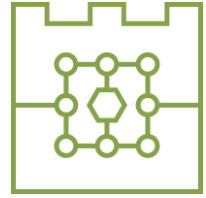




POLITECHNIKA KRAKOWSKA im.
Tadeusza Kościuszki
Wydział Informatyki i Telekomunikacji



Dawid Jopek

Numer albumu: 134206

**Projekt i implementacja czat bota do rozmów o statystykach piłkarskich z
wbudowaną analizą danych**

**Design and implementation of a chat bot for talking about football statistics
with built-in data analysis**

PRACA INŻYNIERSKA
na kierunku Informatyka

Praca wykonana pod kierunkiem:
Dr Radosław Kycia

Kraków, 2024

SPIS TREŚCI

1. WSTĘP	5
1.1. Cel pracy	5
1.2. Zakres pracy	5
2. CZATBOTY	6
2.1. Teoria potrzebna do konstrukcji czatbotów	6
2.2. Rozwój czatbotów	6
2.3. Typy, zastosowania i implementacje czatbotów	7
3. SZTUCZNA INTELIGENCJA W CZATBOTACH	11
3.1. Wprowadzenie do sztucznej inteligencji	11
3.2. Technologie i metody sztucznej inteligencji wykorzystywane w czatbotach	11
3.3. Wykorzystanie sztucznej inteligencji na jakość interakcji z czatbotem	17
3.4. Przykłady zastosowań czatbotów do celów komercyjnych	18
4. PROJEKT APLIKACJI CZATBOTA	19
4.1. Wymagania funkcjonalne aplikacji	19
4.2. Opis funkcjonalności	19
4.3. Struktura aplikacji oraz użyte technologie	20
4.4. Biblioteka ChatterBot	21
4.4.1. Wprowadzenie do biblioteki ChatterBot	21
4.4.2. Architektura ChatterBot	21
4.4.3. Baza danych czatbota	22
4.4.4. Trening czatbota	22
4.4.5. Zasada działania biblioteki Chatterbot	22
4.5. Analiza danych piłkarskich	22
4.5.1. Wykorzystane API i sposób pobierania danych	22
4.5.2. Przygotowanie danych do analizy	23
4.6. Przechowywanie danych	25
5. IMPLEMENTACJA CZATBOTA	26
5.1. Tworzenie czatbota	26
5.2. Moduły czatbota	27

5.2.1. Moduły pozyskujące informacje do bazy danych	27
5.2.2. Moduły użytkowe	29
5.2.3. Moduły służące do zwracania danych dla adaptera.....	31
5.3. Adaptery czatbota	35
5.4. Trenowanie modeli czatbota.....	39
5.4.1. Przygotowanie danych treningowych.....	40
5.4.2. Programy do trenowania modeli.....	41
5.5. Testowanie aplikacji	44
6. PODSUMOWANIE	47
BIBLIOGRAFIA	48
SPIS RYSUNKÓW	50
SPIS LISTINGÓW	51

STRESZCZENIE

Celem pracy jest realizacja projektu czatbota. W pierwszej kolejności omówiony został dział informatyki, jakim są czatboty. W dalszej części pracy została opisana sztuczna inteligencja wraz z metodami przetwarzania języka naturalnego (tokenizacja słów, wektory TF-IDF). Następnie zaprezentowany został projekt aplikacji zawierający wymagania potrzebne do jej prawidłowego funkcjonowania. Aplikacja została napisana w języku Python z głównym wykorzystaniem bibliotek Chatterbot i spaCy oraz Football-API do pozyskiwania danych o piłce nożnej. Ostatnim etapem jest implementacja aplikacji. W tej części opisane zostały elementy składowe czatbota (moduły, adaptory, trenowane modele). Odpowiednie wytrenowanie modeli pozwoliło na lepsze działanie bota w generowaniu odpowiedzi dla użytkownika. Zaletą aplikacji jest łatwa możliwość rozbudowy o dodatkowe funkcje czy modele językowe.

ABSTRACT

The aim of the work is to implement a chatbot project. First, the branch of computer science, which is chat bots, was discussed. In the next part of the thesis, artificial intelligence was described along with methods of natural language processing (word tokenization, TF-IDF vectors). Then the design of the application was presented, including the requirements needed for its proper functioning. The application was written in Python with the main use of Chatterbot and spaCy libraries, as well as Football-API for acquiring soccer data. The final step is the implementation of the application. In this section, the components of the chatbot (modules, adapters, trained models) are described. Proper training of the models allowed the bot to perform better in generating responses for the user. The advantage of the application is that it can be easily extended with additional features or language models.

1. WSTĘP

1.1. Cel pracy

Niniejsza praca inżynierska ma na celu, realizację projektu czatbota do rozmów o statystykach piłkarskich oraz jego implementację. W pracy zostanie przedstawiona gałąź informatyki, jaką jest przetwarzanie języka naturalnego. Elementem takiego przetwarzania są czatboty. Opiszę teorię związaną z przetwarzaniem języka naturalnego oraz wykorzystanie metod przetwarzania języka naturalnego w interakcji z użytkownikiem. Jest to dosyć złożone zagadnienie, ponieważ nauczanie maszyn języka ludzi jest trudne ze względu na naturę języka ludzkiego i na to jakich reguł używamy podczas komunikacji [1]. Opisana teoria została wykorzystana przy tworzeniu czatbota i zostanie zaprezentowana na przykładzie aplikacji.

1.2. Zakres pracy

Praca zaczyna od opisu, czym są czatboty wraz z ich zastosowaniem w życiu codziennym, dodając przykłady gotowych czatbotów, które zostały zaimplementowane w konkretnym celu. Następnie skupi się na ważnym zagadnieniu, jakim jest sztuczna inteligencja, która jest mocno związana z czatbotami. Przetwarzanie języka naturalnego to termin, który zostanie poszerzony ze względu na silne wykorzystanie w tworzeniu czatbotów. Na kolejnym etapie zostanie przedstawiony projekt aplikacji czatbota do rozmów o statystykach piłkarskich, wraz z funkcjonalnościami, strukturą aplikacji oraz użytą technologią (Python, Chatterbot, Football-API). Jednym z ostatnich elementów pracy będzie implementacja właściwej aplikacji czatbota.

2. CZATBOTY

2.1. Teoria potrzebna do konstrukcji chatbotów

Zaczynając omawianie tematu, jakim są boty, należy przybliżyć, postać Alana Turinga, który zapoczątkował pomysł na to, żeby komputer, był w stanie naśladować człowieka w komunikacji tekstowej. Gałąź informatyki, jaką są boty, rozpoczęła się właśnie od niego. Następnie stworzył on test zwany testem Turinga, w którym użytkownik miał za zadanie domyślić się czy rozmawia z maszyną, czy z prawdziwym człowiekiem. Gdy człowiek nie potrafił wskazać, że rozmawia z komputerem, to ten przechodził test pozytywnie. Test ten wykorzystywany jest do dzisiaj [2].

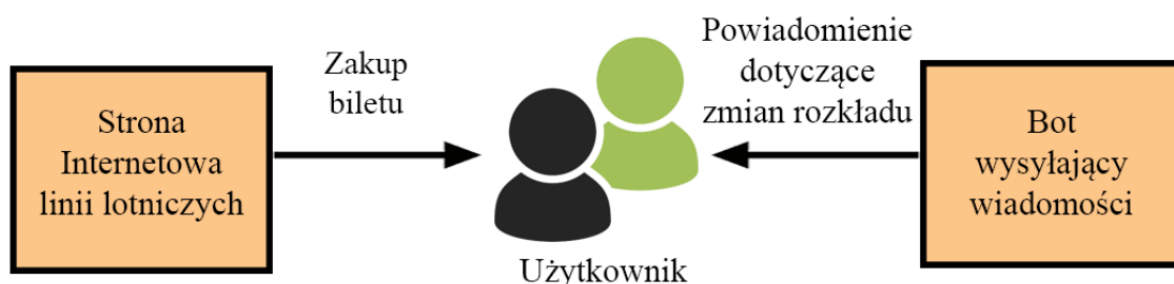
Boty to imitacja człowieka jako wirtualnego rozmówcy. Powstały po to, aby go zastępować w różnych dziedzinach życia. Działają jako interfejs do usługi. W zależności od tego, do czego bot służy, ma pomagać użytkownikowi. Chatboty komunikują się w sposób naturalny dla człowieka, dlatego w niektórych przypadkach są w stanie go zastąpić. Wykorzystują wiadomości tekstowe lub głosowe do przekazania informacji, w sposób bardzo podobny do ludzi [3].

2.2. Rozwój chatbotów

Rozwój technologii i nowe sposoby tworzenia oprogramowania, wraz ze zmianą, w jaki sposób użytkownicy używają aplikacji, pozwolił na wzrost popularyzacji chatbotów. Ludzie aktualnie skupiają się na mobilnej technologii, dlatego twórcy oprogramowania zaczynają kierować siły swoich pracowników właśnie w ten segment. Stopniowo zaczęto przechodzić z interfejsów webowych na mobilne a interfejsom mobilnym opartych na powiadomieniach bliżej do chatbotów. Najczęściej wykorzystywane aplikacje służą do wymiany informacji i do rozmowy. Przez mocną popularyzację smartfonów oraz łatwy dostęp do Internetu, wymiana wiadomości w ten sposób stała się najprostszym środkiem komunikacji. Boty to kolejny etap rozwoju komunikacji użytkownika z maszyną. Do chatbota możemy skierować pytanie i otrzymamy szybką odpowiedź, w innych interfejsach odpowiedzi musimy szukać sami lub jest ona niewystarczająca [4, 5].

Należy pamiętać, że boty nie wyprą całkowicie innych interfejsów, ponieważ każdy z nich jest w jakimś segmencie najlepszy. Interfejs webowy będzie lepszy od mobilnego poprzez dużo mniejsze ograniczenie przestrzeni roboczej, aplikacje mogą być bardziej rozbudowane i nigdy ich odpowiedniki mobilne nie będą miały, aż tak rozbudowanych funkcji. Chatboty nigdy nie zastąpią

segmentu mobilnego ponieważ nie oferują one użytkownikowi kontroli nad aplikacją. Użytkownik dzięki funkcjom interfejsu mobilnego może szybko modyfikować ustawienia aplikacji. Czatboty opierają się na odpowiedziach na pytania, co ogranicza możliwości konfiguracji aplikacji. Jednak każdy z tych interfejsów może współpracować ze sobą i tworzyć najlepsze doświadczenie dla użytkownika. Przykład takiej współpracy został przedstawiony na rysunku numer 1 [3].



Rys. 1 Współpraca interfejsów

(źródło: opracowanie własne na podstawie [3])

2.3. Typy, zastosowania i implementacje czatbotów

Czatboty, podobnie jak w innych obszarach informatyki są ograniczone jedynie oczekiwaniami użytkowników, oraz tym, co będzie możliwe w stworzeniu przez programistę. Dlatego podziałów czatbotów jest całe mnóstwo. Mogą różnić się w zależności od tego, jaki główny zamysł weźmiemy przy tworzeniu takiego podziału [8].

Podział, który zaproponował Digital Deloitte [5]:

- **czatboty informacyjne** - jeden z prostszych, popularnych typów, wskazuje odpowiedzi na najczęściej powtarzające się pytania,
- **czatboty transakcyjne** - ten rodzaj pozwala konsumentowi np. zarezerwować seans w kinie,
- **czatboty doradcze** - doskonalą się wraz z nowymi rozmowami z różnymi ludźmi, służą jako serwisy do dopasowywania treści.

Podział zaproponowany przez Amir Shevat [3]:

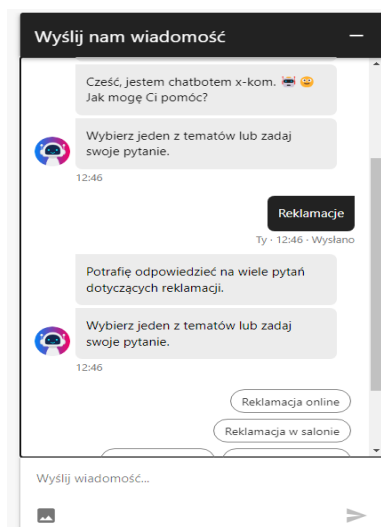
- **czatbot prywatny** - bot który skupia się na jednym użytkowniku. Użytkownik komunikuje się bezpośrednio z tym botem,
- **czatbot zespołowy** - ten rodzaj bota koordynuje działania zespołu. Bot ten powinien komunikować się z różnymi użytkownikami na różne sposoby. Powinien móc wysyłać

wiadomości grupowe, jak i prywatne. Przez wzgląd na komunikację z kilkoma użytkownikami są trudniejsze w implementacji,

- **super czatboty** - boty te łączą w sobie możliwość korzystania z wielu usług które, bot w sobie łączy. Na przykład asystent, który zaprojektowała firma Google. Ten czatbot ma do zaoferowania wiele różnych możliwości, od korzystania z serwisu, jakim jest Wikipedia po odczyt aktualnych warunków atmosferycznych,
- **chatbot o określonej domenie** - bot skupia się na jednej usłudze, którą przedstawia. Bot powinien z tą usługą identyfikować, przez projekt związany z usługą. Na przykład bot do zamawiania jedzenia z restauracji,
- **chatbot tekstowy** - te czatboty występują najczęściej w komunikatorach mobilnych. Współpracują z komunikatorem tak, aby w jego obrębie umożliwić użytkownikowi dostęp do określonych funkcji,
- **chatbot głosowy** - zamiast używać tekstu do komunikacji z użytkownikiem, czatboty wyposażone są w obsługę głosu, za jego pośrednictwem wymieniają informację z użytkownikiem.

