

## **Asystent głosowy**

M. K.

M. K.

J. M.

F. M.

POLITECHNIKA KRAKOWSKA

Przetwarzanie Języka Naturalnego

*11.05.2024*

## Spis treści

<b>1. Abstrakt.....</b>	<b>2</b>
<b>2. Wstęp.....</b>	<b>4</b>
2.1. Cel.....	4
2.2. Zakres.....	4
2.3. Metodyka.....	4
<b>3. Zagadnienia projektowe.....</b>	<b>7</b>
3.1. Rozpoznawanie mowy.....	7
3.2. Analiza składniowa.....	7
3.3. Wektoryzacja TF-IDF.....	7
3.4. Wyszukiwanie semantyczne.....	8
3.5. Wyrażenia regularne.....	8
3.6. Synteza mowy.....	8
<b>4. Implementacja.....</b>	<b>9</b>
4.1. Analiza składniowa.....	9
4.2. Trenowanie asystenta.....	9
4.3. Obsługa zapytania.....	9
4.4. Przetwarzanie mowy na tekst oraz jego synteza.....	10
4.5. Obsługa złożonych komend.....	12
4.6. Wyrażenia regularne.....	13
<b>5. Rezultaty.....</b>	<b>14</b>
<b>6. Podsumowanie.....</b>	<b>16</b>
<b>Bibliografia.....</b>	<b>17</b>

## **1. Abstrakt**

Projekt ma na celu zaprojektowanie i zaimplementowanie intuicyjnego i łatwo dostępnego narzędzia, które pozwala użytkownikom na interakcję głosową z asystentem. Wykorzystuje on zaawansowane technologie rozpoznawania oraz przetwarzania mowy. Oferuje on kilka ciekawych funkcjonalności takich jak informowanie o pogodzie czy też opowiadanie żartów.

## **2. Wstęp**

### **2.1. Cel**

Celem niniejszej pracy jest zaprojektowanie oraz zaimplementowanie asystenta głosowego, który będzie w stanie skutecznie rozumieć i interpretować polecenia użytkownika w języku polskim, a także reagować na nie.

### **2.2. Zakres**

Projekt obejmuje zdefiniowanie architektury systemu, w tym modułów odpowiedzialnych za rozpoznawanie mowy, interpretację poleceń oraz ich obsługę, a także syntezę mowy odpowiedzi asystenta. Kolejnym krokiem będzie implementacja asystenta głosowego, wraz z przedstawionymi wyżej modułami, sporządzenie podstawowych poleceń głosowych i ich odpowiedzi oraz obsługa bardziej złożonych komend, które wymagają pewnej logiki lub dostępu do zewnętrznych źródeł informacji.

### **2.3. Metodyka**

Została przeprowadzona analiza literatury, obejmująca przegląd artykułów naukowych, książek oraz dokumentacji technicznej dotyczącej asystentów głosowych i technik NLP. Następnie zdefiniowano architekturę systemu oraz interfejsy między poszczególnymi modułami, a także został dokonany wybór odpowiednich narzędzi i technologii. Wykorzystano narzędzia do:

- rozpoznawania mowy (speech\_recognition) - popularna biblioteka umożliwiająca rozpoznawanie mowy przy użyciu różnych silników i API, zarówno online, jak i offline. Obsługuje m.in. CMU Sphinx, Google Speech Recognition, OpenAI whisper. Umożliwia przetwarzanie plików audio oraz bezpośrednie nagrywanie z mikrofonu, oferując przy

tym prosty w użyciu interfejs, który pozwala na szybkie integrowanie funkcji rozpoznawania mowy w języku Python [2].

