**Conversational AI z uwzględnieniem praw dostępu do danych na przykładzie frameworka RASA**

Dokumentacja

Autorzy:

Michał Michalski

Damian Olech

[**1.Wstęp**](#_yi34437i7a4y) **2**

[1.1 Co to jest Rasa?](#_o1x8ms219wxd) 2

[1.2 Co to jest NLU?](#_hqd8uk125z0x) 2

[1.3 Gdzie jest wykorzystywana Rasa?](#_z1e79r6eutk) 2

[**2. Instalacja pakietu Rasa w środowisku Windows**](#_2abd6zb3erum) **2**

[2.1 Przygotowanie środowiska Python](#_hkyqe42sapl) 2

[2.1 Przygotowanie środowiska wirtualnego](#_jmlb4mkl6w1s) 3

[2.2 Instalacja Rasa Open Source](#_v4maw41uwg88) 3

[2.3 Potencjalny problem](#_32nkd7nvcjbo) 3

[**3. Nowy projekt**](#_xn28ypwqayl) **4**

[**4.Trening modelu**](#_3uxqnkpx8r6b) **9**

[Wstępnie wytrenowane osadzania: klasyfikator intencji Sklearn](#_s7c2jbr8vbbv) 9

[Nadzorowane osadzania: osadzanie klasyfikatora intencji TensorFlow](#_5nqkkxkc1aqr) 10

[Zasada działania](#_873k3nfbdygb) 11

[**5.Uruchomienie**](#_dvbq51zfeuqe) **11**

[**6. Połączenie z bazą danych**](#_3ppovgjkjgud) **12**

[**7. Przykłady implementacji funkcji dla bota**](#_ahapnonk61xr) **12**

[7.1 Funkcja nie wykorzystująca sprawdzania uprawnień](#_7zb5zink0ckj) 12

[7.2 Funkcja sprawdzająca uprawnienia użytkownika](#_xjzlnfi5ott) 14

[7.3 Przekazywanie części zapytania użytkownika do funkcji](#_ic2onugz1xse) 14

[**8. Umieszczenie Bota na własnej stronie.**](#_wffdwoc88ax9) **15**

[8.1 Kanały Rest](#_vcoets6brkzc) 16

[8.2 Kanał Websocket](#_mgpfzke6cxzt) 16

[JWT Authentication](#_wycgodcd515s) 17

[8.3 Chat widget](#_9v9hktsxwtou) 17

# 1.Wstęp

## 1.1 Co to jest Rasa?

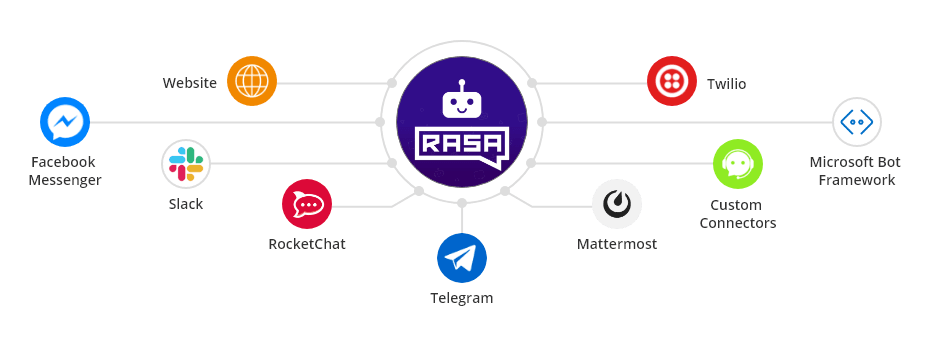
Rasa jest to framework pythona, który umożliwia nam zbudowanie dowolnego rodzaju chatbota. Bazuje on na NLU(Natural Language Processing), który oferuje możliwość zrozumienia czego chce użytkownik. Jest on podzielony na 2 części Rasa NLU i Rasa Core.

## 1.2 Co to jest NLU?

NLU (natural language understanding) jest to podtemat przetwarzania języka naturalnego w sztucznej inteligencji, który zajmuje się czytaniem maszynowym ze zrozumieniem. Rozumienie języka naturalnego jest uważane za problem trudny dla AI. Dziedzina ta cieszy się dużym zainteresowaniem komercyjnym ze względu na jego zastosowanie do automatycznego wnioskowania, tłumaczenia maszynowego, odpowiadania na pytania, gromadzenia wiadomości, kategoryzacji tekstu, aktywacji głosowej, archiwizacji i analizy treści na dużą skalę.