Czatboty przez ich różnorodność potrafią znaleźć zastosowanie w przeróżnych gałęziach aktywności człowieka. Dopasowanie odpowiedniego typu czatbota do odpowiedniej dziedziny jest kluczowe, jeżeli chcemy, aby czatbot spełniał swoje zadanie, przynosił zysk i stawał się coraz bardziej popularny. Dobrze dobrany czatbot powinien dawać użytkownikowi dodatkowe możliwości interakcji z aplikacją. Dzięki temu zadowolony użytkownik, będzie korzystać tego narzędzia częściej. Chatboty znajdują zastosowanie w różnych dziedzinach [5]:

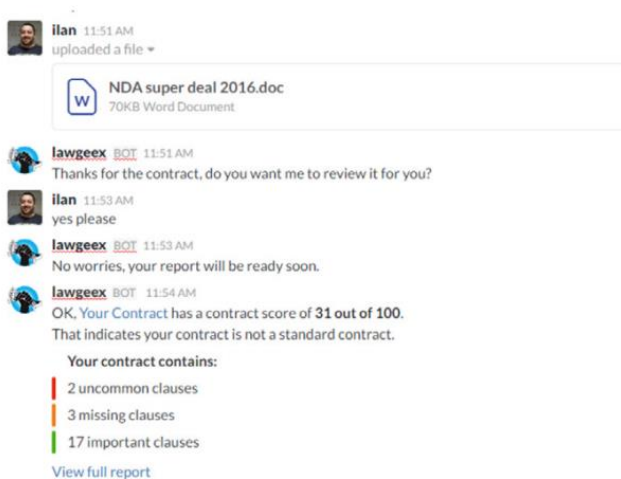
- **chatboty do sprzedaży** - wiele sklepów Internetowych korzysta z usług czatbotów. Takie programy mogą pomóc potencjalnemu klientowi w zakupie lub reklamacji. Taka praktyka pozwala odciążyć pracowników, którzy są dostępni jedynie w określonych godzinach pracy. Bot może pracować bez przerwy,



Rys. 2 Rozmowa z czatbotem x-kom

(źródło: opracowanie własne)

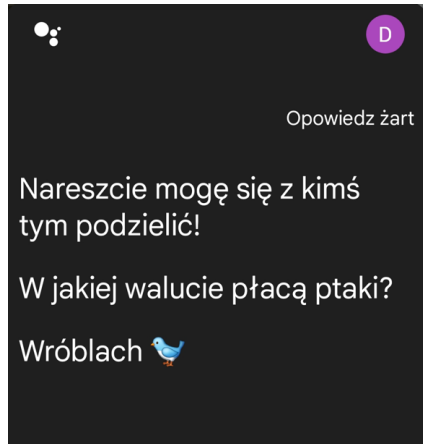
- **chatboty do obsługi klienta** - ten rodzaj czatbotów pozwala rozwiązać powtarzające się problemy użytkowników bez ingerencji człowieka. Takie chatboty mogą być połączone z chatbotami do sprzedaży. Występują najczęściej na stronach sklepów Internetowych,
- **chatboty do biznesu** - firmy coraz częściej używają czatbotów do lepszego zorganizowania przepływu procesów biznesowych. Na przykład chatbot LawGeex, pomaga analizować umowy i wyświetla informacje na jej temat,



Rys. 3 LawGeex - bot będący radcą prawnym

(źródło: [3])

- **czatboty do usług rozrywkowych** - część z botów została zaprojektowana, aby umilić użytkownikom spędzanie czasu. Boty mogą opowiadać dowcipy, ciekawostki lub działać jako proste gry komputerowe.



Rys. 4 Google Assistant opowiadający żart
(źródło: opracowanie własne)

3. SZTUCZNA INTELIGENCJA W CZATBOTACH

3.1. Wprowadzenie do sztucznej inteligencji

Sztuczna Inteligencja (SI), jest obszarem nauki wywołującym długoletnie dyskusje. Zdolność maszyn do samodzielnego myślenia takiego jak u ludzi jest tematem wielu spekulacji. Mimo upływu lat i licznych dyskusji to pytanie pozostaje wciąż otwarte. Wyrafinowane zagadnienia życia takie jak miłość, czy moralne wybory nie zostały do tej pory osiągnięte przez maszyny, które mogą podjąć decyzje jedynie na podstawie danych, na których się uczą i algorytmach, na których operują. Inteligencje opisać można jako możliwość do rozumienia zagadnień i nauki nowych rzeczy, ale również jako umiejętność do rozumienia świata i myślenia. Zwykle w takich momentach powraca test Turinga, o którym pisałem we wcześniejszym rozdziale [1, 6]. Celem tej pracy nie jest rozstrzyganie czy maszyny potrafią myśleć, a to żeby wykorzystać metody sztucznej inteligencji w zaprogramowaniu aplikacji, która będzie udawała rozmówcę, będzie czatbotem odpowiadającym na pytania z dziedziny sportu, jakim jest piłka nożna.

3.2. Technologie i metody sztucznej inteligencji wykorzystywane w czatbotach

Przetwarzanie języka naturalnego (NLP) jest dziedziną nauki, która w głównej mierze skupia się na umożliwieniu komputerom zrozumienia, interpretowania oraz reagowania na język ludzi w sposób dający realną wartość dla człowieka. NLP używane jest w czatbotach do przetwarzania i analizowania danych wejściowych. Język ludzi i komputerów znacząco się różni. Komputer wykonuje zadania obliczeniowe przy przetwarzaniu liczb. Człowiek zwykle nie działa w ten sposób. NLP pozwala komputerowi, zrozumieć to, co człowiek chciał mu przekazać. Języka naturalnego (na przykład języka angielskiego), nie da się przełożyć na zbiór skończonych matematycznych zagadnień. W przeciwieństwie do języka programowania, który został specjalnie zaprojektowany tak aby maszyna go rozumiała. Taki język był projektowany pod maszynę. Języki naturalnie nie zostały stworzone dla komputerów tylko dla ludzi. Komputer posługując się między innymi kompilatorami, jest w stanie przetworzyć języki programowania. Dzięki temu jest w stanie wykonać określone zadanie. Kompilatory dla języków naturalnych nie występują, nie oznacza to, że maszyna nie jest w stanie zrozumieć naszej mowy, potrzeba do tego innego podejścia. Najprostszym wytłumaczeniem idei NLP jest wyobrażenie sobie komputera, który przy pomocy tłumacza, zamienia tekst naturalny na zrozumiałe dla niego dane, najczęściej liczby. Komputer

dzięki takim danym jest w stanie „zrozumieć” intencję użytkownika. Języki programowania zostały stworzone tak, aby móc je interpretować w jeden poprawny sposób. Język naturalny może być poprawny na wiele różnych sposobów. Komputer nie przetworzy danych języka naturalnego na liczby w idealny sposób, ale jest w stanie „wyłapać” określone intencje i informacje, które można odpowiednio kategoryzować tak, aby wykonać na nich określone działania. Uzyskane informacje zwrotne są przetwarzane dla użytkownika. Dla przykładu w aplikacji czatbota do rozmów o statystykach piłkarskich czatbot zrozumie, że użytkownik zapyta o piłkarza i wykryje odpowiednio zamiar użytkownika w celu przekazania mu odpowiednich informacji [1, 7].

Tokenizacja słów jest procesem dzielenia tekstu języka naturalnego na pojedyncze słowa i tokeny. W przetwarzaniu języka naturalnego stosowana jest zwykle na samym początku. Kluczową rolą tokenizacji jest przekształcanie tekstu w strukturę numeryczną. Taka struktura jest już odpowiednia i prawidłowa dla uczenia maszynowego. Działanie tego procesu można przedstawić jako pewien rodzaju sortownik, który dzieli tekst na osobne elementy, takie jak pojedyncze słowa, które mogą zostać zliczone i wykorzystane do przedstawienia danych w postaci wektora liczbowego. Takie wystąpienia tokenów mogą zostać użyte w przeróżnych zadaniach przetwarzania języka naturalnego. Uprozczone działanie tokenizacji jest widoczne na rysunku numer 5 [7].



Rys. 5 Uproszczony schemat działania tokenizacji

(źródło: opracowanie własne na podstawie [7])

Przykłady niektórych zastosowań tokenizacji słów [7]:

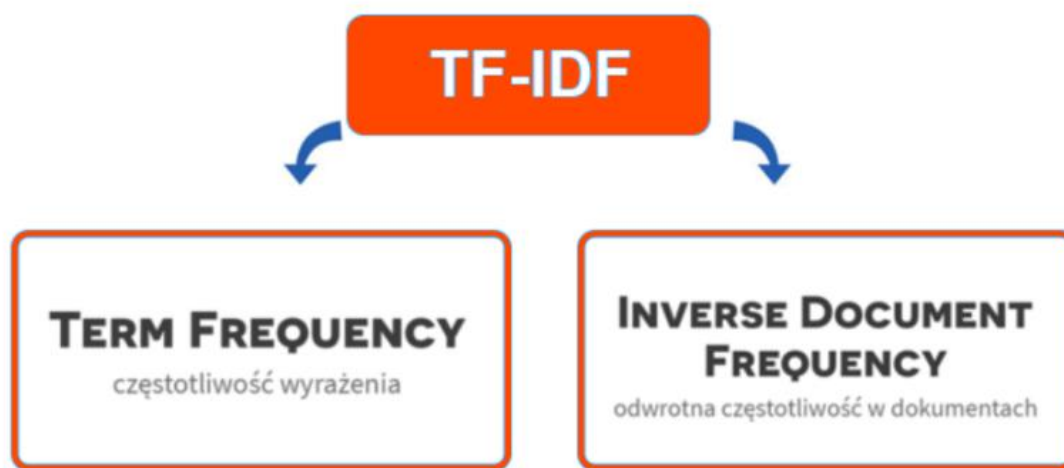
- przetwarzanie wstępne - tokenizacja jako początkowy etap wstępnego przetwarzania języka danych tekstowych, dzieli tekst na tokeny tak, aby stał się łatwiejszy do analizy,
- ekstrakcja cech - aby móc zastosować techniki uczenia maszynowego na tekście można wykorzystać tokenizację do konwersji danych tekstowych na cechy numeryczne, które mogą zostać użyte dla odpowiednich algorytmów,
- reprezentacja wektorowa - po wykonaniu tokenizacji tekst można pokazać jako wektor. Każdy wymiar wektora odpowiada konkretnemu tokenowi. Wartości są reprezentowane przez występowanie lub częstotliwość tokenu w tekście. Dzięki takiemu zabiegowi można użyć algorytmów uczenia maszynowego. Tokenizacja umożliwia nam również tworzenie wektorów zbioru słów, które znajdują zastosowanie w przeszukiwaniu i analizowaniu dokumentów. Token staje się cechą wektora, a badanie występowania tokenów wykorzystuje się do dopasowywania i kategoryzowania dokumentów,
- rozumienie kontekstu - użycie tokenizacji pomaga w zrozumieniu kontekstu złożonych zdań, przez działanie tokenizacji, jakim jest dzielenie tekstu na mniejsze fragmenty. Takie fragmenty mogą zostać przeanalizowane pod kątem rozpoznawania nazwanych jednostek (encji), znakowania części mowy oraz analizy tonacji.

Dla lepszego zrozumienia kolejnego zagadnienia, jakim są wektory TF-IDF, przytoczę często występujące pojęcia [7]:

- dokument - to dany rodzaj tekstu, który chcemy przeanalizować.
- korpus - jest zbiorem dokumentów, do którego będziemy porównywać wybrany przez nas dokument.