- analiza składniowa tzw. tokenizacja (spaCy) - to zaawansowana biblioteka do przetwarzania języka naturalnego (NLP). Umożliwia analizę tekstu, ekstrakcję informacji, tokenizację, lematyzację, tagowanie części mowy, rozpoznawanie nazw własnych, analizę składniową i semantyczną [3].
- analizy poleceń z wykorzystaniem wektorów TF-IDF oraz wyszukiwania semantycznego (sklearn) - popularna biblioteka służąca do uczenia maszynowego w języku Python. Oferuje algorytmy do klasyfikacji, regresji, klasteryzacji, redukcji wymiarów oraz selekcji modeli. Bazuje na bibliotekach NumPy, SciPy i matplotlib, zapewniając efektywne narzędzia do analizy danych [4].
- obsługi wyrażeń regularnych (re) - biblioteka wbudowana w język Python, umożliwiająca pracę z wyrażeniami regularnymi. Umożliwia wyszukiwanie, dopasowywanie i manipulowanie wzorcami w łańcuchach tekstowych [5].
- wysyłania zapytań do zewnętrznych źródeł informacji (requests) - Biblioteka jest popularnym narzędziem do wykonywania zapytań HTTP w języku Python. Upraszcza wysyłanie żądań i obsługę odpowiedzi z serwerów. Oferuje intuicyjne API do wykonywania zapytań GET, POST, PUT, DELETE i innych, umożliwiając łatwe dodawanie nagłówek, parametrów i danych. Requests obsługuje również autoryzację, weryfikację certyfikatów SSL i obsługę sesji, co czyni ją wszechstronnym narzędziem do integracji z różnymi API i serwisami internetowymi [6].

- syntezy mowy (pyttsx3) - biblioteka służąca do konwersji tekstu na mowę, działająca w trybie offline. Umożliwia wybór różnych rodzajów głosów zainstalowanych w systemie, kontrolę szybkości mowy oraz głośności, a także zapis mowy do pliku audio [7].

Z powodu problemów z instalacją systemu Blather oraz brakiem wsparcia języka polskiego przez bibliotekę Sphinx, zostały wykorzystane inne implementacje. W ich miejsce została wybrana biblioteka speech\_recognition wykorzystująca Google Speech Recognition.

### **3. Zagadnienia projektowe**

#### **3.1. Rozpoznawanie mowy**

Rozpoznawanie mowy to proces przekształcania mowy na tekst. Umożliwia to użytkownikom komunikację z systemem za pomocą wypowiedzianych słów. Proces ten obejmuje rejestrację dźwięków za pomocą mikrofonu, a następnie ich analizę w celu identyfikacji wypowiedzianych słów i zdań. Elementami składowymi są: kalibracja mikrofonu do warunków otoczenia oraz przetwarzanie dźwięku w celu uzyskania tekstu, który następnie może być analizowany przez system [1, 2].

#### **3.2. Analiza składniowa**

Analiza składniowa jest podstawowym krokiem przetwarzania języka naturalnego. To proces polegający na rozkładaniu łańcucha znaków na mniejsze jednostki, czyli tokeny. Analiza składniowa umożliwia zrozumienie struktury tekstu i jest podstawą do dalszej analizy semantycznej. Pozwala ona systemowi skutecznie rozpoznawać i interpretować zapytania użytkowników [1].

#### **3.3. Wektoryzacja TF-IDF**

Wektoryzacja TF-IDF (ang. Term Frequency-Inverse Document Frequency) to technika przekształcania tekstu na reprezentację numeryczną, która odzwierciedla wagę słów w kontekście całego zbioru dokumentów. Proces ten polega na obliczaniu częstości występowania słów w danym dokumencie oraz ich odwrotnej częstości występowania w całym korpusie [1]. W projekcie asystenta głosowego TF-IDF jest używane do modelowania zapytań i odpowiedzi. Dzięki temu system może ocenić, które słowa są najistotniejsze dla danego zapytania.

### **3.4. Wyszukiwanie semantyczne**

Wyszukiwanie semantyczne to zaawansowana metoda porównywania zapytań użytkowników z wyuczonymi poleceniami, uwzględniająca kontekst i znaczenie słów, a nie tylko ich powierzchowne dopasowanie. W przeciwieństwie do samej wektoryzacji TF-IDF, która tworzy wektory na podstawie ważności słów, wyszukiwanie semantyczne interpretuje te wektory, aby określić relacje i kontekst między słowami w różnych dokumentach [1].

