**Biblioteki do języka polskiego w Chatbot – zagadnienia teoretyczne**

# Rasa – czym jest i jak działa?

# 

Rasa Open Source to narzędzie o otwartym kodzie źródłowym, oparte o uczenie maszynowe. Jego obszary wykorzystania to: rozmowy tekstowe i głosowe ( zrozumienie otrzymanej wiadomości, podtrzymanie rozmowy, wydobycie informacji z wypowiedzi użytkownika ). Rasa’ę możemy zintegrować z API oraz kanałami komunikacji.

Odkąd udostępniona została jest otwarta wersja, produkt pobrany został ponad 10 mln razy. Oprócz braku opłat dużymi zaletami Rasa są: możliwości wytrenowania bota w dowolnym języku oraz użycia gotowych bibliotek wspierających dużą liczbę języków, łatwość instalacji i korzystania z chatbota w interfejsie konsolowym ( niewielka liczba komend oraz wysoka intuicyjność w ich stosowaniu ).

1. Funkcjonalności

* **Rozumienie języka naturalnego**

Odpowiedzialny jest za nie komponent Rasa NLU. Ta część analizy wypowiedzi pozwala zdecydować o jej znaczeniu i przyporządkować treść wiadomości do odpowiedniej kategorii. Platforma z wykorzystaniem bibliotek Tensorflow i spaCy przekształca zwykły tekst pozyskany z wiadomości na intencje i encje – dane uformowane w sposób zrozumiały dla chatbota.

Jak to się robi?

Na początku musimy zdecydować, który z dostępnych komponentów chcemy wykorzystać. Dostępne są osadzenia już wytrenowane lub nadzorowane.

W przypadku wcześniej wytrenowanego osadzenia korzystamy z klasyfikatora „Intent classifier sklearn”. W tej metodzie używamy gotowych modeli językowych z biblioteki spaCy w celu wykreowania osadzenia słowa. Osadzenie słowa to reprezentacja wyrazu w postaci wektora liczb, na którą wpływ mają aspekty znaczeniowe i składniowe słowa. Podobne słowa są reprezentowane przez podobne wektory. Osadzenia słów są charakterystyczne dla modeli językowych, dla których zostały wytrenowane. Silnik Rasa NLU wylicza średnią z osadzeń wszystkich słów z danej wiadomości i trenuje wiele klasyfikatorów różniących się parametrami, aby znaleźć konfigurację parametrów, która osiągnie najlepsze wyniki w testach. To z jej wykorzystaniem silnik dokona przyporządkowania do intencji.

Gdy zdecydujemy się na osadzenie nadzorowane wykorzystamy klasyfikator „Intent classifier tensorflow embedding” utworzonego przez Rasa’ę. Zamiast trenować klasyfikator w oparciu o gotowe modele, komponent przygotowuje osadzenia słowa od zera m.in. zliczając wystąpienia konkretnych słów w jednej wiadomości. Zaletą zastosowania tego sposobu przyporządkowania wypowiedzi jest możliwość klasyfikacji wiadomości do wielu intencji oraz to, że umożliwia używanie Rasa’y w mniej popularnych językach.

* **Zarządzanie dialogiem**

Jest to kompetencja Rasa Core. Narzędzie pozwala na dopasowanie odpowiedzi nie tylko na podstawie ostatniej wysłanej wiadomości, lecz również kontekstu wynikającego z aktualnie rozpatrywanej historii rozmowy. Wyznacza właściwą reakcję w oparciu o dostarczone historyjki i zasady.

Asystent wykorzystuje polityki podczas podejmowania decyzji, jak zareagować w kolejnym kroku rozmowy. W pliku „config.yml” możemy wybrać polityki, które chcemy wykorzystać w naszej aplikacji ( istnieje również możliwość utworzenia własnej polityki ). Przy decydowaniu o kolejnym nowym kroku w dyskusji każda ze wskazanych polityk proponuje odpowiedź chatbota wraz z poziomem pewności, że dany tekst będzie dobrze dopasowany do aktualnej rozmowy. Następnie wybierana jest propozycja o największym wskaźniku pewności dobrego dopasowania.

Przykładowe możliwe do zastosowania polityki to: zasad, zapamiętywania oraz TED.

Polityka zasad jest stosowana, gdy rozmowa pasuje do jednego ze schematów zdefiniowanych w pliku „rules.yml”.

