik zawiera również słowniki, które pozwalają zakodować użyte
przez użytkownika w zapytaniu
słowa.

#####################################################################
###################################
"""

"""
Poniższe słowniki zawierają klucze pozwalające zakodować
wypowiedź na użytek dalszego przetwarzania.
Stosowany jest tu następujący wzór:
klucz: [lista słów, które mogą pojawić się w zapytaniu]
"""

# klucze rozpoczęcia zapytania
questions_keys = {
    "kto": ["kto", "któż"],
    "ile": ["ile", "ileż"],
    "ilu": ["ilu", "iluż"],
    "w": ["w"]
}

# klucze określenia czasu
time_keys = {
    "jest": ["jest", "jestże"],
}

# klucze zapytania o funkcję
occupations_keys = {
    "prezydent": ["prezydent", "prezydentem", "prezydentów"],
    "posel": ["poseł", "posłem", "posłów"],
    "premier": ["premier", "premierem", "premierów"],
      "leader": ["przewodniczący", "przewodniczącym", "lider",
"liderem", "szef", "szefem"],
    "rpo": ["rpo", "rzecznik", "rzecznikiem"]
}

# klucze zapytania o płeć
gender_keys = {
    "k": ["k",
          "kobieta",
```

```
            "kobiet",
            "kobiety"],
    "m": ["m",
            "mężczyzna",
            "mężczyzn",
            "mężczyźni"]
}

# klucze zapytania o przynależność partyjną
# druga pozycja w liście to nazwa użytkowa
clubs_keys = {
    "pis": ["pis",
             "prawo i sprawiedliwość",
             "pisu",
             "pisowi",
             "prawa",
             "prawo",
             "prawie",
             "prawu"],
    "ko": ["ko",
            "koalicja obywatelska",
            "koalicja",
            "koalicji"],
    "lewica": ["lewica",
               "lewica",
               "lewicy"],
    "kp": ["kp",
            "koalicja polska",
            "polska",
            "polskiej",
            "kape"],
    "konfederacja": ["konfederacja",
                     "konfederacja",
                     "konfederacji",
                     "konfa",
                     "konfie",
                     "konfy"],
    "polska2050": ["2050",
                   "pięćdziesiąt",
                   "polska 2050"],
    "porozumienie": ["porozumienie",
                     "porozumienie",
                     "porozumieniu",
                     "porozumienia"],
    "kukiz15": ["kukiz",
                "piętnaście",
                "kukiz15"],
    "ps": ["ps",
            "polskie sprawy",
            "pees",
            "sprawy",
```

```
            "spraw",
            "sprawach",
            "polskich"],
    "pps": ["pepees",
            "polska partia socjalistyczna",
            "pps"],
    "niezrzeszony": ["niezrzeszony",
                     "niezrzeszeni",
                     "niezrzeszonych"],
    "partia": ["partia",
               "partii",
               "ugrupowanie",
               "ugrupowaniu",
               "klub",
               "klubie",
               "klubu"]
}


def query_splitting(user_query):
    """
     Funkcja rozbijająca zapytanie na listę wchodzących w jego
skład słów.
    :param user_query: zapytanie użytkownika.
          :return:  zapytanie  użytkownika  w  formie  listy
poszczególnych słów.
    """
    split_user_query = user_query.split()
    return split_user_query


# lista słowników słów kluczowych
dicts   =   [questions_keys,   time_keys,   occupations_keys,
gender_keys, clubs_keys]


def parse_from_input(split_query):
    """
      Funkcja kodująca rozbite zapytanie użytkownika na listę
kodów odpowiadających słowom z zapytania.
    Tworzy listę, która będzie użyta w dalszym przetwarzaniu.
     :param split_query: zapytanie użytkownika w formie listy
poszczególnych słów.
    :return: zakodowane zapytanie użytkownika.
    """
    list_of_keys = []
     for dictionary in dicts:                              # dla
każdego słownika kluczy
          for word in split_query:                         # dla
każdego słowa w zapytaniu
```

```
                for key in dictionary.keys():                # dla
każdego klucza w konkretnym słowniku
                    if word in dictionary.get(key):
                        list_of_keys.append(key)
                        break

    # print(list_of_keys)

        """Blok odpowiadający komunikatowi zwrotnemu: <<Nie
rozumiem ani słowa>>"""
    if list_of_keys == []:
        return "Nie rozumiem"

    return list_of_keys
```