## 1.3 Gdzie jest wykorzystywana Rasa?

Dzięki Rasa możemy zbudować asystentów kontekstowych dla Messengera, Slack, Google Hangouts, Webex Teams, Microsoft Bot Framework, RocketChat, Mattermost, Telegram, Twilio. Swoje własne niestandardowe kanały do konwersacji jako umiejętności Alexy lub akcje Google Home.



# 2. Instalacja pakietu Rasa w środowisku Windows

## 2.1 Przygotowanie środowiska Python

Sprawdzamy czy środowisko jest już skonfigurowane

**python3 --version**

**pip3 --version**

Jeżeli te pakiety są już zainstalowane, użycie tych komend powinno wyświetlić numery wersji

Jeżeli nie, postępuj zgodnie z poniższymi instrukcjami

Upewnij się że Microsoft VC++ Compiler jest zainstalowany, aby python mógł kompilować

dowolne zależności. Możesz go wziąć z [Visual Studio](https://visualstudio.microsoft.com/visual-cpp-build-tools/). Pobierz instalator i wybierz VC++ Build tools z listy.Zainstaluj [Python 3](https://www.python.org/downloads/windows/) (64-bit version) dla Windows.

**pip3 install -U pip**

## 2.1 Przygotowanie środowiska wirtualnego

Jest to krok opcjonalny, ale rekomendowany. Domyślnie stworzenie środowiska wirtualnego za pomocą narzędzi virtualenv i virtualenvwrapper pozwala na stworzenie izolowanego środowiska dla projektu, co powoduje, że jest bardziej przejrzysty niż instalowanie pakietów dla całego systemu.

Osobiście polecam użyć condy.

Jeżeli posiadamy Conde stworzenie nowego środowisko z zainstalowanym Pythonem w odpowiedniej wersji i nazwie rasa, tworzymy za pomocą komendy

**conda create --name rasa python=3.8**

Następnie aktywujemy wirtualne środowisko za pomocą

**conda activate rasa**

## 2.2 Instalacja Rasa Open Source

Pierwsze powinniśmy się upewnić że wersja pip jest aktualna

**pip3 install -U pip**

Aby zainstalować Rasa Open Source

**pip3 install rasa**

Jeżeli nie przeszkadzają nam dodatkowe pakiety, które mogą się kiedyś przydać możemy zainstalować wszystkie dla każdej konfiguracji.

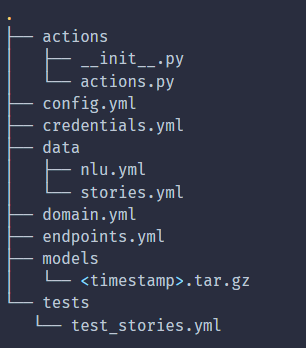
**pip3 install rasa[full]**

## 2.3 Potencjalny problem

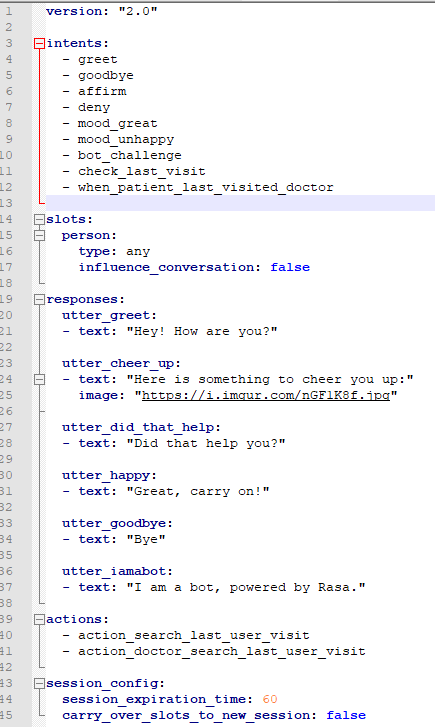
Istnieje prawdopodobieństwo, że po zainstalowaniu Rasa Open Source, nie będzie on działał prawidłowo. Bardzo prawdopodobne, że będziemy musieli cofnąć wersję niektórych pakietów zainstalowanych automatycznie.