Polityka zapamiętywania jest realizowana, kiedy obecna konwersacja pasuje do historyjki zdefiniowanej w danych treningowych. Wtedy kolejna wypowiedź bota będzie zgodna z zaprojektowanym scenariuszem.

Polityka TED, która stosuje uczenie maszynowe do przewidzenia kolejnej najlepszej reakcji asystenta bazując na historii rozmowy. Wykorzystuje się ją, gdy inne używane polityki nie są w stanie zaproponować dobrego rozwiązania.

* **Integracja**

Rasa zapewnia wiele wbudowanych rozwiązań umożliwiających podłączenia chatbota do popularnych kanałów komunikacyjnych lub strony internetowej. Na stronie firmy możemy znaleźć instrukcję np. podłączenie asystenta do Facebook Messenger’a ( <https://rasa.com/docs/rasa/connectors/facebook-messenger/> ).

Do interakcji z serwerem Rasa Open Source możemy używać http API. Funkcja ta nie jest dostępna domyślnie. Należy posłużyć się poleceniem „rasa run --enable-api”, aby skorzystać z funkcjonalności.

1. Dane treningowe

Historyjki to typ danych służący do kreowania sposobu zarządzania dialogiem przez asystenta Rasa. Wraz ze wzrostem jakości i ilości danych przekazywanych do modelu, chatbot razy jest w stanie coraz lepiej odpowiadać na wiadomości rozmówcy. Historyjki mają formę konwersacji między użytkownikiem a botem. Wypowiedzi użytkownika są wyrażane przez intencje ( kategorie, do których przyporządkowane zostały konkretne wypowiedzi ) lub encje ( ustrukturyzowane informacje, które można wydobyć z wiadomości użytkownika ). Odpowiedzi asystenta są reprezentowane przez nazwę reakcji, do której w innym pliku przyporządkowany jest właściwy tekst wyświetlany podczas rozmowy.

Do trenowania modelu zarządzania dialogiem Rasa Open Source wykorzystuje także zasady. Zasady opisują krótkie fragmenty rozmowy, które zawsze powinny potoczyć się jednakowo np. sposób zachowania chatbota po otrzymaniu wypowiedzi użytkownika scharakteryzowanej jako „przywitanie” ( nawet jeżeli zostało ono wykryte w dalszej części konwersacji ). Aby utworzyć zasadę musimy skonfigurować model dodając politykę zasad do używanych polityk. Zasadę tworzymy podając daną intencję oraz reakcję chatbota, gdy ta intencja zostanie rozpoznana. Należy pamiętać, że utworzone zasady muszą być również zapisane jako część jednej lub wielu historyjek. W innym przypadku zostaną zignorowane.

Podczas trenowania modelu nie powinno się nadużywać zasad, aby w pełni skorzystać z możliwości algorytmów uczenia maszynowego wykorzystywanego narzędzia.

1. Rozwiązania programistyczne

Rasa Open Source napisana jest w języku Python i można ją zainstalować korzystając z komend posiadanej na urządzeniu wersji tego języka. W ramach modeli językowych Rasa pozwala na skorzystanie z dwóch bibliotek:

* spaCy – darmowa biblioteka języka Python o otwartym kodzie źródłowym, przeznaczona do przetwarzania języka naturalnego. Wybierając to rozwiązanie możemy wykorzystać wcześniej wytrenowane modele językowe ( w tym dla języka polskiego ) wpisując odpowiedni identyfikator języka we właściwym miejscu. Możemy wykorzystać bibliotekę w przypadku większości komponentów. Aby z niej skorzystać należy zdefiniować język, w którym zamierzamy wytrenować chatbota.
* MITIE – biblioteka dostarczająca darmowe narzędzia wysokiej jakości do ekstrakcji informacji. Możemy skorzystać z niej dla znacznej części komponentów. MITIE może być również wykorzystana do przygotowania własnych wektorów słów z wykorzystaniem korpusu językowego np. z Wikipedii.

Wyboru bibliotek można również dokonać dla tokenizera, featurizer’a , klasyfikatora intencji, ekstraktora encji oraz komponentu dokonującego selekcji odpowiedzi.

# Zastosowania rozwiązania Rasa

1. Komercyjne przypadki użycia

Rozwiązania w technologii chatbot’a produkowane przez firmę Rasa są stosowane przez wiele organizacji z różnych gałęzi biznesowych do obsługi klientów.