Wektory TF-IDF (ang. Term Frequency-Inverse Document Frequency) używane są do przeszukiwania wyznaczonych dokumentów w korpusie. Takie przeszukiwanie pozwala na pomiar ważności słów w każdym dokumencie i w całym korpusie. Wartość takiej ważności używa się do identyfikacji odpowiednich dokumentów, opierając się na ważności słów kluczowych w każdym z dokumentów. Wektory TF-IDF tworzy się poprzez zliczanie częstotliwości pojęć (tokenów) w dokumencie. Następnie normalizuje się ich pojawienia, biorąc pod uwagę ich częstotliwość występowania w całym korpusie. Wynika z tego pewna zależność, jeżeli słowa, które są

powszechne w konkretnym dokumencie, będą rzadkie w całym korpusie, to ich wartość TF-IDF będzie wyższa. Oznacza to ich większe znaczenie dla konkretnego dokumentu. Wektory TF-IDF posiadają dwie składowe, które zostały przedstawione na rysunku numer 6 [7].

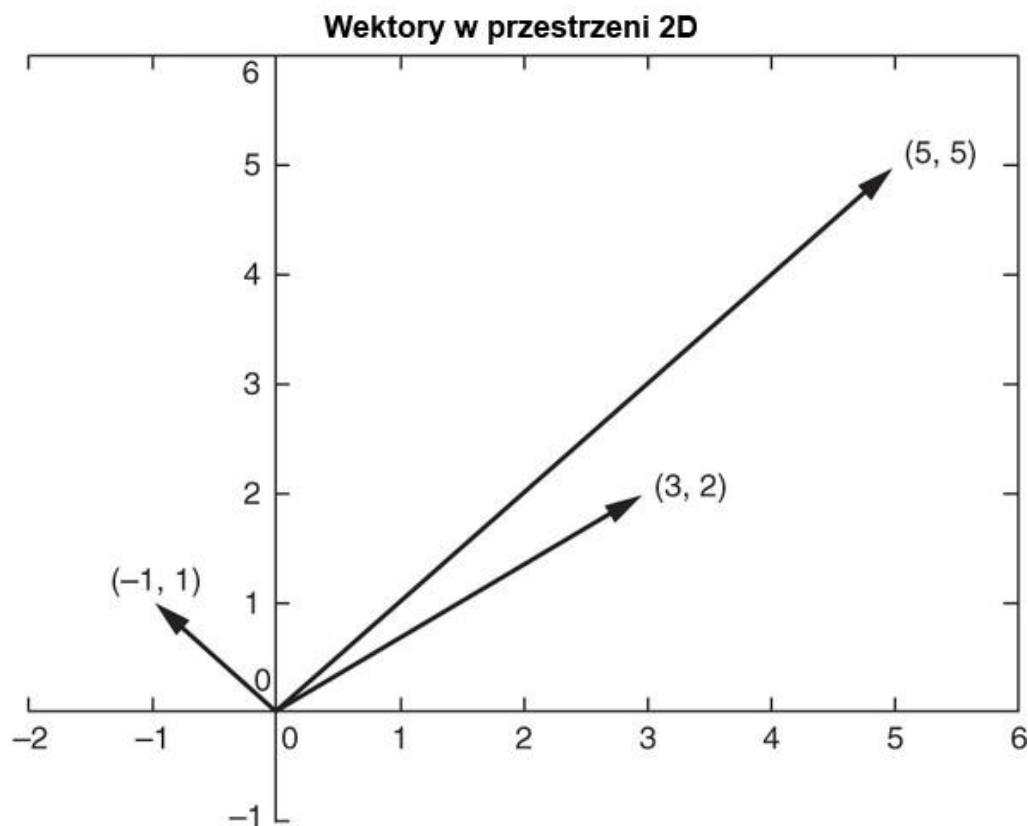


Rys. 6 Uproszczony Schemat działania wektorów TF-IDF

(źródło: opracowanie własne na podstawie [8])

Odwrotna częstotliwość występowania słów w dokumentach jest wykorzystywana do wyszukania odpowiednich dokumentów w korpusie poprzez pomiar ważności słów w stosunku do całego korpusu. IDF obliczane jest jako stosunek całkowitej liczby dokumentów do liczby dokumentów, w którym dane pojęcie występuje. Jeżeli pojęcie pojawia się często w jednym z dokumentów, a rzadziej w pozostałej części korpusu to zakłada się, że takie pojęcie jest ważne dla konkretnego dokumentu. Taką ważność można następnie wykorzystać do identyfikacji dokumentów, biorąc pod uwagę znaczenie określonych terminów w każdym z dokumentów [7].

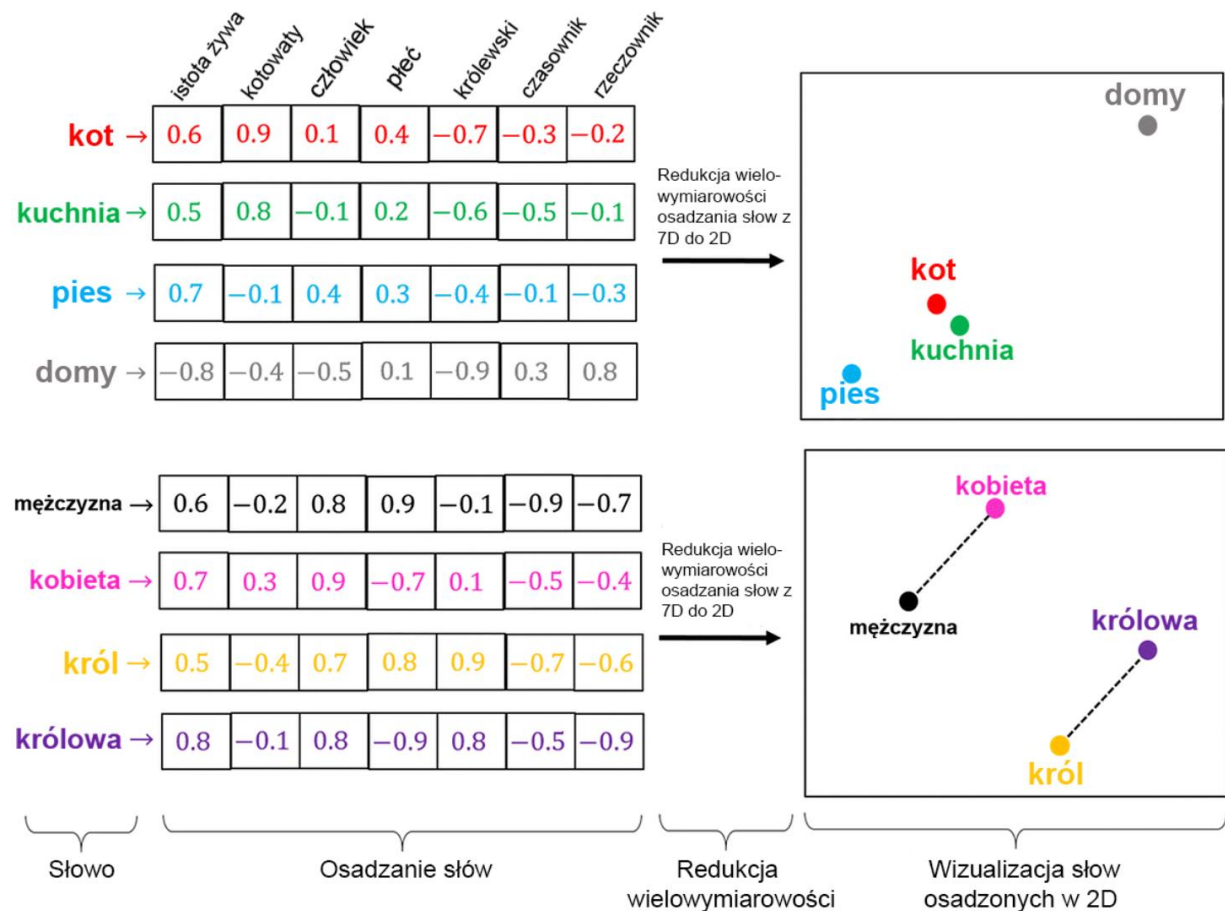
Wektory możemy przedstawić w przestrzeni wektorowej. Koncept ten polega na reprezentowaniu słów, dokumentów lub instrukcji w postaci wektorów w przestrzeni wielowymiarowej. Na przykład przestrzeni dwuwymiarowej przedstawionej na rysunku numer 7. Dzięki takiemu przedstawieniu możemy modyfikować wektorami przy użyciu algebry liniowej. Pozwala nam to na obliczanie statystyk związanych z tekstem w języku naturalnym. Jednym z celów przestrzeni wektorowej jest możliwość uchwycenia relacji semantycznych między słowami, które później można przetworzyć i użyć do tworzenia modeli uczenia maszynowego wykorzystywanych w czatbotach [7].



Rys. 7 Wektory w przestrzeni 2D

(źródło: [7])

Opisując wektory TF-IDF, nie sposób nie wspomnieć o prawie Zipf'a, które odgrywa kluczową rolę w przewidywaniu prawdopodobieństwa występowania słowa. Mówi nam, czy częstotliwość występowania słowa jest odwrotnie proporcjonalna do jego pozycji w tabeli częstości. Jest to równoznaczne z tym że jeżeli słowo częściej pojawia się w korpusie, to tym wyższa jest jego ranga i mniejsza częstotliwość. Prawo Zipf'a umożliwia oszacowanie statystycznego rozbiecia prawdopodobieństwa pojawienia się słowa w dowolnym dokumencie z korpusu. Poprzez zrozumienie takiego podziału możemy przewidzieć prawdopodobieństwo występowania słów w wybranym tekście języka naturalnego. Wizualizację wektorów na przykładzie przedstawia rysunek numer 8 [7].



Rys. 8 Wizualizacja wektorów na przykładzie słów

(źródło: opracowanie własne na podstawie [9])

Możliwości wykorzystania wektorowej reprezentacji słów [7] :

- *tworzenie wektorów słów*: wektory tworzy się poprzez uwzględnianie kontekstu słowa, który pasuje najbliżej z trafnością do kontekstu. Zwykle symboli jest mniej niż 10. Należy upewnić się, że znaczenia nie przeniosą się na sąsiednie zdania.
- *Identyfikacja relacji*: wektory słów mogą zostać użyte do identyfikacji antonimów, synonimów lub słów, które należą do tej samej kategorii, takich jak rzeczy, nazwy, miejsca, ludzie, pojęcia, zwierzęta. Taka identyfikacja pozwala na zrozumienie relacji pomiędzy słowami i ich znaczeniami.
- *Wykorzystanie wytrenowanych modeli*: wytrenowane modele mogą posłużyć w aplikacjach poprzez wykorzystanie wektorów słów do rozwiązywania i identyfikowania rzeczywistych zagadnień w przetwarzaniu języka naturalnego.

Jednym z elementów, które oblicza się przy użyciu wektorów, jest prawdopodobieństwo cosinusowe. Jest to miara, którą używa się do określenia podobieństwa pomiędzy wektorami. Obliczenie takiego podobieństwa polega na znalezieniu cosinusa kąta między wektorami. Skala podobieństwa waha się od 0 do 1. W tym wypadku 1 oznacza wektory identyczne, skierowane równolegle i wskazujące w tym samym kierunku, 0 jest oznaką braku podobieństwa. Do obliczenia podobieństwa cosinus, iloczyn skalarny dzieli się przez iloczyn ich wielkości. Ten iloczyn jest miarą tego, jak podobny jest kierunek wektorów, bez względu na ich wielkość. W przetwarzaniu języka naturalnego, podobieństwo cosinusowe jest używane do porównywania wektorów dokumentów, pokazując podobne wartości między dokumentami. Jest to użyteczne narzędzie przy analizowaniu języka naturalnego [10].

Metoda prawdopodobieństwa cosinusowego została ulepszona przez naukowców z londyńskiego City University. Oferuje ona nowe udoskonalenia w porównaniu do tradycyjnej metody. Tradycyjne obliczanie prawdopodobieństwa cosinusowe oblicza podobieństwo TF-IDF, a Okapi BM25 (taką nazwę nosi ta metoda), polega na wygładzeniu i normalizacji podobieństwa. Potrafi ignorować powtarzające się słowa w dokumencie, aby skutecznie wyznaczyć ich częstotliwość. Taka częstotliwość używana jest do określenia podobnych znaczeń. Pozwala na trafniejsze i doskonalsze przetwarzanie języka naturalnego [1, 7].