# 3. Nowy projekt

Po wpisaniu komendy **rasa init** tworzymy nowy projekt wraz z przykładowymi danymi treningowymi, akcjami i ustawieniami, znajdują się one w następujących plikach:

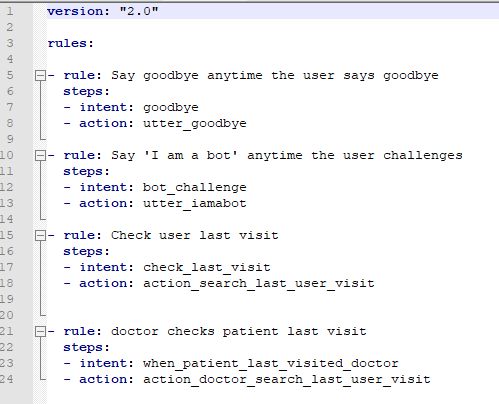


1. domain.yml



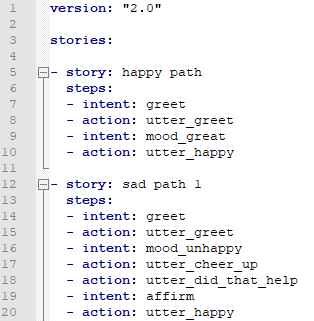
Powyższy plik zawiera listę obsługiwanych przez bota akcji; zdefiniowane przez użytkownika sloty; szybkie, jednolinijkowe zachowania typu powitanie i pożegnanie; oraz ustawienia sesji.

1. rules.yml

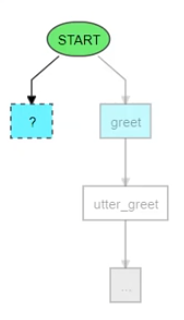


Zawiera on połączenia zamierzenia z akcją, do tego pliku odnosi się bot po odczytaniu wiadomości od użytkownika aby dobrać odpowiadającą mu odpowiedź.

1. stories.yml

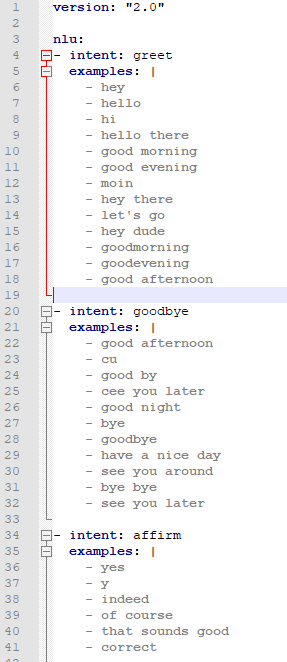


W powyższym pliku użytkownik ma możliwość zdefiniowania ogólnych scenariuszy całej interakcji z botem krok po kroku. Po ustawieniu oczekiwanych konwersacji przy użyciu komendy rasa visualize można sprawdzić ich poprawność i zgodność z zamysłem.



Powyższe zdjęcie przedstawia część wykresu dla konwersacji opartej na “happy path” .

1. nlu.yml

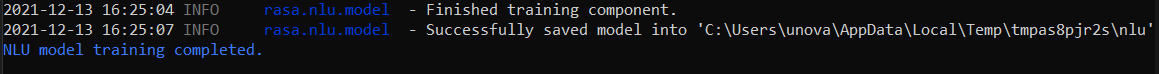


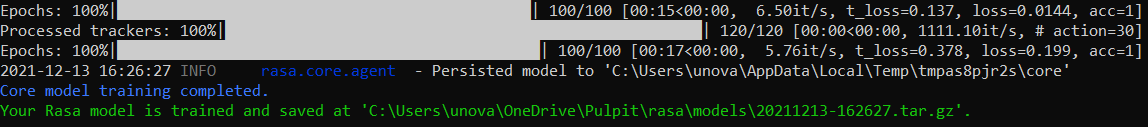
Najważniejszy plik z kompletu dostarczanego z nowym projektem, zawiera on listy słów i kawałków zdań na podstawie których bot uczy się dopasowywać otrzymaną wiadomość tekstową do ukrytego za nią założenia, jak również działając w drugą stronę, stanowi bazę potencjalnych odpowiedzi bota w wypadku gdy stwierdzi on, że jego akcją powinna być odpowiedź zawarta na liście, na przykład “greet” dla akcji utter\_greet.

# 4.Trening modelu

Po wpisaniu komendy **rasa train**, trenujemy model używając naszych NLU danych i historii (stories), zapisuje nasz model w ./models







<https://rasa.com/blog/rasa-nlu-in-depth-part-1-intent-classification/>

<https://scholar.google.com/scholar?hl=pl&as_sdt=0%2C5&q=framework+RASA&btnG=>

Rasa używa pojęcia intencji, aby opisać, jak należy kategoryzować wiadomości użytkownika. Rasa NLU będzie klasyfikować wiadomości użytkownika do jednej lub wielu intencji użytkownika. Dwa komponenty, pomiędzy którymi można wybrać to:

* Wstępnie wytrenowane osadzania ( Intent\_classifier\_sklearn )
* Nadzorowane osadzania ( Intent\_classifier\_tensorflow\_embedding )