### **3.5. Wyrażenia regularne**

Wyrażenia regularne to narzędzie do przetwarzania tekstu, które umożliwia wyszukiwanie i manipulowanie wzorcami tekstowymi. W projekcie asystenta głosowego wyrażenia regularne są używane do ekstrakcji specyficznych informacji z tekstu oraz do rozpoznawania określonych wzorców w zapytaniach użytkowników. Pozwalają one na skuteczne filtrowanie i analizę danych tekstowych, co jest ważne dla poprawnego rozpoznawania i interpretacji poleceń [1].

### **3.6. Synteza mowy**

Synteza mowy to proces przekształcania tekstu na mowę, który umożliwia asystentowi głosowemu komunikowanie się z użytkownikami w sposób naturalny. W projekcie synteza mowy obejmuje konwersję generowanych odpowiedzi tekstowych na dźwięk, który jest następnie odtwarzany użytkownikowi. Technologia syntezy mowy pozwala na realistyczne i płynne odtwarzanie mowy, co zwiększa interaktywność i użyteczność asystenta głosowego.



## 4. Implementacja

### 4.1. Analiza składniowa

Asystent wykorzystuje analizę składniową do przetwarzania tekstu. Tokenizacja, czyli dzielenie tekstu na mniejsze jednostki, jest istotnym elementem tego procesu. Tokeny są lematyzowane, co oznacza, że są sprowadzane do ich podstawowej formy [1].

```
def tokenize(self, sentence):  
    return [token.lemma_ for token in self.nlp(sentence) if  
            not token.is_punct and not token.is_space and not token.is_stop]
```

### 4.2. Trenowanie asystenta

Model TF-IDF jest trenowany na zapytaniach i komendach asystenta. Proces ten obejmuje budowę korpusu tekstów, które są następnie przekształcane na wektory TF-IDF. Model pozwala na ocenę podobieństwa między zapytaniem użytkownika a zapisanymi odpowiedziami [1].

```
def train(self):  
    self.vectorizer = TfidfVectorizer(tokenizer=self.tokenize)  
  
    self.instructions = [(q, a) for q, a in self.queries]  
    for command in self.commands:  
        self.instructions.extend([(p, command) for p in command.prompts])  
  
    corpus = [sentence.lower() for sentence in [q for q, a in self.instructions]]  
    self.model = self.vectorizer.fit_transform(corpus)
```

### 4.3. Obsługa zapytania

Główna funkcja odpowiedzialna za interakcję z użytkownikiem porównuje zapytanie użytkownika z wyuczonymi poleceniami przy użyciu miary podobieństwa kosinusowego. Kiedy użytkownik wprowadza zapytanie, system najpierw przetwarza je w celu uzyskania reprezentacji

wektorowej. Następnie wyliczana jest wartość podobieństwa tego zapytania z wektorami TF-IDF.

```
def respond(self, query):
    if not self.vectorizer:
        raise Exception("Asystent nie został jeszcze przeszkolony.")

    query_model = self.vectorizer.transform([query])
    similarities = cosine_similarity(query_model, self.model).flatten()

    best_match_index = similarities.argmax()
    best_similarity = similarities[best_match_index]
    best_answer = self.instructions[best_match_index][1]

    if best_similarity < 0.3:
        return "Przepraszam, ale cię nie rozumiem..."

    return best_answer
```