3.3. Wykorzystanie sztucznej inteligencji na jakość interakcji z czatbotem

Sztuczna inteligencja (SI) wywarła znaczący wpływ na jakość interakcji z czatbotami. Dzięki sztucznej inteligencji czatboty potrafią lepiej symulować ludzkie zachowania, ich odpowiedzi lepiej imitują komunikację, jaką stosuje człowiek. Odpowiedzi mogą mieć bardziej emocjonalny wydźwięk. Zastosowanie takich narzędzi pozwala czatbotom na szybszą naukę i możliwość szybszego przetwarzania dużej ilości informacji. Realne korzyści wynikające z zastosowania sztucznej inteligencji [10, 11]:

- dzięki narzędziom NLP czatboty są w stanie rozróżniać niejednoznaczne zapytania użytkowników, są w stanie reagować, nawet jeżeli pytanie w języku naturalnym nie jest dokładne gramatycznie. Pozwala również na lepsze zrozumienie wydźwięku rozmowy, przez co czatboty są w stanie wyłapać więcej informacji z kontekstu,
- dzięki algorytmom uczenia maszynowego czatboty udoskonalają swoje wypowiedzi. Badanie zachowań użytkowników dzięki dostępowi do danych pozwala na stworzenie

nowych modeli językowych, które zastosowane w czatbotach, będą potrafiły lepiej odpowiadać na zadane pytania,

- dzięki sztucznej inteligencji możemy przeanalizować duże zbiory danych i znajdowanie w nich kluczowych informacji. Dla przykładu zastosowanie wektorów TF-IDF i badanie ich odchyleń pozwala na szybsze wyłapywanie błędów i poprawę jakości rozmowy czatbota,
- sztuczna inteligencja jest używana również do rozpoznawania mowy, poprzez analizę danych dźwiękowych modele mogą rozpoznawać użytkowników w celu prowadzenia lepszej konwersacji z daną osobą.

3.4. Przykłady zastosowań czatbotów do celów komercyjnych

Czatboty otaczają nas na każdym kroku, czy przy przeglądaniu sklepów Internetowych, czy wyszukując informacje w popularnych wyszukiwarkach. Nowoczesne czatboty dzięki swoim możliwościom potrafią działać w kilku dziedzinach równocześnie [7]:

- boty odpowiadające na pytania: Asystent Google, Siri, Alexa,
- boty będące asystentami wirtualnymi: Asystent Google, Siri, Alexa,
- boty marketingowe - boty Twittera (X), boty Facebooka (Meta),
- boty do obsługi klienta - boty sklepowe, boty wsparcia technicznego,
- boty służące do zarządzania społecznością - Slackbot.

Ciekawym zagadnieniem marketingowym jest wykorzystanie czatbotów do promowania produktów (między innymi, gier wideo, filmów czy programów telewizyjnych). Firmy promujące swoje produkty za pomocą botów [7]:

- HBO promowało serial „Westworld” za pomocą „Aeden”,
- Disney promował film „Zwierzogród” botem „Oficer Judy Hopps”,
- Activision promowało grę „Call of Duty” za pomocą bota „Lt. Reyes”.

4. PROJEKT APLIKACJI CZATBOTA

4.1. Wymagania funkcjonalne aplikacji

Celem aplikacji jest stworzenie czatbota do rozmów o statystykach piłkarskich z wbudowaną analizą danych. Projekt czatbota od samego początku zakładał stworzenie czatbota wyspecjalizowanego w dziedzinie piłki nożnej. Wybranie jednej dziedziny pozwoliło mi lepiej skupić się na zadaniach, jakie niesie ze sobą stworzenie czatbota. Z racji ukierunkowanej wiedzy bota, mogłem skupić się na dokładniejszym zrozumieniu, jak przetwarza zapytania. Struktura aplikacji oparta jest na języku programowania Python z wykorzystaniem biblioteki Chatterbot oraz API Api-football.com do pobierania danych na temat piłki nożnej. Biblioteka Chatterbot opiera się na odpowiednio dobranych adapterach. Część z wykorzystanych w aplikacji adapterów współpracuje ze specjalnie przygotowanymi modułami. Moduły to przygotowane programy, których zadaniem jest pobranie z API niezbędnych danych do przetwarzania zapytań użytkownika. Jednym z głównych założeń była realizacja aplikacji, którą można w łatwy sposób rozbudować. Dlatego połączenie modyfikowalnych adapterów i modułów, które można dowolnie dodawać bądź usuwać było najlepszym rozwiązaniem. Chatbot działa w języku angielskim, ponieważ biblioteka Chatterbot oraz wybrane API najlepiej współpracują w tym języku. Dzięki mnogości danych do trenowania bota właśnie w języku angielskim wybór ten był najbardziej uzasadniony.

4.2. Opis funkcjonalności

Projektowany chatbot to przede wszystkim rozmówca wirtualny, który zna odpowiedzi na pytania. Chatbot zostanie wyposażony w szereg funkcji tak, aby użytkownik otrzymywał konkretne odpowiedzi na temat piłki nożnej i powiązanych z nią zagadnień. Dzięki zastosowaniu technik przetwarzania naturalnego, możliwe będzie dokładniejsze dopasowanie zwracanych odpowiedzi.

Funkcje czatbota:

- pytania o wyniki meczów,
- pytania o dane zawodników (aktualny/poprzedni klub, statystyki),
- pytania o statystyki drużyn (pozycja w tabeli, bilans goli, ostatnie mecze),
- pytania o zwycięzców lig, pucharów.

Funkcje czatbota z wykorzystaniem analizy danych:

- porównanie zwycięzców lig, pucharów,
- porównanie statystyk zawodników (liczba goli, meczów),
- analiza formy drużyn pobranie statystyk dotyczących formy drużyn i ich wyników w ostatnich meczach.

4.3. Struktura aplikacji oraz użyte technologie

Podstawą do napisania aplikacji był wybór języka programowania. Odpowiedni dobór narzędzi pozwala na lepsze wykonanie zadania dlatego, do projektu czatbota został wykorzystany język programowania Python. **Python**, jest językiem programowania wysokiego poziomu, przeznaczonym do różnych zagadnień. Zaletą Pythona jest przejrzystość i czytelność kodu. Python posiada bogaty zasób bibliotek i narzędzi do przetwarzania języka naturalnego. Biblioteki pythona NLTK oraz spaCy idealnie nadają się do zbudowania aplikacji czatbota. Python dobrze radzi sobie z ciągami znaków oraz przetwarzaniem tekstu co ułatwia proces trenowania czatbotów. Dzięki elastyczności Pythona wiele funkcji można wykorzystać w kilku miejscach tak, aby aplikacja stała się jeszcze lepsza. Jego przejrzystość kodu oraz łatwość rozbudowy pozwala na stosunkowo proste stworzenie czatbota. Oczywiście będzie to bardzo podstawowa wersja, jednak jest ona podstawą do rozbudowy i poszerzania aplikacji o nowe funkcje. Python używany jest w wielu miejscach na świecie, między innymi w badaniach naukowych czy edukacji. Dzięki szerokiemu zastosowaniu Pythona, czatboty w nim stworzone mogą działać w różnych dziedzinach życia ludzi [12, 13].

NLTK, jest biblioteką używaną do przetwarzania języka naturalnego. Dzięki dużemu zakresowi funkcji umożliwia bardzo dobrą analizę tekstu. Pozwala między innymi na tokenizację i klasyfikację tekstu, analizę składniową czy wydobywanie kluczowych informacji [14].

spaCy - rozbudowana biblioteka przeznaczona do przetwarzania języka naturalnego. Wykorzystujące między innymi tokenizację, wektory słów czy rozpoznawanie jednostek nazwanych (modeli NER) do analizy tekstu. Jej zaletą jest wysoka jakość wyników i duża wydajność [15].

4.4. Biblioteka ChatterBot



Rys. 9 Logo Chatterbot

(źródło: [16])

4.4.1. Wprowadzenie do biblioteki ChatterBot

ChatterBot jest biblioteką w Pythonie, która została stworzona do łatwiejszego budowania aplikacji czatbotów. Jest wyposażona w różne algorytmy uczenia maszynowego, co pozwala jej na generowanie odpowiedzi dla użytkownika. Jej zaletą jest przede wszystkim wygoda, dzięki czemu jesteśmy w stanie zbudować czatboty bardziej responsywne oraz bardziej zautomatyzowane. Biblioteka Chatterbot jest funkcjonalna na wiele sposobów, a jednym z nich jest bardzo dobra analiza wypowiedzi i uczenie się na podstawie danych [16].

4.4.2. Architektura ChatterBot

Adaptery logiki (Logic Adapters)

Adaptery logiki to komponenty biblioteki chatterbot, odpowiadające za analizę danych wejściowych podanych przez użytkownika i generowanie odpowiedzi zwrotnej. Chatterbot umożliwia użycie kilku adapterów logiki równocześnie. Adaptery można dostosować do różnych celów na przykład [16]:

- **best match adapter:** wybiera najbardziej pasującą odpowiedź na podstawie podobieństwa,
- **time logic adapter:** służy do generowania odpowiedzi, które mają związek z czasem.
- **mathematical evaluation adapter:** sprawdza czy podane zapytanie zawiera wyrażenia matematyczne, które można wykonać. Jeżeli znajdzie istniejące wyrażenie zwróci odpowiedź z wynikiem,
- **specific response adapter:** zwraca konkretne odpowiedzi na określone pytania.

4.4.3. Baza danych czatbota

Chatterbot przechowuje dane konwersacji w bazie danych. W tym celu może wykorzystać różne systemy baz danych między innymi SQLite czy MongoDB. Baza danych przechowuje dane wejściowe oraz odpowiedzi wygenerowane przez program, dzięki temu pozwala na ponowne uczenie się i lepsze dopasowanie przyszłych odpowiedzi bota [16].

4.4.4. Trening czatbota

Chatterbot zawiera narzędzia, za pomocą których można trenować czatbota. W tym celu można użyć plików wejściowych takich jak: pliki tekstowe, odpowiedzi z poprzednich czatów, czy dedykowane korpusy. Biblioteka Chatterbot zawiera wbudowane moduły, pozwalające na łatwe trenowanie czatbota na podstawie powszechnych zestawów danych (chatterbot.corpus) [16].

4.4.5. Zasada działania biblioteki Chatterbot

Chatterbot uczy się na podstawie analizy rozmów zapisanych w bazie danych. Bot przetwarza dane i dostosowuje się tak, aby następnym razem lepiej odpowiadać na różne rodzaje pytań. Uczenie się czatbota opiera się na algorytmach przetwarzania języka naturalnego [16].

Chatterbot, przetwarza zapytanie otrzymane przez użytkownika za pomocą adapterów logiki. Każdy z użytych adapterów analizuje dane wejściowe podane przez użytkownika i generuje odpowiedź. Kończącym etapem jest wybranie najlepiej pasującej odpowiedzi, które wygenerowały wcześniej adaptery logiki [16].

Przetworzone zapytanie wraz z wybraną najlepszą odpowiedzią, przekazywane jest do użytkownika. Odpowiedź zwrotna zostaje zapisana w bazie danych i może posłużyć jako przyszłe dane treningowe czatbota [16].

4.5. Analiza danych piłkarskich

4.5.1. Wykorzystane API i sposób pobierania danych

API - jest interfejsem programistycznym aplikacji, opisujący zestaw reguł, w jakim powinny komunikować się ze sobą programy. API można interpretować na kilka sposobów, w przypadku użycia w projekcie aplikacji czatbota do rozmów o statystykach piłkarskich, jest on

zbiorem danych, dzięki któremu aplikacja pozyskuje informacje między innymi o zawodnikach czy drużynach piłkarskich. API pracuje na konkretnych punktach końcowych. **Punkty końcowe** to określone adresy, dzięki którym aplikacje mogą się połączyć. Każdy punkt końcowy przypisany jest do określonego zasobu aplikacji [17].

Api-football.com, jest API komercyjnym dostarczającym informacje na temat piłki nożnej. API pozwala na zwrócenie za pomocą dedykowanych punktów końcowych, wielu przydatnych informacji o piłce nożnej. Dzięki rozbudowanemu API czatbot ma duży zasób danych, które może wykorzystać w przygotowaniu odpowiedzi. Kolejnym czynnikiem odpowiadającym się za wyborem właśnie tego API, jest to, że firma udostępnia w ramach dostępnych pakietów, inne dyscypliny sportowe. Dzięki temu czatbot może zostać w łatwy sposób przekształcony na funkcjonowanie w innej dziedzinie sportu [18].

Przykładowe punkty końcowe używane przez Api-football.com [18]:

- /leagues: zwraca listę dostępnych lig,
- /teams: zwraca informacje o drużynach,
- /standings: zwraca aktualne tabele ligowe,
- /fixtures: zwraca dane na temat meczów,
- /players: zwraca informacje o zawodnikach i ich statystykach,
- /transfers: zwraca informacje na temat transferów zawodników.