### Wstępnie wytrenowane osadzania: klasyfikator intencji Sklearn

Ten klasyfikator używa biblioteki spaCy do załadowania wstępnie wytrenowanych modeli językowych, które następnie są używane do reprezentowania każdego słowa w wiadomości użytkownika jako osadzania słów. Osadzanie słów to wektorowe reprezentacje słów, co oznacza, że ​​każde słowo jest konwertowane na gęsty wektor numeryczny. Osadzanie słów przechwytuje semantyczne i składniowe aspekty słów. Oznacza to, że podobne słowa powinny być reprezentowane przez podobne wektory. Osadzanie słów jest specyficzne dla języka, w którym zostały przeszkolone. Dlatego należy wybrać różne modele w zależności od języka, który jest używany. W zależności od potrzeb, dostępne również są inne osadzenia słów, np . osadzenia fastText Facebooka . Rasa NLU pobiera średnią wszystkich osadzeń słów w wiadomości, a następnie przeprowadza wyszukiwanie w siatce, aby znaleźć najlepsze parametry dla klasyfikatora wektora nośnego, który klasyfikuje uśrednione osadzenia według różnych intencji. Grid Search trenuje wiele klasyfikatorów wektorów pomocniczych z różnymi konfiguracjami parametrów, a następnie wybiera najlepszą konfigurację na podstawie wyników testu.

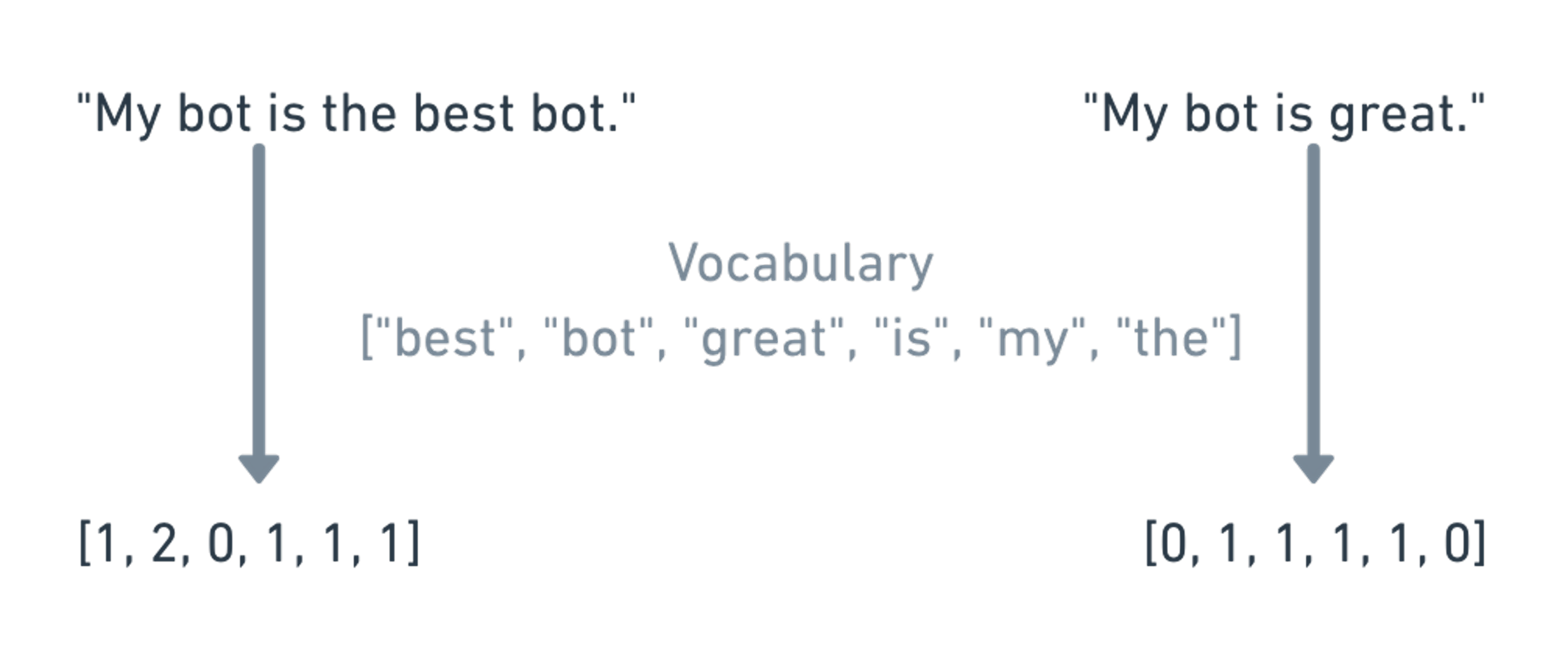
Sytuacje optymalne do użycia tego komponentu:

Kiedy używane są wstępnie wytrenowane osadzania słów, można skorzystać z najnowszych postępów badawczych w trenowaniu mocniejszych i bardziej znaczących osadzeń słów. Ponieważ osadzania są już przeszkolone, SVM wymaga tylko niewielkiego szkolenia, aby móc przewidywać pewne intencje. To sprawia, że ​​ten klasyfikator idealnie pasuje, w przypadku rozpoczęcia projektu kontekstowego asystenta AI. Nawet jeśli dostępne są tylko niewielkie ilości danych treningowych, co jest normalne w tym momencie, pozwala to otrzymać solidne wyniki klasyfikacji. Ponieważ trening nie zaczyna się od zera, trening będzie również bardzo szybki, co zapewnia krótkie czasy iteracji.

Niestety dobre osadzanie słów nie jest dostępne dla wszystkich języków, ponieważ są one w większości uczone na publicznie dostępnych zestawach danych, które są głównie w języku angielskim. Nie obejmują również słów specyficznych dla domeny, takich jak nazwy produktów lub akronimy. W takim przypadku lepiej byłoby wytrenować własne osadzania słów za pomocą nadzorowanego klasyfikatora osadzeń.

### Nadzorowane osadzania: osadzanie klasyfikatora intencji TensorFlow

Klasyfikator intencji intent\_classifier\_tensorflow\_embedding został opracowany przez Rasę i jest inspirowany artykułem starspace Facebooka . Zamiast używać wstępnie wytrenowanych osadzeń i dodatkowo trenować klasyfikatora, szkoli osadzenia słów od podstaw. Jest zwykle używany z komponentem intent\_featurizer\_count\_vectors , który zlicza, jak często różne słowa danych treningowych pojawiają się w wiadomości i udostępnia je jako dane wejściowe dla klasyfikatora intencji. Na poniższej ilustracji można zobaczyć jak różniłyby się wektory zliczania dla zdań Mój bot jest najlepszym botem i Mój bot jest świetny , np. bot pojawia się dwa razy w Mój bot jest najlepszym botem. Zamiast używać liczby słów tokenów, można również użyć liczby Ngram zmieniając pole analizatora intent\_featurizer\_count\_vectors składowych do char. To sprawia, że ​​klasyfikacja intencji jest bardziej odporna na literówki, ale także wydłuża czas szkolenia.



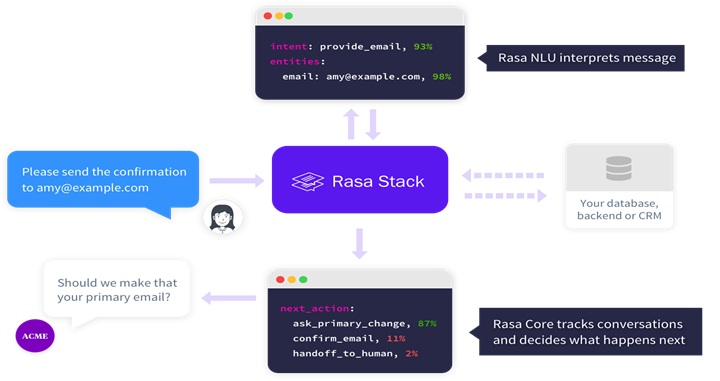
Ponadto tworzony jest inny wektor licznika dla etykiety intencji. W przeciwieństwie do klasyfikatora ze wstępnie wytrenowanymi osadzaniami słów, klasyfikator tensorflow do osadzania obsługuje również wiadomości z wieloma intencjami (np. jeśli użytkownik mówi Cześć, jaka jest pogoda? wiadomość może mieć intencje powitać i zapytać\_pogodę), co oznacza, że ​​wektor zliczania nie musi być zakodowany na gorąco. Klasyfikator uczy się oddzielnych osadzeń dla wektorów cech i intencji. Oba osadzania mają te same wymiary, co umożliwia pomiar odległości wektora między osadzonymi cechami a osadzonymi etykietami intencji przy użyciu podobieństwa cosinusów. Podczas szkolenia zmaksymalizowane jest podobieństwo cosinusowe między komunikatami użytkownika a powiązanymi etykietami intencji.