#### 4.4. Przetwarzanie mowy na tekst oraz jego synteza

Klasa VoiceAssistant integruje rozpoznawanie mowy oraz syntezę mowy. Rozpoznawanie mowy umożliwia odbieranie poleceń głosowych od użytkownika, natomiast synteza mowy pozwala na odtwarzanie odpowiedzi w formie dźwiękowej.

```

class VoiceAssistant(Assistant):
    └ Marcin Kokoszka
    def __init__(self, name):
        super().__init__(name)
        self.tts = pyttsx3.init()
        rate = self.tts.getProperty('rate')
        self.tts.setProperty('rate', rate / 5 * 4)

    4 usages └ Marcin Kokoszka
    def synthesise(self, text):
        print(text)
        self.tts.say(text)
        self.tts.runAndWait()

    1 usage └ Marcin Kokoszka
    def listen(self):
        import speech_recognition as sr

        recognizer = sr.Recognizer()

        with sr.Microphone() as source:
            print("Trwa kalibracja mikrofonu. Proszę czekać...")
            recognizer.adjust_for_ambient_noise(source, duration=5)

            while True:
                print("> ", end="", flush=True)

                audio = recognizer.listen(source)

                try:
                    query = recognizer.recognize_google(audio, language='pl-PL')
                    print(query)

                    answer = self.respond(query)
                    if answer:
                        if isinstance(answer, Command):
                            self.synthesise(answer.execute(query))
                            answer.post_execute(query)
                        else:
                            self.synthesise(answer)
                    else:
                        self.synthesise("Nie rozumiem pytania.")
                except sr.UnknownValueError:
                    pass
                except sr.RequestError as e:
                    print("")
                    self.synthesise("Wystąpił błąd: {0}".format(e))

```

## 4.5. Obsługa złożonych komend

Klasa “Command” służy jako szablon dla komend, z wymaganą do implementacji metodą `execute` i opcjonalną metodą `post_execute`.

```
class Command(ABC):
    def __init__(self, prompts):
        self.prompts = prompts

    @abstractmethod
    def execute(self, query):
        pass

    def post_execute(self, query):
        pass
```

Klasa “QuitCommand” to konkretna implementacja komendy, która zarządza procesem zamykania aplikacji.

```
class QuitCommand(Command):
    def __init__(self):
        super().__init__([
            "Żegnaj.",
            "Dobranoc.",
            "Do widzenia.",
            "Do zobaczenia.",
            "Do zobaczenia później.",
            "Na razie.",
            "Pa",
        ])

    def execute(self, query):
        return "Do usłyszenia!"

    def post_execute(self, query):
        sys.exit()
```

## 4.6. Wyrażenia regularne

W celu wyodrębniania z polecenia nazwy miasta oraz docelowego dnia zostały wykorzystane wyrażenia regularne [1,7]. Zostały one przedstawione na poniższym obrazku.

```
def extract_city(self, query):
    pattern = r"(?:w mieście|we wsi|w miejscowości)\s+(.+)?:\s+(za|na)\s+(.*)?"
    match = re.search(pattern, query, re.IGNORECASE)
    if match:
        return match.group(1).strip().rstrip(',')
    else:
        return None

def extract_time(self, query):
    day_patterns = [
        (r'jutro|1 dzień|jeden dzień', 1),
        (r'pojutrze|2 dni|dwa dni', 2),
        (r'3 dni|trzy dni', 3),
        (r'4 dni|cztery dni', 4),
        (r'dni', 100),
        (r'tydzień', 7),
        (r'tygodni', 35),
        (r'tygodnie', 14),
        (r'miesiąc', 30),
    ]
    for pattern, days in day_patterns:
        match = re.search(pattern, query)
        if match:
            if days is not None:
                if days >= 5:
                    return None
                return days
    return 0
```