4.5.2. Przygotowanie danych do analizy

Zastosowanie odpowiednich punktów końcowych Api-football, pozwala na stworzenie dedykowanych modułów. Moduły mają za zadanie pobranie odpowiednich danych z API, i sformatowanie ich w taki sposób, aby instancja czatbota mogła je wykorzystać.

Pierwszym krokiem w budowie modułu jest połączenie z API przy pomocy specjalnie generowanego klucza API. **Klucz API** to unikalny ciąg znaków używany do autoryzacji dostępu do aplikacji API. Pozwala na identyfikację użytkownika w celu pobierania opłat za korzystanie z aplikacji API [17].

Następnie należy zdefiniować punkt końcowy, z którego chcemy skorzystać. Oraz dane wejściowe dla określonego punktu końcowego. W dokumentacji Api-football, punkty końcowe są dokładnie opisane wraz z danymi, które udostępniają. Poniżej przedstawiony przykład gotowego modułu odpowiedzialnego za pobieranie danych z API na temat zajmowanych miejsc w tabeli ligowej drużyn z odpowiedniego kraju. W przypadku programu z listingu numer 1, wyświetlone zostaną informacje na temat ligi włoskiej.

```
class StandingsAnalyzer:
    def __init__(self, api_key):
        self.api_key = api_key
        self.base_url = "https://v3.football.api-sports.io"

    def get_standings(self, league_id, season=None):
        endpoint = "/standings"
        params = {"league": league_id, "season": season}
        headers = {"x-rapidapi-key": self.api_key}

        url = self.base_url + endpoint
        response = requests.get(url, headers=headers, params=params)
        data = response.json()

        if 'response' in data and len(data['response']) > 0:
            standings = data['response'][0]['league']['standings'][0]
            table = []

            for team in standings:
                rank = team['rank']
                name = team['team']['name']
                points = team['points']
                goals_diff = team['goalsDiff']
                table.append([rank, name, points, goals_diff])
            else:
                print("Brak danych dla danej ligi i sezonu.")

api_key = "xxx" # ukryty klucz api
league_id = "135" # ID ligi włoskiej
season = '2023' # rok w którym aplikacja ma sprawdzać dane
analyzer = StandingsAnalyzer(api_key)
analyzer.get_standings(league_id, season)
```

Listing 1 Kod programu odpowiedzialny za analizę tabeli ligowej ligi włoskiej

Tak pobrane dane można dowolnie sformatować i przekazać do instancji czatbota. Ten program wykorzystuje jedynie cztery parametry zwrotne, jednak ich ilość można dowolnie dostosować, ograniczeniem jest jedynie API i to jakie dane udostępnia. Tak przygotowane dane można w łatwy sposób wyświetlić. Ma to na celu przetestowanie działania modułu, zanim dane zostaną przekazane do właściwej instancji czatbota, jak przedstawiono na rysunku numer 10.

Rank	Team	Points	Goals Difference
1	Inter	94	67
2	AC Milan	75	27
3	Juventus	71	23
4	Atalanta	69	30
5	Bologna	68	22
6	AS Roma	63	19
7	Lazio	61	10

Rys. 10 Działanie programu przedstawiającego informacje o tabeli ligowej

(źródło: opracowanie własne)

4.6. Przechowywanie danych

Ważnym elementem czatbota jest możliwość przechowywania rozmów oraz wykorzystanie pobranych danych z API do przetwarzania zapytań. Dlatego w tym celu wybrałem popularną bibliotekę Pythona SQLite. SQLite, to biblioteka napisana w języku C, jej zaletami jest wysoka szybkość działania oraz niezawodność. Dlatego idealnie nadaje się do projektu czatbota napisanego w Python [3, 19].

5. IMPLEMENTACJA CZATBOTA

5.1. Tworzenie czatbota

Projekt aplikacji zakładał stworzenie programu, który można w łatwy sposób modyfikować. Dlatego czatbot jest oparty na specjalnie przygotowanych adapterach Chatterbota, które wspierane są modułami pobierającymi dane z API. Takie podejście pozwala na skalowalność projektu i jego dalszą rozbudowę o nowe funkcje. Podstawowym elementem biblioteki Chatterbot jest tworzenie instancji czatbota przy pomocy dedykowanych narzędzi.

```
chatbot = ChatBot(  
    'Football-Bot',  
    storage_adapter='chatterbot.storage.SQLiteStorageAdapter',  
    logic_adapters=[  
        'chatterbot.logic.BestMatch',  
        {  
            'import_path': 'adapters.playerInfoAdapter.PlayerInfoAdapter',  
            'api_key': 'xxx',  
        }  
    ],  
    database_uri='sqlite:///database.sqlite3',  
    trainer='chatterbot.trainers.ListTrainer'  
)
```

Listing 2 Kod programu odpowiadający za stworzenie instancji czatbota

Po dodaniu odpowiedniej konfiguracji i zastosowaniu nieskończonej pętli, czatbot jest gotowy do uruchomienia. Dzięki możliwościom Chatterbota nie ma potrzeby dodawania wszystkich adapterów przy tworzeniu instancji czatbota. Biblioteka ta pozwala na późniejsze rozszerzenie bota o dodatkowe adaptory dzięki zastosowaniu odpowiedniego polecenia, które przedstawiono w listingu numer 3.

```
chatbot.logic_adapters.extend("tutaj należy wpisać odpowiednio nowe adaptory")  
#nieskończona pętla programu  
while True:  
    query = input("You: ")  
    if query == 'exit':  
        break  
    answer = chatbot.get_response(query)  
    print("bot: ", answer)
```

Listing 3 Kod odpowiedzialny za rozszerzanie czatbota o nowe adaptory, oraz za nieskończoną pętlę czatbota.

```
You: Hi
bot: Hello
You: how are you
bot: I'm doing great.
```

Rys. 11 Podstawowe działanie czatbota

(źródło: opracowanie własne)

Implementowany czatbot działa w terminalu Pythona. Zastosowane formatowanie widoczne jest na rysunku 11, to dostosowany terminal, w którym prowadzone są rozmowy z czatbotem. Do wyświetlania analizy danych piłkarskich wykorzystana została biblioteka **tabulate**. **Tabulate**, jest biblioteką Pythona wykorzystywaną do formatowania danych w tablice tekstowe (tabele) [20].

5.2. Moduły czatbota

Moduły są bardzo ważnym elementem aplikacji. Na jednych opierają się adaptory Chatterbota, inne służą do celów użytkowych dla aplikacji. Wykorzystywane są w różnych miejscach tak, aby mogły jak najlepiej wspomagać czatbota. Dzielą się na:

- moduły pozyskujące informacje do bazy danych,
- moduły użytkowe,
- moduły służące do zwracania danych dla adaptera, który następnie używany jest przez główną instancję czatbota.

5.2.1. Moduły pozyskujące informacje do bazy danych

Najlepszym rozwiązaniem dla czatbota jest wcześniejsze pobranie części ważnych i powtarzających się informacji do bazy danych. Chatbot, zamiast pobierać za każdym razem dane, może najpierw sięgnąć do lokalnej bazy, aby pobrać odpowiednie informacje. Takie podejście pozwala odciążyć API, ponieważ w zależności od wybranego planu, darmowe zapytania do API mogą się wyczerpać. Najważniejszym typem danych, które znajdują się w lokalnej bazie są numery

identyfikacyjne zawodników i drużyn piłkarskich. **Numery identyfikacyjne (ID)** w API to niezmienny i unikatowy zasób pozwalający na identyfikację obiektu. Raz pobrane nie muszą być aktualizowane. Za pomocą jednego programu i zaledwie jednego zapytania do API, moduł jest w stanie pobrać i zapisać wszystkie ID drużyn z danej ligi. Kolejnym bardzo ważnym elementem tego typu modułów jest pobieranie danych z API tak, aby można było je wykorzystać do trenowania modeli czatbota. Takie rozwiązanie pozwala na trenowanie bota danymi, na których będzie w przyszłości bezpośrednio pracował [17, 18]

```
def get_team_id_by_league_season(endpoint, params = {}):
    parameters = ""
    if params:
        parameters = '?' + '&'.join([f'{k}={v}' for k, v in params.items()])

    headers = {
        'x-rapidapi-key': 'xxx'
    }

    url = 'https://v3.football.api-sports.io/' + endpoint + parameters
    response = requests.get(url, headers=headers)
    response = response.json()
    return response

teams = get_team_id_by_league_season('teams', {'league': 135, 'season': 2022})
for team in teams['response']:
    team_id = team['team']['id']
    team_name = team['team']['name']
```

Listing 4 Kod programu do pobierania identyfikatorów drużyn

Kod programu pokazany w listingu 4, na podstawie ID ligi, pobiera identyfikatory oraz nazwy wszystkich drużyn w danej lidze. Takie dane następnie zapisane zostaną w lokalnej bazie danych i wykorzystane są w celach treningowych oraz do przetwarzania zapytań użytkownika. Analogiczne podejście zastosowane zostało przy pobieraniu nazwisk piłkarzy. Należy zmienić ustawienia programu na przeszukiwanie API po odpowiednim punkcie końcowym. Ten program przesyła do API jedynie jedno zapytanie, pozyskując przy tym znaczną ilość danych. Jeżeli będzie wymagała tego rozbudowa aplikacji, moduł można dowolnie przekształcić lub na jego podstawie stworzyć nowy tak, aby wczytać do bazy inne potrzebne dane.

5.2.2. Moduły użytkowe

Api-football swoje działanie dostarczania konkretnych danych, opiera na unikalnym identyfikatorze obiektu. Aby mogło zwrócić odpowiednie dane, potrzebuje odpowiedniego ID dla wskazanego obiektu. Jednak podczas rozmowy z czatbotem, trudno wymagać od użytkownika, aby zadawał pytanie o zawodnika i używał zamiast nazwiska ID. Rozważając taki przypadek, konieczne było stworzenie modułu, który na podstawie nazwiska, będzie potrafił odnaleźć odpowiednie ID piłkarza. Analogiczna sytuacja dotyczy nazw lig oraz drużyn. Takie moduły dostarczają odpowiednie ID do modułów służących do zwracania danych dla adaptera [18].

```
def find_player(player_name):
    connection = sqlite3.connect('football_base.db')
    cursor = connection.cursor()
    query = "SELECT id_player, full_name FROM players_id"
    cursor.execute(query)
    rows = cursor.fetchall()
    player_info = None
    match_ratio = 0

    for row in rows:
        current_player_id, current_team_name = row
        current_match_ratio = difflib.SequenceMatcher(None, player_name.lower(),
        current_team_name.lower()).ratio()

        if current_match_ratio > match_ratio:
            match_ratio = current_match_ratio
            player_info = row

    cursor.close()
    connection.close()

    if match_ratio > 0.7: # Minimalny próg dopasowania
        print(player_info)
        return player_info
    else:
        print("Football player not found looking for similar...")
        connection = sqlite3.connect('football_base.db')
        cursor = connection.cursor()
        query = "SELECT full_name FROM players_id"
        cursor.execute(query)
        rows = cursor.fetchall()
        similar_teams = []
        for row in rows:
```

```

        current_team_name = row[0]
        current_match_ratio = difflib.SequenceMatcher(None, player_name.lower(),
current_team_name.lower()).ratio()
        if current_match_ratio > 0.4: # Minimalny próg podobieństwa
            similar_teams.append(current_team_name)
        cursor.close()
        connection.close()

    if similar_teams:
        print("No exact match found. Similar players:")
        for team in similar_teams:
            print(team)
    else:
        print("No similar players found.")

player_name = input("Enter the name of the player: ")
team_info = find_player(player_name)

```

Listing 5 Kod programu do odnajdywania identyfikatora zawodnika

Kod programu pokazany w listingu 5, wykorzystuje bibliotekę difflib (wykorzystywana do porównywania znaków czy sentencji). Difflib pozwala ustawić minimalny próg dopasowania. Następnie po przekazaniu do niej zapytania użytkownika porównuje dane z lokalnej bazy danych. Kończy pracę, jeżeli znajdzie odpowiednie ID. Jeżeli zapytanie przekracza minimalny próg dopasowania, biblioteka sprawdza ponownie, jednak tym razem używa mniejszego progu dopasowania. Użytkownicy popełniają błędy przy podawaniu zapytań. Może to doprowadzić do nieodnalezienia zapytania. Kolejnym przypadkiem, który wymaga użycia mniejszego progu dopasowania, jest to, że zawodnicy mogą posiadać takie same nazwiska. W obu przypadkach program dopytuje użytkownika, o którego konkretnie zawodnika mu chodziło. Na rysunkach 12 i 13 przedstawiono odrębne działanie modułu w celach testowych poza czatbotem.

```

Enter the name of the player: Bero
(37122, 'M. Bero')

```

Rys. 12 Poprawne zwrócenie przez moduł ID piłkarza

(źródło: opracowanie własne)

```
Enter the name of the player: delpi
Football player not found looking for similar...
No exact match found. Similar players:
F. Delph
L. Delap
```

Rys. 13 Zwrócenie przez moduł piłkarzy do wyboru

(źródło: opracowanie własne)

Jest to przykład modułu pracującego z nazwiskami piłkarzy. Analogiczne podejście zostało zastosowane przy nazwach drużyn, nazwach lig oraz innych potrzebnych danych.