Kiedy należy używać tego komponentu:

Ponieważ ten klasyfikator uczy osadzania słów od zera, potrzebuje więcej danych szkoleniowych niż klasyfikator, który wykorzystuje wstępnie wytrenowane osadzania, aby dobrze uogólniać. Ponieważ jednak jest szkolony na danych treningowych, dostosowuje się do komunikatów specyficznych dla danej domeny, ponieważ nie ma np. brakujących osadzonych słów. Ponadto jest z natury niezależny od języka i nie jest zależny od dobrego osadzenia słów dla określonego języka. Kolejną wspaniałą cechą tego klasyfikatora jest to, że obsługuje wiadomości o wielu intencjach, jak opisano powyżej. Ogólnie rzecz biorąc, jest to bardzo elastyczny klasyfikator dla zaawansowanych przypadków użycia.

Należy zwrócić uwagę, że w niektórych językach (np. chińskim) nie jest możliwe użycie domyślnego podejścia Rasa NLU do dzielenia zdań na słowa przy użyciu białych znaków (spacji, spacji) jako separatora. W takim przypadku konieczne jest użycie innego komponentu tokenizera (np. Rasa dostarcza tokenizer Jieba dla języka chińskiego).

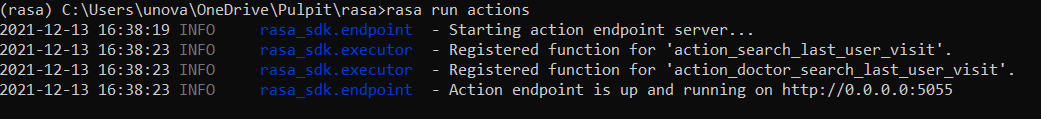
### Zasada działania



Powyższy schemat przedstawia zasadę działania bot opartego na bibliotece Rasa, po otrzymaniu wiadomości tekstowej w pierwszej kolejności konwertowana jest ona na intent przez system NLU i dopasowywana do znanych przez bota funkcji, następnie konsultowane są dostępne akcje, opcje klasyfikowane są pod względem dokładności dopasowania do udostępnionych botowi schematów rozmowy oraz adekwatności do wcześniejszej części konwersacji, na końcu opcja o najlepszym stopniu dopasowania jest zwracana użytkownikowi.

# 5.Uruchomienie

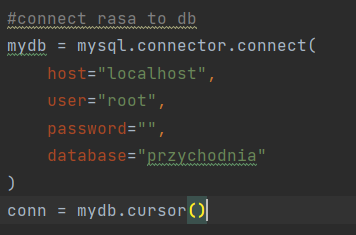
Aby uruchomić bota w konsoli należy użyć **rasa shell**, dodatkowo jeżeli posiadamy jakieś akcje to musimy uruchomić action serwer w osobnej konsoli używając **rasa run actions.**

****

Istnieje również interaktywnej nauki bota w konsoli korzystając z **rasa interactive**

# 6. Połączenie z bazą danych

Jeżeli posiadamy bazę danych z którą bot może się połączyć możemy to zrobić w pliku actions.py, który tak naprawdę umożliwia nam użycie dowolnej rzeczy, którą możemy zrobić w pythonie.



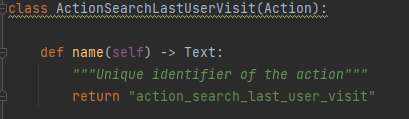
Rasa może teraz użyć w swoich akcjach połączenia z bazą, aby pobrać dane z tabel, które może odpowiednio wykorzystać w rozmowie z użytkownikiem.

# 7. Przykłady implementacji funkcji dla bota

## 7.1 Funkcja nie wykorzystująca sprawdzania uprawnień

Pierwszym krokiem przy implementacji nowej funkcjonalności bota jest stworzenie nowej klasy w pliku actions.py. Klasa musi zawierać dwie metody:

1. name - metoda ta jest wykorzystywana przez bota do odnalezienia właściwej reakcji opisanej w pliku rules.yml i nie powinna zawierać nic poza tekstem widocznym na poniższym zdjęciu

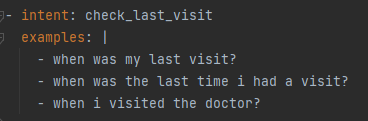


1. run - metoda ta zawiera działanie jakie ma się wykonać gdy bot zdecyduje, że zapytanie użytkownika odpowiada scenariuszowi użycia.

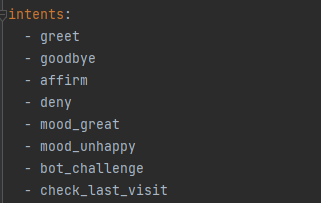


Powyższa metoda zwraca dane ostatniej wizyty pacjenta w przychodni, przyjmując userID jako dane z logowania.