Przykładem może być firma Lemonade, która wykorzystuje produkt Rasa’y, aby zdigitalizować proces zakupu ubezpieczenia przez wynajmujących oraz właścicieli domów i mieszkań oraz zautomatyzować własne wewnętrzne zadania. Utworzony został „Cooper” – wewnętrzny mózg automatyzacji, który zajmuje się i dostarcza zadania do produkcji. Cooper kontaktuje się z zespołem za pomocą platformy „Slack” i m.in. rozdysponowuje zadania członkom zespołu, przeprowadza automatyczne testy, czy wdraża aktualizacje w produkcie. Zastosowanie Cooper’a pozwoliło na szybszy rozwój organizacji, bezproblemowe wdrożenia oraz bardziej efektywne procesy, co przyczyniło się do sukcesu Lemonade.

( Więcej informacji o komercyjnych przypadkach wykorzystania platformy można znaleźć na: <https://rasa.com/customers/> . )

1. Problemy w rozwiązaniu Rasa

Budowanie modeli rozumienia języka naturalnego o wysokiej jakości jest wymagającym zadaneim. Aby tego dokonać należy zadbać o elementy takie, jak:

* zastosowanie danych pochodzących z rzeczywistych rozmów do trenowania modelu, aby bot mógł właściwie reagować na slang, synonimy i skróty,
* zbieranie danych z kolejnych rozmów z klientami i stosowanie ich do ulepszania modelu,
* wypowiedzi przyporządkowane do oddzielnych intencji nie mogą być zbyt podobne, aby asystent nie dokonywał błędnej klasyfikacji.

W wypadku błędów i problemów deweloperzy mogą skorzystać zasobów bloga lub dokumentacji. Istnieje również forum społeczności Rasa’y , na którym można napisać post o swoim kłopocie.

# Wykorzystanie narzędzia w ramach zrealizowanego projektu

Celem projektu było poszerzenie wiedzy dotyczącej zagadnień rozumienia języka naturalnego oraz sposobu budowania rozwiązań typu chatbot. Dodatkowo projekt obejmował podstawy tworzenia stron internetowych za pomocą framework’u Django. Podczas realizowanego projektu Rasa został wykorzystany jako silnik chatbota dla języka polskiego. Wybór rozwiązań Rasa’y w celu zrealizowania zadania był podyktowany jego wieloma zaletami m. in.: łatwo dostępną dokumentacją oraz porad dotyczących korzystania, brakiem kosztów związanych ze stosowaniem framework’u, możliwością stworzenia asystenta w języku polskim.

* **Jak zrealizować podobny projekt ?**

Budowę podobnego rozwiązania należy rozpocząć od przygotowania środowiska pracy. Należy pobrać i skonfigurować pakiety dla narzędzi: Anaconda ( <https://www.anaconda.com/products/distribution> ), Django ( <https://www.djangoproject.com/download/> ), Python i Rasa.( <https://www.youtube.com/watch?v=RVoFqxmG8p0> ).

( W nawiasach podane są linki pomocne podczas instalacji. )