5.2.3. Moduły służące do zwracania danych dla adaptera

Istotnym elementem adapterów czatbota są moduły przekazujące do nich dane z API. Dzięki takiemu rozwiązaniu adapter po zinterpretowaniu intencji użytkownika jest w stanie wykorzystać odpowiedni moduł i otrzymać dane z API, w celu późniejszego zwrócenia informacji zwrotnej dla użytkownika. Moduły te zostały odpowiednio dostosowane w celu przeszukania API, i wydobycia z niej tylko tych informacji, które są aktualnie potrzebne.

```
class PlayerAnalyzer:
    def __init__(self, api_key):
        self.api_key = api_key
        self.base_url = "https://v3.football.api-sports.io"

    def get_player_details(self, player_id, season=None):
        endpoint = f"/players?id={player_id}"
        if season is None:
            current_year = datetime.datetime.now().year
            season = current_year - 1
            season = season

        params = {'season': season}
        headers = {"x-rapidapi-key": self.api_key}
        url = self.base_url + endpoint
        response = requests.get(url, headers=headers, params=params)
        data = response.json()
        return data
```

```

def analyze_player(self, player_id, season):
    player_data = self.get_player_details(player_id, season)
    if 'response' in player_data and len(player_data['response']) > 0:
        player_name = player_data['response'][0]['player']['name']
        player_firstname = player_data['response'][0]['player']['firstname']
        player_lastname = player_data['response'][0]['player']['lastname']
        player_age = player_data['response'][0]['player']['age']
        player_nationality = player_data['response'][0]['player']['nationality']
        player_position = player_data['response'][0]['statistics'][0]['games']['position']
        player_goals = player_data['response'][0]['statistics'][0]['goals']['total']
        player_assists = player_data['response'][0]['statistics'][0]['goals']['assists']
        player_team = player_data['response'][0]['statistics'][0]['team']['name']
        return player_name, player_firstname, player_lastname, player_age, player_nationality,
        player_position, player_goals, player_assists, player_team

def player_details_by_name(player_name, season=None):
    player_info = find_player(player_name)
    if type(player_info) == list:
        return player_info
    elif player_info:
        player_id = find_player(player_name)[0]
        analyzer = PlayerAnalyzer(api_key)
        return analyzer.analyze_player(player_id, season)

```

Listing 6 Kod programu do pozyskiwania danych na temat zawodnika

Moduł przedstawiony w listingu numer 6 współpracuje z modulem użytkowym do odnajdywania ID piłkarza po nazwisku. Następnie wysyła odpowiednie zapytanie do API w celu pozyskania danych. Istotnym elementem tego programu jest podanie odpowiedniego roku, dla którego statystyki mają zostać wyświetlone. Domyślnie jeżeli użytkownik nie poda roku zostanie użyty aktualny rok kalendarzowy. Jeden sezon piłkarski rozciąga się pomiędzy dwa lata kalendarzowe, dlatego w celu ujednolicenia Api-football, stosuje rok wcześniejszy jako aktualny. Testowe działanie modułu przedstawione jest na rysunku numer 14.

```
Name: David de Gea
Age: 34
Nationality: Spain
Position: Goalkeeper
Total Goals: 0
Total Assists: None
player_team: Fiorentina
```

Rys. 14 Zwrócone dane przez moduł dla określonego zawodnika

(źródło: opracowanie własne)

Innym przykładem modułu służącego do zwracania danych dla adaptera, jest moduł odpowiadający za analizę danych. Jego kod przedstawiony jest w listingu numer 7. W tym przypadku moduł analizuje ostatnie rozegrane spotkania pomiędzy dwoma drużynami i wyświetla informacje na ich temat. Przykład takiej analizy przedstawiony jest na rysunku numer 15.

```
class TeamComparisonAnalyzer:
    def __init__(self, api_key):
        self.api_key = api_key
        self.base_url = "https://v3.football.api-sports.io"

    def compare_teams(self, team1_id, team2_id):
        endpoint = "/fixtures/headtohead"
        params = {
            "h2h": f"{team1_id}-{team2_id}"
        }
        headers = {
            "x-rapidapi-key": self.api_key
        }
        url = self.base_url + endpoint
        response = requests.get(url, headers=headers, params=params)
        data = response.json()
        if 'response' in data and len(data['response']) > 0:
            fixtures = data['response']
            table_data = []
            headers = ["Date", "Home Team", "Away Team", "Result", "Score", "Winner"]
            for fixture in fixtures:
```

```

    fixture_date = fixture['fixture']['date']
    fixture_home_team = fixture['teams']['home']['name']
    fixture_away_team = fixture['teams']['away']['name']
    fixture_result = fixture['fixture']['status']['short']
    fixture_home_score = fixture['goals']['home']
    fixture_away_score = fixture['goals']['away']

    if fixture_home_score is None or fixture_away_score is None:
        fixture_score = "N/A"
        fixture_winner = "N/A"
    else:
        fixture_score = f"{fixture_home_score}-{fixture_away_score}"
        if fixture_home_score > fixture_away_score:
            fixture_winner = fixture_home_team
        elif fixture_home_score < fixture_away_score:
            fixture_winner = fixture_away_team
        else:
            fixture_winner = "Draw"

    table_data.append([fixture_date, fixture_home_team, fixture_away_team,
fixture_result, fixture_score, fixture_winner])
    return tabulate(table_data, headers, tablefmt="grid")
else:
    return print("No data found for the specified teams or no comparison.")

```

Listing 7 Kod programu zawierającego moduł do analizy danych dwóch drużyn

2022-09-03T16:00:00+00:00	AC Milan	Inter	FT	3-2	AC Milan
2023-01-18T19:00:00+00:00	AC Milan	Inter	FT	0-3	Inter
2023-05-10T19:00:00+00:00	AC Milan	Inter	FT	0-2	Inter
2024-04-22T18:45:00+00:00	AC Milan	Inter	FT	1-2	Inter

Rys. 15 Porównanie dwóch drużyn i wyników ostatnich meczów

(źródło: opracowanie własne)

5.3. Adaptery czatbota

Aby czatbot stworzony z użyciem biblioteki Chatterbot działał poprawnie, trzeba dobrać do niego właściwe adaptery. Na adapterach opiera się podstawowa mechanika działania czatbota. To one odpowiadają za odpowiednie zrozumienie sensu wypowiedzi użytkownika i wyłapanie odpowiednich informacji. Dzięki adapterom czatbot jest w stanie przetworzyć zapytanie i zwrócić poprawną odpowiedź. Niestandardowe adaptery użyte w aplikacji bazują na podstawowej klasie Chatterbota 'Logic Adapter'. Takie adaptery wykorzystują bibliotekę SpaCy do przetworzenia tekstu zapytania. Następnie stosują specjalnie wytrenowany model NER, do wykrywania specyficznych nazw. Po przeszukaniu zapytania dopasowuje znalezione dane z odpowiednimi encjami i przekazuje te dane do odpowiedniego modułu tak, aby zapytanie mogło zostać przetworzone, a wynik mógł zostać zwrócony do użytkownika. **NER (Named-entity recognition)** - jest techniką przetwarzania języka naturalnego. Służy do klasyfikacji i identyfikacji nazwanych jednostek (encji) w teście [13, 10, 16]. W aplikacji czatbota użyte są między innymi wbudowane encje 'PERSON', 'DATE' oraz dodatkowe encje stworzone specjalnie na potrzeby czatbota, 'SPORT_TEAM', 'LEAGUE', 'TROPHY'.

Poniżej omówię jeden z adapterów czatbota. Odpowiada on za wyświetlanie statystyk danego piłkarza. Adapter jest rozbudowany dlatego kod zostanie rozbity na kilka osobnych elementów. Działanie każdego z kawałka kodu zostanie opisane w bardziej przejrzysty sposób.

```
class PlayerStatsAdapter(LogicAdapter):
    def __init__(self, chatbot, stat_keywords, stat_label, stat_index, **kwargs):
        super().__init__(chatbot, **kwargs)
        self.nlp = spacy.load('en_core_web_sm')
        self.custom_ner_model = spacy.load("football_ner_model3")
        self.stat_keywords = stat_keywords
        self.stat_label = stat_label
        self.stat_index = stat_index
```

Listing 8 Kod programu do inicjalizacji instancji adaptera

Fragment kodu w listingu numer 8 odpowiada za inicjalizację instancji adaptera. Ładuje dwa modele językowe SpaCy: en_core_web_sm (standardowy model do analizy tekstu) oraz football_ner_model3 (wytrenowany niestandardowy model służący do wykrywania nazwisk piłkarzy i nazw drużyn).

Kolejnym krokiem inicjalizacji adaptera jest załadowanie kluczowych słów (keywords), których adapter szuka w zapytaniu oraz załadowanie etykiet (label), które posłużą do generowania odpowiedzi. Takie ładowanie przedstawione jest w listingu numer 9.

```
def extract_year_from_date(self, date_str):
    # Użyj modelu spaCy do analizy tekstu i wydobycia informacji o dacie
    doc = self.nlp(date_str)
    tokens = [token.text.lower() for token in doc]

    year = None
    last_index = tokens.index("last") if "last" in tokens else -1
    if last_index != -1:
        if last_index + 1 < len(tokens) and tokens[last_index + 1] == "year":
            year = datetime.now().year - 1
    if year is None:
        for token in tokens:
            try:
                year = w2n.word_to_num(token)
                break
            except ValueError:
                continue
    if year is None:
        return None
    if "ago" in tokens:
        now = datetime.now()
        return now.year - year
    else:
        return year
```

Listing 9 Kod programu do ładowania etykiet dla odpowiedzi

Ten kod odpowiada za przetworzenie daty wyrażonej w formie słownej na zapis liczbowy. Dzięki temu czat jest w stanie przetworzyć różnego rodzaju zapytania zawierające datę. Następnie przesłać odpowiednie dane do API. Na przykład „three years ago” (trzy lata temu), metoda `extract_year_from_date` przetworzy na rok 2021. Ponieważ użyte API pracuje na dacie w postaci roku, rola tej metody jest istotna w dobrym generowaniu odpowiedzi przez czatbota.

```

def process(self, input_statement, additional_response_selection_parameters):
    player_name = None
    response_text = ""
    target_date = None
    doc = self.nlp(input_statement.text)

    for ent in self.custom_ner_model(doc).ents:
        if ent.label_ == "PERSON":
            player_name = ent.text
            break

    for ent in doc.ents:
        if ent.label_ == "DATE":
            target_date = self.extract_year_from_date(ent.text)
            break

    if player_name:
        player_info = player_details_by_name(player_name, target_date)
        if player_info:
            confidence = 1
            if type(player_info) == list:
                response_text += "soccer player not found here are similar:\n"
                for players_new in player_info:
                    response_text += f"{players_new}\n"
            elif player_info and len(player_info) >= self.stat_index + 1:
                stat_value = player_info[self.stat_index]
                if target_date is None:
                    date_print = datetime.now().year
                else:
                    date_print = target_date
                if self.stat_label == "Goals":
                    response_text = f"{player_name} has {stat_value} {self.stat_label} in
{date_print}."
                elif self.stat_label == "Assists":
                    response_text = f"{player_name} has {stat_value} {self.stat_label} in
{date_print}."
                elif self.stat_label == "Nationality":
                    response_text = f"{player_name} is from {stat_value}."
                else:
                    response_text = f"{player_name} has {stat_value} {self.stat_label} in
{date_print}."
            else:
                response_text = f"Sorry, I couldn't find {self.stat_label.lower()} information for
{player_name}."
        else:
            response_text = "I'm sorry, I don't understand your query."

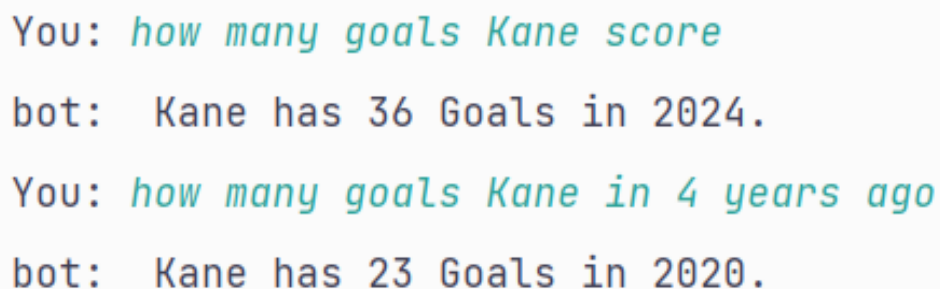
```

```
confidence = 1 if player_name else 0
response_statement = self.chatbot.storage.create(text=response_text)
response_statement.confidence = confidence
return response_statement
```

Listing 10 Kod programu do przetworzenia zapytań i generowania odpowiedzi.