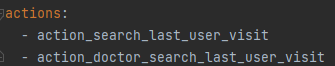
Następnym krokiem jest dodanie do pliku nlu.yml przykładowych zapytań jakie może skierować użytkownik aby uzyskać daną reakcję, nie muszą być to wszystkie możliwe opcje, ponieważ przy treningu sieć automatycznie utworzy listę podobnych zapytań, które również będą się odnosić do tej kategorii.



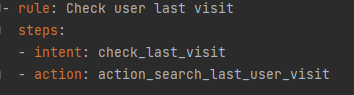
Kolejną czynnością jest dodanie intentu i akcji do list w domain.yml.



Zgodnie z opisem list intenty dodajemy do intents, natomiast akcje do actions.

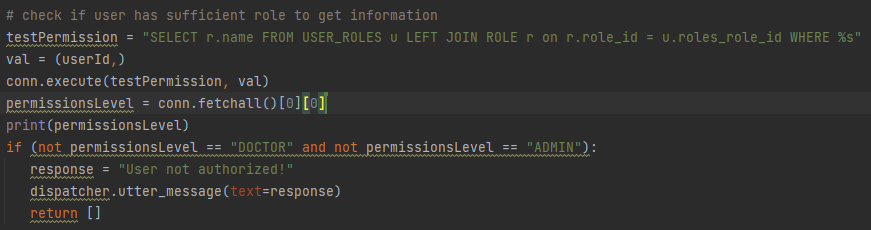


Ostatnim krokiem jest dodanie w pliku rules.yml poniższej formuły, która łączy wykryty intent z przewidzianą do niego akcją.



## 7.2 Funkcja sprawdzająca uprawnienia użytkownika

Wykrywanie poziomu uprawnień użytkownika odbywa się przy pomocy następującego fragmentu kodu dodanego to metody run klasy akcji:

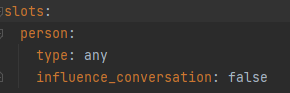


Powyższy fragment kodu bazuje na przekazaniu userID z logowania przed uruchomieniem bota.

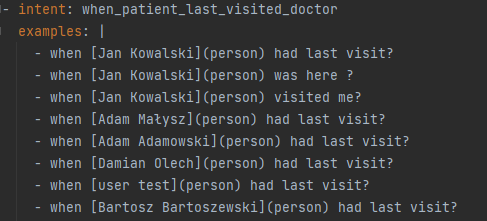
## 7.3 Przekazywanie części zapytania użytkownika do funkcji

W celu przekazania argumentów zapytania do funkcji ( na przykład gdy chcemy by bot wyszukał coś w bazie danych) musimy podjąć następujące kroki:

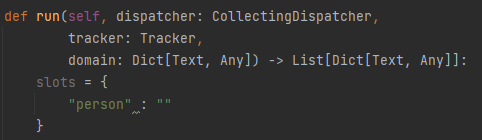
1. W pliku domain.yml deklarujemy slot który ma zostać pobrany z zapytania użytkownika:



1. Deklarując przykładowe zapytania użytkownika w pliku nlu.yml dodajemy w nawiasach kwadratowych przykładowe dane wejściowe które mogą zostać podane, w tym wypadku imię i nazwisko, następnie po zamknięciu nawiasu dodajemy w nawiasach okrągłych typ slota którego oczekujemy.



1. W metodzie, która potrzebuje otrzymać input od użytkownika najpierw dodajemy informacje o slocie:



a następnie wywołujemy funkcję tracker.get\_slot w następujący sposób:



W tym wypadku z zapytania wyciągamy imię i nazwisko osoby o którą pyta użytkownik. 



# 8. Umieszczenie Bota na własnej stronie.

Jeżeli mamy już istniejącą witrynę internetową i chcemy dodać do niej asystenta Rasa, możemy użyć widgetu Rasa Chat Widget, który możemy dodać do istniejącej strony internetowej, dodając fragment kodu HTML. Alternatywnie możemy także zbudować własny widget czatu.

## 8.1 Kanały Rest

Kanały RestInput i CallbackInput mogą być używane do niestandardowych integracji. Zapewniają adres URL, pod którym można publikować wiadomości i odbierać wiadomości z odpowiedziami bezpośrednio lub asynchronicznie za pośrednictwem webhooka.