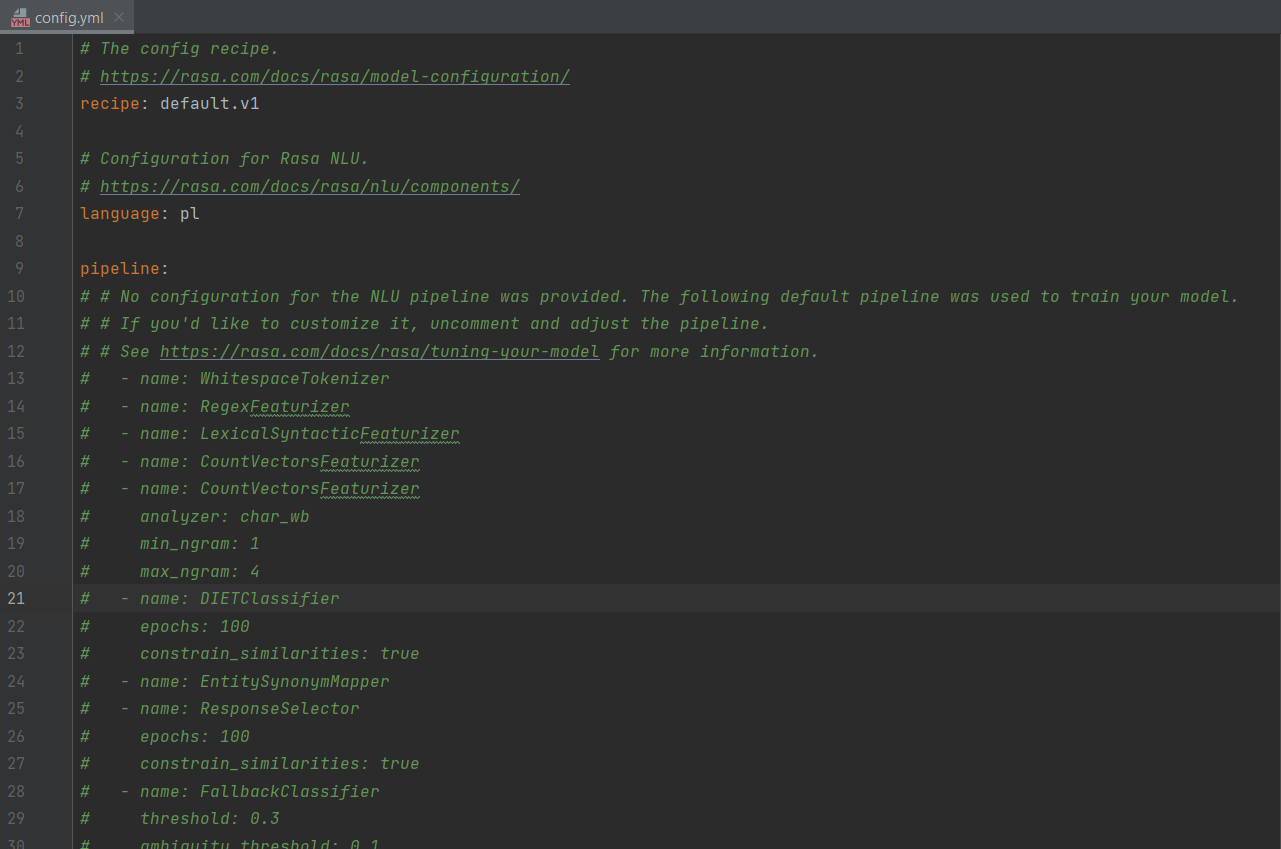
W moim przypadku niezbędne okazało się również zainstalowanie środowiska Pycharm ( <https://www.jetbrains.com/help/pycharm/installation-guide.html#silent> ) z powodu napotkanych problemów z właściwym działaniem poszczególnych komponentów projektu zarówno w systemie Windows, jak i Ubuntu.

Kolejnym krokiem było stworzenie nowego projektu oraz aplikacji Django w środowisku Pycharm. W tym celu należy utworzyć nowy projekt języka Python w katalogu przeznaczonym na projekty Pycharm, a następnie otworzyć projekt. Po uruchomieniu tworzymy nasz projekt Django w terminalu za pomocą komendy „django-admin startproject”, a wewnątrz niego zakładamy aplikację Django poleceniem „python manage.py startapp”.

Aplikacja Django jest pakietem Python’a dostosowanym do wykorzystywania w projektach Django. Zawiera pliki potrzebne do implementacji funkcjonalności na budowanej stronie internetowej zgodne z konwencją framework’u Django.