Najważniejszym elementem adaptera jest przetworzenie zapytania i zwrócenie odpowiedzi. Kod programu z listingu 10, odnajduje odpowiednie encje w zapytaniu ('PERSON', 'DATE') i na ich podstawie generuje odpowiedź zwrotną. Adapter korzysta z funkcji jednego z modułów `player_details_by_name`, która zwraca informacje o piłkarzu dla konkretnego roku. Działanie adaptera w skrótych krokach można opisać następująco:

- Adapter rozpoznaje słowo kluczowe np. goals,
- Biblioteka SpaCy na podstawie wytrenowanych modeli wykrywa encje piłkarza jako nazwisko (np. Harry Kane), oraz encję daty (np. four years ago). Jeżeli data nie występuje w zapytaniu, adapter używa roku bieżącego)
- Wywołuje moduł `player_details_by_name` w celu uzyskania informacji o ilości bramek zdobytych przez piłkarza (Harry Kane bramki cztery lata temu),
- Generuje odpowiedź 'Kane has 23 Goals in 2020.' Co zaprezentowano na rysunku numer 16.



```
You: how many goals Kane score
bot: Kane has 36 Goals in 2024.
You: how many goals Kane in 4 years ago
bot: Kane has 23 Goals in 2020.
```

Rys. 16 Przykładowe działania adaptera

(źródło: opracowanie własne)

Aplikacja czatbota posiada wiele adapterów tak, aby mogła dopasować odpowiedzi na różne pytania dotyczące piłki nożnej. Ich zasada działania jest podobna do adaptera wyświetlającego statystyki piłkarza. Dokładne działanie adapterów należało dostosować do

odpowiedniego przypadku. Musiały zostać dopasowane pod odpowiednie encje oraz przetwarzanie odpowiednich informacji. Są to między innymi:

- adaptery odpowiedzialne za statystyki i informacje o drużynach,
- adaptery wyświetlające informacje lub statystyki na temat lig piłkarskich,
- adaptery zwracające informacje o zwycięzcach lig, pucharów,
- adaptery służące do analizy danych (porównanie zawodników, drużyn).

Dzięki możliwości jaką daje biblioteka Chatterbot, czatbota można w każdej chwili rozbudować o nowe adaptery. W tym celu należy dodać nową instancję adaptera w głównym kodzie aplikacji, co przedstawia listing numer 11. W zależności od typu adaptera, nowa instancja może zawierać słowa kluczowe i etykiety.

```
goals_adapter = PlayerStatsAdapter(chatbot, ['goals', 'scored', 'goal', 'netted'], 'Goals', 6)
assists_adapter = PlayerStatsAdapter(chatbot, ['assists', 'assisted', 'assist'], 'Assists', 7)
nationality_adapter = PlayerStatsAdapter(chatbot, ['nationality', 'nation', 'origin', 'from',
'citizenship', 'birthplace', 'native'], 'Nationality', 4)
position_adapter = PlayerStatsAdapter(chatbot, ['position', 'role'], 'Position', 5) ...
```

Listing 11 Kod programu dodający nową instancję adaptera

Gotowe adaptery należy dodać do listy `logic_adapters` (która została zainicjowana przy początkowym tworzeniu czatbota). Odpowiada za to polecenie widoczne w listingu numer 12.

```
chatbot.logic_adapters.extend([goals_adapter, assists_adapter, nationality_adapter,
position_adapter, team_adapter])
```

Listing 12 Kod programu rozszerzający instancję czatbota o nowe adaptery

5.4. *Trenowanie modeli czatbota*

Czatbot to program, który powinien wykrywać intencje użytkownika. Aby mógł robić to dobrze, należy go do tego odpowiednio przygotować. Takie przygotowanie czatbota nazywamy trenowaniem. Dzięki technikom przetwarzania języka naturalnego czatbot jest w stanie nauczyć się wykrywać dla nas odpowiednie dane. Jest to niezwykle istotny element programu. Przy wyspecjalizowanym czat bocie jest to szczególnie ważne. Trening nowych modeli jest trudnym zadaniem. Bardzo często wytrenowane modele nie nadają się do użytku. Niestety jedynym rozwiązaniem jest eksperymentowanie. Należy testować różne dane wejściowe oraz różne parametry trenowania. Model może być „przetrenowany” i wykrywać za dużo niepotrzebnych informacji. Każdy model po treningu należy dokładnie przetestować. Po takim audycie można dodać model do instancji czatbota. Żaden model nie będzie idealny, składać się może na to kilka

czynników takich jak: zły dobór danych, źle dobrane parametry treningowe. Przy aspekcie jakim jest trenowanie modeli nie sposób wspomnieć o dużym problemie jakim jest moc obliczeniowa komputera na którym model jest tworzony. Im większy zbiór danych tym trenowanie modelu trwa dłużej [4]. Po przetestowaniu różnych modeli należy wybrać ten najlepszy. Nazwy własne stosowane w piłce nożnej mogą powodować błędy programu, dlatego konieczne jest prawidłowe wytrenowanie odpowiedniego modelu. Przykładowe trudności, z którymi musi radzić sobie czatbot:

- nazwiska piłkarzy mogą nawiązywać do rzeczy codziennego użytku np. piłkarz Kalvin Phillips do marki Phillips,
- nazwy drużyn mogą być nazwami miast np. AC Milan (Mediolan), Liverpool FC (Liverpool),
- nazwy drużyn mogą być do siebie bardzo podobne np. Manchester United, Manchester City (widoczne na rysunku numer 17)



Rys. 17 Podobne nazwy drużyn

(źródło: [21])

5.4.1. Przygotowanie danych treningowych

Istotnym elementem treningu nowego modelu, jest odpowiednie przygotowanie danych treningowych. Nowy model musi otrzymać odpowiednio dobrane dane tak, aby w przyszłości mógł wykryć informacje z pytania zadanego przez użytkownika. Uzyskane informacje czatbot jest w stanie przetworzyć na prawidłową odpowiedź. Najlepszym rozwiązaniem dla czatbota do rozmów o statystykach piłkarskich jest wykorzystanie danych z API, z którego będzie korzystał. Takie rozwiązanie pozwala dokładniej wykrywać intencje z zapytania, ponieważ czatbot trenowany jest na danych, z których będzie korzystał. W rozdziale 5.2.1 traktującym o modułach pozyskujących informacje do bazy danych opisywałem wykorzystanie takiego modułu do pobierania danych treningowych z API. Dzięki takiemu rozwiązaniu w stosunkowo prosty sposób program jest w stanie pobrać poprawne dane, wykorzystane następnie do treningu. Takie dane po sformatowaniu nadają się do wykorzystania w treningu. Chatbot wykorzystuje głównie dane o nazwiskach

piłkarzy, nazwach klubów, nazwach lig i pucharów. Przykładowe dane pobrane do lokalnej bazy danych przedstawione są na rysunku numer 18.

	id_player	full_name
	Filtr	Filtr
1	623	B. Mendy
2	630	F. Delph
3	883	Lee Grant
4	897	M. Greenwood
5	2281	Gary Cahill
6	2288	Daniel Noel Drinkwater
7	18826	Mark Noble
8	19308	L. Grabban
9	138411	L. Hughes

Rys. 18 Pobrane dane z nazwiskami piłkarzy w tabeli

(źródło: opracowanie własne)

5.4.2. Programy do trenowania modeli

W trenowaniu modeli na danych z API najlepiej sprawdziła się biblioteka spaCy, która w przypadku takich danych wykorzystuje tokenizację. Każdy tekst, który przechodzi przez funkcję `nlp.make_doc()` jest tokenizowany. Następnie tekst jest przetwarzany przez model, a spaCy dzięki swoim funkcjom tokenizuje go i tworzy obiekt `Doc`, zawierający informacje o tokenach w zdaniu. Tworzenie przykładów treningowych również realizowane jest przy pomocy tokenizacji. Zrealizowany tekst następnie zostaje przekazany w formacie encji. W listingach numer 13, 14 i 15 zostały przedstawione fragmenty programu do trenowania nowego modelu NER.

```
nlp = spacy.load("en_core_web_sm")

with open('player_names.txt', 'r', encoding='utf-8') as f:
    player_names = [line.strip() for line in f.readlines()]

TRAIN_DATA = []
for player_name in player_names:
    lowercase_name = player_name.lower()
    uppercase_name = player_name.upper()
    titlecase_name = player_name.title()

    combined_names = [player_name, lowercase_name, uppercase_name, titlecase_name]
```

Listing 13 Fragment kodu do trenowania modeli NER (przygotowanie danych treningowych)

Najlepszym rozwiązaniem jest wczytanie modelu językowego i dostosowanie go pod odpowiednie dla nas parametry. W tym przypadku encje „PERSON” dla nazwisk piłkarzy. Dane wczytane są z pliku tekstowego, następnie mieszane są wielkości znaków. Taka czynność ma na celu rozszerzenie danych treningowych. Została zaprezentowana w listingu numer 13.

```
for combined_name in combined_names:
    complex_sentences = [
        f"How many goals has {combined_name} scored this season?",
        f"goals {combined_name}",
        f"How many assists has {combined_name} provided in the league?",
        f"assist {combined_name}",
        f" {combined_name} assist",
        f"What is the nationality of {combined_name}?",
        f"Where is {combined_name} from?",
        f"What position does {combined_name} play?",
        f"Can you tell me the playing position of {combined_name}?",
        f"What team does {combined_name} currently play for?",
        f"Which squad does {combined_name} belong to?",
    ]
    for sentence in complex_sentences:
        entities = [
            (sentence.index(combined_name), sentence.index(combined_name) +
len(combined_name), "PERSON")
        ]
        TRAIN_DATA.append((sentence, {"entities": entities}))
```

Listing 14 Fragment kodu do trenowania modeli NER (przygotowanie zdań treningowych)

Następnie generowane są różne zdania, w których używa się nazwisk piłkarzy. Fragment kodu odpowiadający za to zadanie przedstawiony jest w listingu numer 14. Nazwiska są specjalnie etykietowane tak, aby program wiedział, że to ma być oznaczenie encji „PERSON”

```
n_iter = 30
optimizer = nlp.resume_training()
other_pipes = [pipe for pipe in nlp.pipe_names if pipe != "ner"]

with nlp.disable_pipes(*other_pipes):
    best_loss = float("inf")
    for i in range(n_iter):
        random.shuffle(TRAIN_DATA)
        losses = {}
        for batch in minibatch(TRAIN_DATA, size=compounding(4.0, 32.0, 1.001)):
            texts, annotations = zip(*batch)
            examples = [Example.from_dict(nlp.make_doc(text), ann) for text, ann in zip(texts,
annotations)]
```

```
nlp.update(examples, drop=0.3, losses=losses)
print("Iteracja", i, "Strata", losses)
if losses['ner'] < best_loss:
    best_loss = losses['ner']
    nlp.to_disk("custom_ner_model_player_names")
    print(f"Zapisano najlepszy model w iteracji {i}, Strata: {losses['ner']}")
```

Listing 15 Fragment kodu do trenowania modeli NER (właściwe trenowanie)

W tym fragmencie kodu z listingu numer 15, następuje właściwe trenowanie modelu, należy ustawić odpowiednio parametr ‘drop’ tak, aby model nie został przetrenowany. Oprócz wielkości danych treningowych znaczny wpływ na czas tworzenia nowego modelu ma liczba iteracji. Im większa tym większe prawdopodobieństwo stworzenia lepszego modelu. Ten program porównuje model po każdej iteracji i zapisuje jedynie ulepszoną wersję. Im mniejsza strata (wskaźnik ner) tym lepiej dla modelu. Na rysunku numer 19 przedstawiony jest program w trakcie działania.

```
Iteracja 0 Strata {'ner': 74.6450577271664}
Zapisano najlepszy model w iteracji 0, Strata: 74.6450577271664
Iteracja 1 Strata {'ner': 1.611756489042345}
Zapisano najlepszy model w iteracji 1, Strata: 1.611756489042345
Iteracja 2 Strata {'ner': 11.879714235342892}
Iteracja 3 Strata {'ner': 3.064192849548493}
Iteracja 4 Strata {'ner': 13.174208560675718}
Iteracja 5 Strata {'ner': 3.2088608752856673e-06}
Zapisano najlepszy model w iteracji 5, Strata: 3.2088608752856673e-06
Iteracja 6 Strata {'ner': 0.00240122438909174}
Iteracja 7 Strata {'ner': 4.212075749330384}
Iteracja 8 Strata {'ner': 4.071042503260343}
Iteracja 9 Strata {'ner': 2.0018695543103013}
Iteracja 10 Strata {'ner': 7.662132596564098e-06}
```