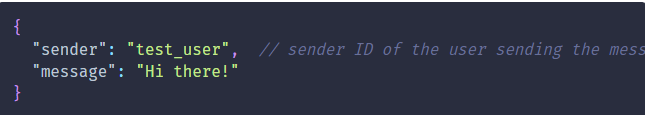
8.1.1 RestInput

Kanał REST zapewnia punkt końcowy REST, w którym można publikować wiadomości i odbierać wiadomości asystenta w odpowiedzi. Kanał Rest dodajemy do pliku credentials.yml

Po restarcie serwera możemy odbierać i wysyłać wiadomości do http://<host>:<port>/webhooks/rest/webhook zastępując host i port odpowiednimi wartościami z naszego serwera Rasa.

8.1.2 Żądania i odpowiedzi

Po udostępnieniu kanału wejściowego, możemy wysyłać POST wiadomości w następującym formacie:

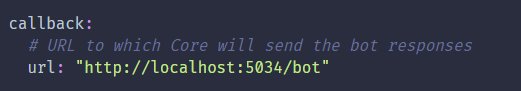


Odpowiedź z Rasa będzie treścią JSON odpowiedzi bota, na przykład:



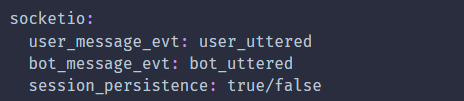
8.1.3 CallbackInput

Kanał wywołania zwrotnego zachowuje się bardzo podobnie do kanału REST, ale zamiast bezpośrednio zwracać komunikaty botów do żądania HTTP, które wysyła komunikat, wywoła adres URL, który można określić, aby wysyłać komunikaty botów. Aby skorzystać z tego kanału potrzebujemy dodać dane uwierzytelniające do pliku credentials.yml:



## 8.2 Kanał Websocket

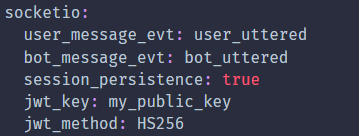
Kanał SocketIO używa gniazd sieciowych i działa w czasie rzeczywistym. Aby użyć kanału SocketIO, musimy dodać poświadczenia w pliku credentials.yml:



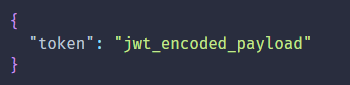
Pierwsze dwie wartości konfiguracyjne definiują nazwy zdarzeń używane przez Rasa Open Source podczas wysyłania lub odbierania wiadomości przez socket.io. Po ponownym uruchomieniu serwera możemy wysyłać wiadomości do http://<host>:<port>/socket.io, zastępując host i port odpowiednimi wartościami z działającego serwera.

## JWT Authentication

Kanał SocketIO można opcjonalnie skonfigurować do przeprowadzania uwierzytelniania JWT podczas połączenia, definiując jwt\_key i opcjonalnie jwt\_method w pliku credentials.yml

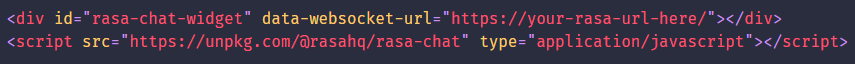


Podczas początkowego żądania połączenia klient powinien przekazać zakodowany ładunek jako obiekt JSON pod tokenem klucza:



## 8.3 Chat widget

Po skonfigurowaniu kanału SocketIO możemy skorzystać z oficjalnego widgetu Rasa Chat Widget na dowolnej stronie internetowej. Po wklejeniu poniższego kodu HTML do swojej witryny i dodaniu adresu URL instancji Rasa do atrybutu websocket-url attribute



Opcjonalnie dla aplikacji React, w repozytorium pakietów NPM istnieje odpowiednia biblioteka.

# 9. Podsumowanie

Framework Rasa to nowe (firma założona w 2016 roku , w grudniu wyszła wersja 3.x) narzędzie do konstruowania chatbotów. Ogromną wadą była dla mnie sama instalacja oprogramowania, ponieważ musiałem cofnąć wersję kilku pakietów, po domyślnej instalacji, żeby w ogóle uruchomić poprawnie projekt(ale była to jeszcze wersja 2.x, nie wiem jak teraz). Zaletą wydaje się duże zapotrzebowanie na produkty, które mogą maszynowo zrozumieć tekst w różnych dziedzinach(np. wprowadzenie takich botów w urzędach), jak również łatwość dodawania nowych funkcjonalności / modyfikowania już istniejących. Kolejną dobrą stroną jest bardzo duża baza słów rozpoznawanych przez sieć oraz dokładność z jaką klasyfikuje on nawet niedokładne wiadomości do intencji jakie miał użytkownik pisząc do bota. Rasa ma również pomocne narzędzie(rasa interactive), która podczas rozmowy z botem informuje nas z jaką intencją bot odczytał naszą wiadomość, co możemy potwierdzić lub zaprzeczyć co tworzy nowe dane na których model może się uczyć.