## 5. Rezultaty

W celu zaprezentowania działania zaimplementowanego asystenta głosowego została przeprowadzona rozmowa testowa. Przedstawia ona obsługiwane pytania oraz bardziej złożone komendy.

```
Trwa kalibracja mikrofonu. Proszę czekać...
> Witaj
Cześć
> jak masz na imię
Moje imię to Siri
> Jak się czujesz
Dobrze, dziękuję.
> Uruchom stoper
Stoper został włączony.
> Jaki jest dzisiaj dzień
Dzisiejsza data to 18.05.2024
> która godzina
Teraz jest 15:26
> Jaka będzie pogoda w miejscowości Bochnia za 2 dni
Pogoda w Bochnia na dzień 2 dni od dzisiaj: zachmurzenie umiarkowane, temperatura: 23.28°C
> Jaka jest dzisiaj pogoda w mieście Kraków
Pogoda w Kraków na dziś: pochmurnie, temperatura: 22.75°C
> Powiedz dowolny żart
Dlaczego ściany nie toczą ze sobą wojen? - Bo pomiędzy nimi jest pokój
> Opowiedz żart o teściowej
Jak się nazywają żona i teściowa, siedzące razem w samochodzie? - Zestaw głośnomówiący
> Zakończ mierzenie czasu
Stoper został zatrzymany. Upłynęło minuta.
> żegnaj
Do usłyszenia!

Process finished with exit code 0
```

Przeprowadzona rozmowa z asystentem głosowym demonstruje jego zdolność do prowadzenia naturalnych interakcji z użytkownikiem. Asystent rozpoznał i odpowiedział na pytania dotyczące imienia oraz samopoczucia, co wskazuje na poprawną implementację modułów przetwarzania języka naturalnego i syntezy mowy. System poprawnie zrealizował

polecenia związane z uruchamianiem i zatrzymywaniem stopera, potwierdzając jego funkcjonalność w zakresie obsługi komend.

Asystent również udzielił poprawnych odpowiedzi na pytania o aktualną datę i godzinę, jak również prognozę pogody dla różnych miejscowości. Pokazuje to zdolność systemu do integracji z usługami zewnętrznymi i aktualizowania danych w czasie rzeczywistym.

Na zakończenie, asystent poprawnie zinterpretował polecenie zakończenia sesji i odpowiednio się pożegnał, co potwierdza jego zdolność do obsługi poleceń końcowych. Wyniki rozmowy potwierdzają, że asystent głosowy działa zgodnie z założeniami projektowymi, efektywnie realizując zaplanowane funkcje i zapewniając interaktywność oraz użyteczność w codziennych zadaniach użytkownika.

## 6. Podsumowanie

Projekt asystenta głosowego został pomyślnie zrealizowany, osiągając założone cele. Asystent skutecznie rozpoznawał i odpowiadał na pytania dotyczące imienia, samopoczucia, daty, godziny oraz prognozy pogody, a także realizował polecenia związane ze stoperem i opowiadał żarty. Podczas implementacji napotkaliśmy garstkę problemów. Pierwszym z nich była niewspierany już system Blather. W repozytorium GitHub autor oznaczył ten projekt jako *deprecated*. Brakowało również odpowiedniej dokumentacji, jak ten system uruchomić. Kolejnym problemem była trudność podczas implementowania poleceń pogodowych. W języku polskim nazwy miejscowości są odmieniane co powoduje konieczność konkretnego formułowania zdań. Ten sam problem występował podczas prośby o żart z konkretnej kategorii.



## Bibliografia

- [1] dr R. Kycia, Materiały udostępnione w ramach wykładów i laboratoriów z przedmiotu  
“Przetwarzanie Języka Naturalnego”
- [2] Dokumentacja techniczna Speech Recognition API,  
[https://github.com/Uberi/speech\\_recognition](https://github.com/Uberi/speech_recognition) (data uzyskania dostępu: 4 maj 2024)
- [3] Dokumentacja spaCy <https://spacy.io/api/doc> (data uzyskania dostępu: 4 maj 2024)
- [4] Dokumentacja biblioteki sklearn, <https://scikit-learn.org/stable/> (data uzyskania dostępu: 4 maj 2024)
- [5] Dokumentacja biblioteki re, <https://docs.python.org/3/library/re.html> (data uzyskania dostępu: 5 maj 2024)
- [6] Dokumentacja biblioteki requests, <https://realpython.com/python-requests/> (data uzyskania dostępu: 5 maj 2024)
- [7] Dokumentacja biblioteki pyttsx3, <https://github.com/nateshmbhat/pyttsx3> (data uzyskania dostępu: 5 maj 2024)