Rys. 19 Działanie programu do trenowania modeli

(źródło: opracowanie własne)

Wytrenowany model można przetestować przy pomocy prostego programu, przedstawionego w listingu numer 16. Program wykrywa encje z tekstu. Jego działanie przedstawione jest na rysunku numer 20, program wydziela z tekstu jedynie te słowa, które jest w stanie dopasować do określonych encji.

```
import spacy
nlp = spacy.load("custom_ner_model_player_names")
test_texts = ["tell me about Messi, he played in Barcelona, if ronaldo tells me a story, i tell you a better story says Kane. "]
for test_text in test_texts:
    doc = nlp(test_text)
    print(f"\nTest: {test_text}")
    for ent in doc.ents:
        print(ent.text, ent.label_)
```

Listing 16 Kod programu do testowania encji

```
Test: tell me about Messi, he played in Barcelona, if ronaldo tells me a story, i tell you a better story says Kane.
Messi PERSON
Barcelona SPORT_TEAM
ronaldo PERSON
Kane PERSON
```

Rys. 20 Wynik działania programu do testowania modelu (źródło: opracowanie własne)

5.5. Testowanie aplikacji

Dzięki odpowiedniemu przygotowaniu czatbota, jest gotowy do testów. Poniżej zostały przedstawione przykładowe rozmowy z botem. Oczywiście czatbot popełnia błędy, ponieważ nie jest to program doskonały jednak dzięki możliwościom zapisu historii czatów, można poprawić źle przetworzone zapytania w kolejnej wersji programu. Jeżeli czatbot nie zrozumie zapytania wyświetla odpowiedni monit. Na podstawie tych monitów można przeszukać bazę danych i poprawić działanie programu.

650	tell me about Milan stadium
651	I'm sorry, I don't understand your query.

Rys. 21 Przegląd bazy danych z niezrozumiałym pytaniem

(źródło: opracowanie własne)

You: *hello*

bot: Hi there!

You: *tell me about Kane last year*

bot: Here is some information about Kane:

Name: H. Kane
 Full Name: Harry Edward Kane
 Player age: 31
 Player Nationality: England
 Player position: Attacker
 Player goals: 36
 Player assists: 8
 Player team: Bayern München

You: *compare kane to Lewandowski*

bot:

Attribute	Player 1	Player 2
Name	H. Kane	R. Lewandowski
Age	31	36
Nationality	England	Poland
Player Team	Bayern München	Barcelona
Position	Attacker	Attacker
Appearances	32	35
Total Goals	36	19
Total Assists	8	8

You: *barcelona league position last year*

bot:

Rank	Team	Points	Goals Difference
2	Barcelona	85	35

You: *barcelona league position*

bot:

Rank	Team	Points	Goals Difference
1	Barcelona	12	10

Rys. 22 Przykładowa rozmowa użytkownika z czatbotem 1
 (źródło: opracowanie własne)

You: *give me last matches between milan and Chelsea*

bot:

Date	Home Team	Away Team	Result	Score	Winner
2016-08-04T01:35:00+00:00	AC Milan	Chelsea	FT	1-3	Chelsea
2022-10-11T19:00:00+00:00	AC Milan	Chelsea	FT	0-2	Chelsea
2022-10-05T19:00:00+00:00	Chelsea	AC Milan	FT	3-0	Chelsea

You: *League table premier league*

bot:

Rank	Team	Points	Goals Difference
1	Manchester City	9	7
2	Liverpool	9	7
3	Brighton	7	4
4	Arsenal	7	4
5	Newcastle	7	2
6	Brentford	6	1
7	Aston Villa	6	0
8	Bournemouth	5	1
9	Nottingham Forest	5	1
10	Tottenham	4	3
11	Chelsea	4	2
12	Fulham	4	0
13	West Ham	3	-1
14	Manchester United	3	-3
15	Leicester	1	-2
16	Crystal Palace	1	-3
17	Ipswich	1	-5
18	Wolves	1	-6
19	Southampton	0	-4
20	Everton	0	-8

You:

Rys. 23 Przykładowa rozmowa użytkownika z czatbotem 2
(źródło: opracowanie własne)

6. PODSUMOWANIE

Celem pracy było stworzenie czatbota do rozmów o statystykach piłkarskich z wbudowaną analizą danych, cel ten został osiągnięty. Dzięki szerokim możliwościom bibliotek Pythona i rozbudowanym narzędziom przetwarzania języka naturalnego czatbot stał się bardziej komunikatywny. Biblioteka Chatterbot pozwoliła na zbudowanie instancji czatbota, a biblioteka spaCy pozwoliła na trening nowych wyspecjalizowanych modeli. Odpowiednie wytrenowanie czatbota to niezwykle trudne i czasochłonne zadanie. Zaletą stworzonej aplikacji jest możliwość ulepszania jej o nowe lepsze modele. Dzięki początkowym założeniom o możliwości rozbudowy oraz modyfikacji programu, czatbot może być stale ulepszany przez dodanie nowych modułów oraz adapterów. W niektórych przypadkach nie obsługuje odpowiedzi natychmiast, ale użycie zapisu czatów pozwala w łatwy sposób korygować błędy. Z racji wyspecjalizowania czatbota w jednej dziedzinie, radzi on sobie słabiej przy ogólnej rozmowie. Jednak nie taki był cel jego powstania.

BIBLIOGRAFIA

- [1] D. S. Regin, D. S. S. Rajest, S. T, J. Angelene Christabel G and S. R, An Automated Conversation System Using Natural Language Processing (NLP) Chatbot in Python, Central Asian Journal of Medical and Natural Sciences, 2022.
- [2] C. Petzold, The Annotated Turing, Wiley Publishing, Inc, 2008.
- [3] A. Shevat, Zaprojektuj bota. Tworzenie interfejsów konwersacyjnych, Helion SA, 2018.
- [4] S. Raj, Building Chatbots with Python Using Natural Language Processing and Machine Learning, Apress, 2019.
- [5] A. Kozłowska i A. Rodzik, CHATBOTY: PERSPEKTYWY ROZWOJU TECHNOLOGII INFORMATYCZNYCH W KONTAKCIE Z KLIENTEM, Acta Universitatis Nicolai Copernici, 2018.
- [6] M. Negnevitsky, Artificial Intelligence A Guide to Intelligent Systems, Pearson Education Canada, 2011.
- [7] L. Hobson, C. Howard i H. Hapke, Przetwarzanie języka naturalnego w akcji, Wydawnictwo Naukowe PWN SA, 2021.
- [8] „Czym jest TF*IDF i jak wpływa na pozycjonowanie,” [Online]. Dostępny: <https://www.sunrisesystem.pl/blog/czym-jest-tfidf-i-jak-wplywa-na-pozycjonowanie/>. [Data uzyskania dostępu: Czerwiec 2024].
- [9] M. Müller, "Comparing Text Similarity Measurement Methods: Sentence Transformers vs. Fuzzy," [Online]. Dostępny: <https://dev.to/miguelsmuller/comparing-text-similarity-measurement-methods-sentence-transformers-vs-fuzzy-og3>. [Accessed Sierpień 2024].
- [10] D. Jurafsky i J. H. Martin, Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, Stanford University, 2024.
- [11] L. Moroney, AI and Machine Learning for Coders, O'Reilly Media, 2020.
- [12] „Dokumentacja Python,” [Online]. Dostępny: <https://docs.python.org/pl/3/reference/index.html>. [Data uzyskania dostępu: Sierpień 2024].
- [13] S. Bird, E. Klein i E. Loper, Natural Language Processing with Python, O'Reilly Media, 2009.
- [14] „Dokumentacja i opis biblioteki nltk,” [Online]. Dostępny: <https://github.com/nltk/nltk/wiki>. [Data uzyskania dostępu: Sierpień 2024].
- [15] „Dokumentacja biblioteki spaCy,” [Online]. Dostępny: <https://spacy.io/api/doc>. [Data uzyskania dostępu: Sierpień 2024].
- [16] „Dokumentacja biblioteki Chatterbot,” [Online]. Dostępny: <https://chatterbot.readthedocs.io/en/stable/index.html>. [Data uzyskania dostępu: Sierpień 2024].

- [17] B. Jin, S. Sahni i A. Shevat, Designing Web APIs: Building APIs That Developers Love, O'Reilly Media, 2018.
- [18] „Dokumentacja api-football.com,” [Online]. Dostępny: <https://api-sports.io/documentation/football/v3>. [Data uzyskania dostępu: Sierpień 2024].
- [19] „Dokumentacja sqlite,” [Online]. Dostępny: <https://docs.python.org/3/library/sqlite3.htm>. [Data uzyskania dostępu: Sierpień 2024].
- [20] „Dokumentacja biblioteki tabulate,” [Online]. Dostępny: <https://pypi.org/project/tabulate/>. [Data uzyskania dostępu: Sierpień 2024].
- [21] „Zapowiedź: Manchester United – Manchester City,” [Online]. Dostępny: <https://www.manutd.pl/2021/11/05/zapowiedz-manchester-united-manchester-city-11-kolejka-premier-league/>. [Data uzyskania dostępu: Sierpień 2024].

SPIS RYSUNKÓW

Rys. 1 Współpraca interfejsów	7
Rys. 2 Rozmowa z czatbotem x-kom	9
Rys. 3 LawGeex - bot będący radcą prawnym	9
Rys. 4 Google Assistant opowiadający żart	10
Rys. 5 Uproszczony schemat działania tokenizacji.....	12
Rys. 6 Uproszczony Schemat działania wektorów TF-IDF	14
Rys. 7 Wektory w przestrzeni 2D.....	15
Rys. 8 Wizualizacja wektorów na przykładzie słów	16
Rys. 9 Logo Chatterbot.....	21
Rys. 10 Działanie programu przedstawiającego informacje o tabeli ligowej	25
Rys. 11 Podstawowe działanie czatbota	27
Rys. 12 Poprawne zwrócenie przez moduł ID piłkarza.....	30
Rys. 13 Zwrócenie przez moduł piłkarzy do wyboru.....	31
Rys. 14 Zwrócone dane przez moduł dla określonego zawodnika.....	33
Rys. 15 Porównanie dwóch drużyn i wyników ostatnich meczów	34
Rys. 16 Przykładowe działania adaptera	38
Rys. 17 Podobne nazwy drużyn	40
Rys. 18 Pobrane dane z nazwiskami piłkarzy w tabeli.....	41
Rys. 19 Działanie programu do trenowania modeli	43
Rys. 20 Wynik działania programu do testowania modelu (źródło: opracowanie własne)	44
Rys. 21 Przegląd bazy danych z niezrozumiałym pytaniem	44
Rys. 22 Przykładowa rozmowa użytkownika z czatbotem 1	45
Rys. 23 Przykładowa rozmowa użytkownika z czatbotem 2	46

SPIS LISTINGÓW

Listing 1 Kod programu odpowiedzialny za analizę tabeli ligowej ligi włoskiej	24
Listing 2 Kod programu odpowiadający za stworzenie instancji czatbota	26
Listing 3 Kod odpowiedzialny za rozszerzanie czatbota o nowe adaptory, oraz za nieskończoną pętlę czatbota.	26
Listing 4 Kod programu do pobierania identyfikatorów drużyn	28
Listing 5 Kod programu do odnajdywania identyfikatora zawodnika	30
Listing 6 Kod programu do pozyskiwania danych na temat zawodnika	32
Listing 7 Kod programu zawierającego moduł do analizy danych dwóch drużyn.....	34
Listing 8 Kod programu do inicjalizacji instancji adaptera.....	35
Listing 9 Kod programu do ładowania etykiet dla odpowiedzi.....	36
Listing 10 Kod programu do przetworzenia zapytań i generowania odpowiedzi.	38
Listing 11 Kod programu dodający nową instancję adaptera.....	39
Listing 12 Kod programu rozszerzający instancję czatbota o nowe adaptory.....	39
Listing 13 Fragment kodu do trenowania modeli NER (przygotowanie danych treningowych)..	41
Listing 14 Fragment kodu do trenowania modeli NER (przygotowanie zdań treningowych).....	42
Listing 15 Fragment kodu do trenowania modeli NER (właściwe trenowanie)	43
Listing 16 Kod programu do testowania encji.....	44