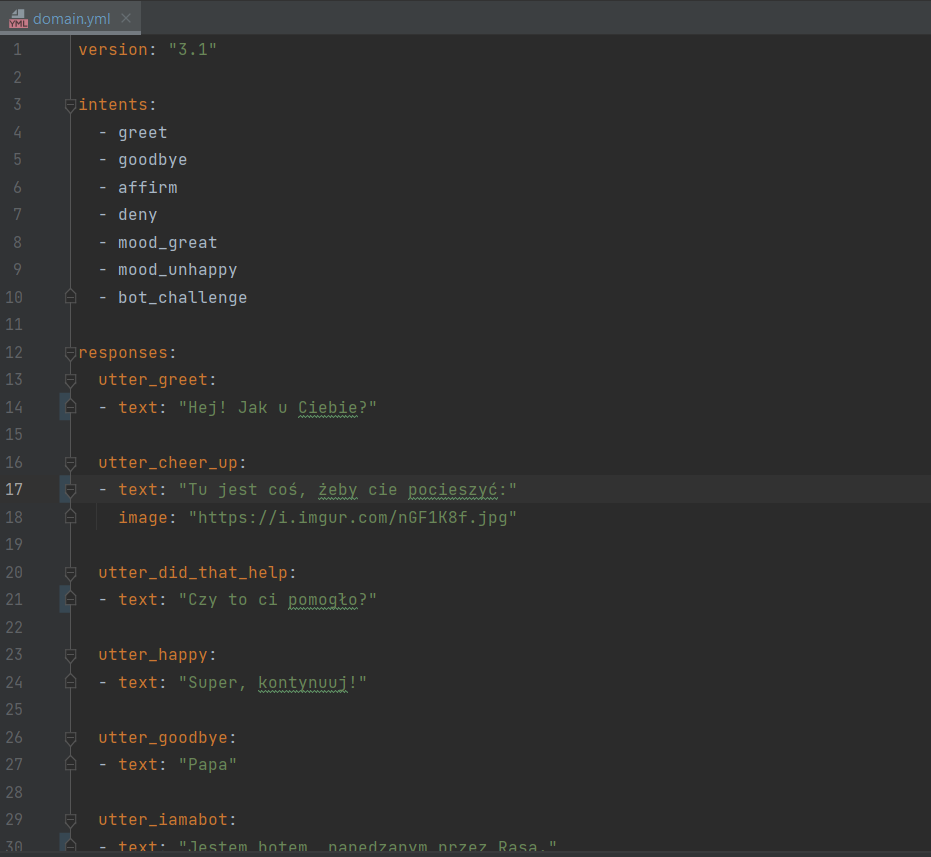
Następnym etapem zadania było uruchomienie chatbota w konsoli terminala. Pliki dotyczące chatbota ( pochodzące z instalacji ) umieściłam w tym samym katalogu, w którym znajdował się folder projektu Django. Po udanym skorzystaniu z wersji w języku angielskim, wytrenowałam modele Rasa’y w języku polskim. W tym celu zmodyfikowałam pliki: „config.yml”, „domain.yml”, „test\_stories.yml”.

W pliku „config.yml” zmodyfikowałam kod języka, dla którego przygotowywany jest model na kod języka polskiego:

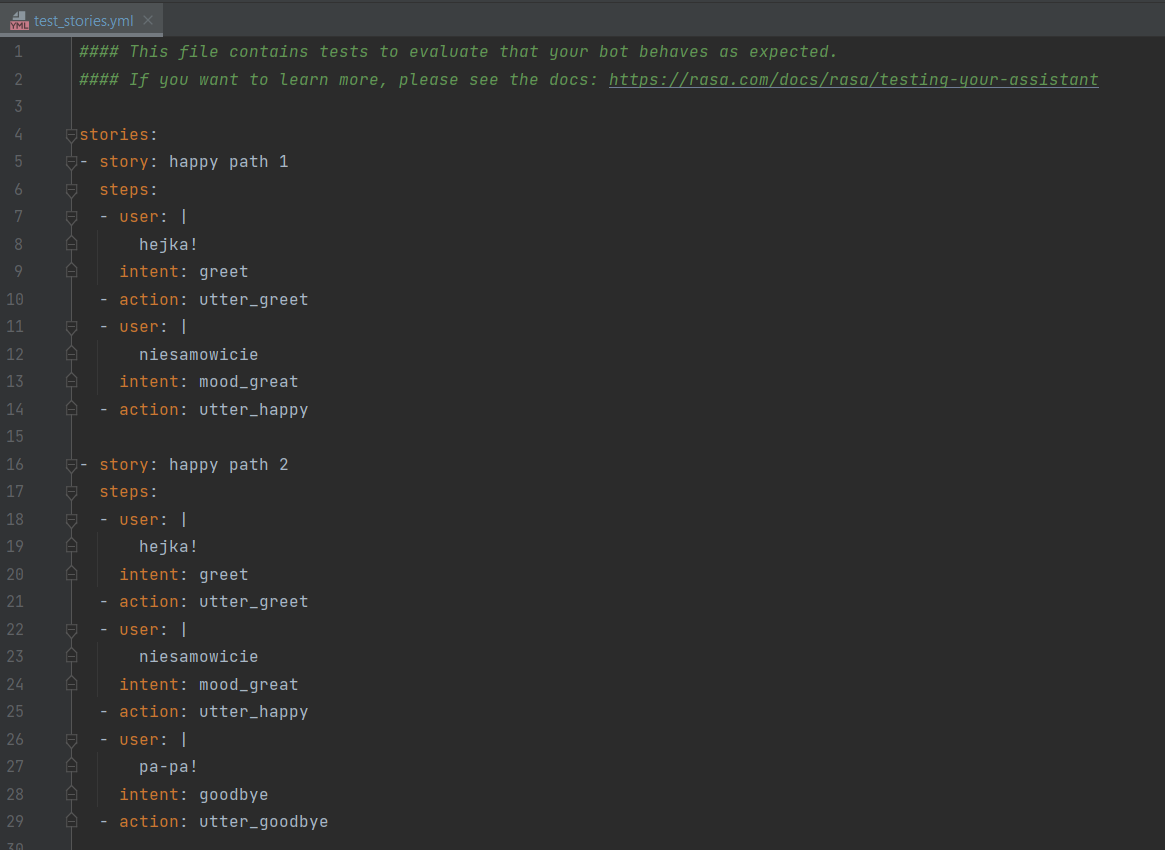


Zdjęcie 1 Zmodyfikowany plik "config.yml"

W plikach „domain.yml” oraz „test\_stories.yml” przetłumaczyłam angielskie wypowiedzi przygotowane w modelu startowym na język polski, a następnie wytrenowałam model korzystając z komendy „rasa train”. Potem przetestowałam działanie modelu w terminalu.



Zdjęcie 2 Zmodyfikowany plik "domain.yml"



Zdjęcie 3 Zmodyfikowany plik "test\_stories.yml"

Kolejną częścią projektu było nawiązanie połączenia między tworzoną stroną internetową a serwerem chatbota. Kiedy asystent Rasa’y dokonuje predykcji, serwer Rasa’y wysyła zapytanie POST do serwera akcji z ładunkiem w formacie JSON. Gdy serwer akcji zakończy przetwarzanie otrzymanych informacji zwraca odpowiedź do serwera Rasa’y, skąd można uzyskać dostęp do zawartości zwróconego pliku.

Komunikacja między stroną a serwerem odbywa się według REST API. W kodzie realizowana jest następująco:



Zdjęcie 4 Metoda POST

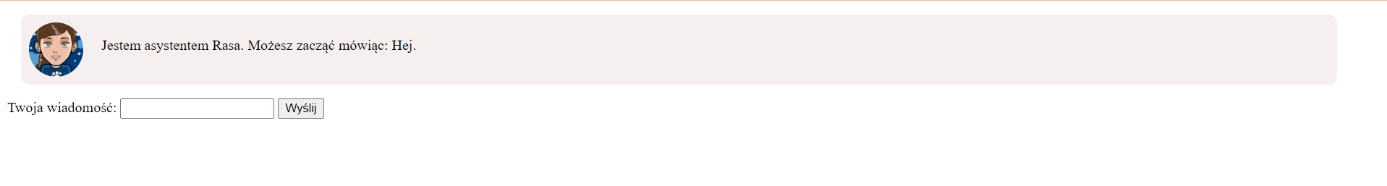
Znaczna część projektu obejmowała zapoznanie się z technologią Django, utworzenie szablonu HTML wyświetlanej strony oraz utworzenie funkcji realizującej obsługę chatu. Za początek kolejnej rozmowy uznawane jest nowe przeładowanie strony. Obiekty wypowiedzi użytkownika i chatbota posiadają pola id rozmowy (unikalne id rozmowy). Na podstawie tego atrybutu pobierane są z bazy danych wypowiedzi należące do aktualnie prowadzonej konwersacji i w szablonie HTML generowane zostają dla nich odpowiednie pola chatu.



Zdjęcie 5 Fragment szablonu HTML

Cały utworzony kod udostępniony został w repozytorium w celach edukacyjnych.

* **Efekt końcowy projektu:**



Zdjęcie 6 Widok strony po uruchomieniu



Zdjęcie 7 Rozpoczęcie rozmowy



Zdjęcie 8 Reakcja chatbota na otrzymaną wiadomość



Zdjęcie 9 Kontynuacja rozmowy

# Źródła

* <https://rasa.com/docs/rasa/>
* <https://github.com/mit-nlp/MITIE>
* <https://rasa.com/open-source>
* <https://rasa.com/blog/rasa-nlu-in-depth-part-1-intent-classification/>
* <https://rasa.com/blog/dialogue-policies-rasa-2/>
* <https://rasa.com/docs/rasa/policies/#ted-policy/>
* <https://www.turtle-techies.com/common-pitfalls-developing-a-chatbot-with-rasa-and-how-to-overcome-them/>
* <https://www.bootdey.com/img/Content/avatar/avatar3.png>
* <https://www.bootdey.com/img/Content/avatar/avatar5.png>
* <https://www.w3schools.com/howto/howto_css_chat.asp>