Dokumentacja techniczna – WhisperPress

Autorzy: Marta Głuszkowska, Emilia Kłobukowska, Piotr Bubeła, Kacper Tomczak

Spis treści

Wstęp

Celem projektu jest stworzenie systemu monitorującego i analizującego treści internetowe dotyczące piłki nożnej. Nasz serwis pobiera informacje o zawodnikach z portali Twitter, Transfermarkt oraz 90minut.pl, nadaje im odpowiednie tagi i prezentuje zebrane informacje na portalu internetowym.

Aplikacja składa się z następujących komponentów:

1. Aplikacja do pobierania i analizowania danych (stworzona w technologii Django)
2. Aplikacja internetowa służąca do przechowywania i prezentowania danych:
    a. Frontend aplikacji (napisany w technologii Angular)
    b. Backend aplikacji (stworzony przy użyciu technologii .NET core 3.1, Entity Framework Core oraz LINQ)
    c. Relacyjna baza danych T-SQL

Aplikacja internetowa

Frontend aplikacji

Autor: Piotr Bubeła

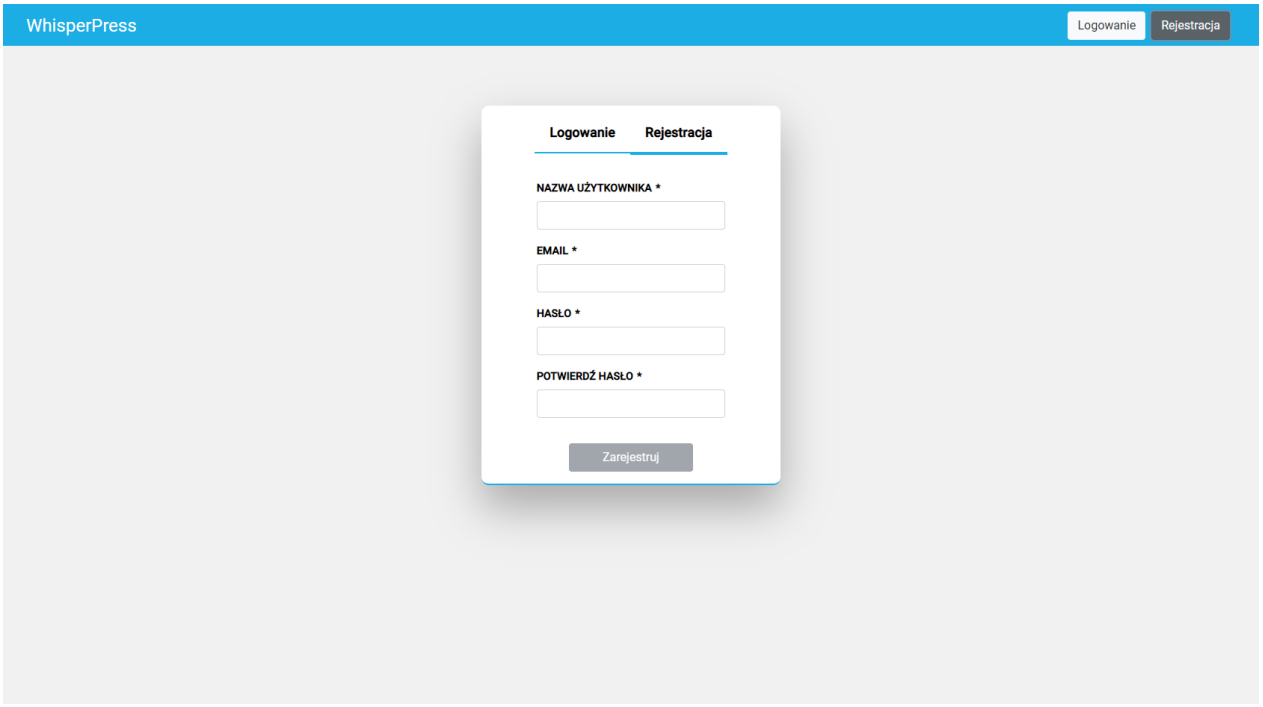
Frontend aplikacji internetowej służy do prezentacji danych użytkownikom. Został wykonany przy użyciu technologii Angular, HTML oraz CSS.

Przegląd widoków

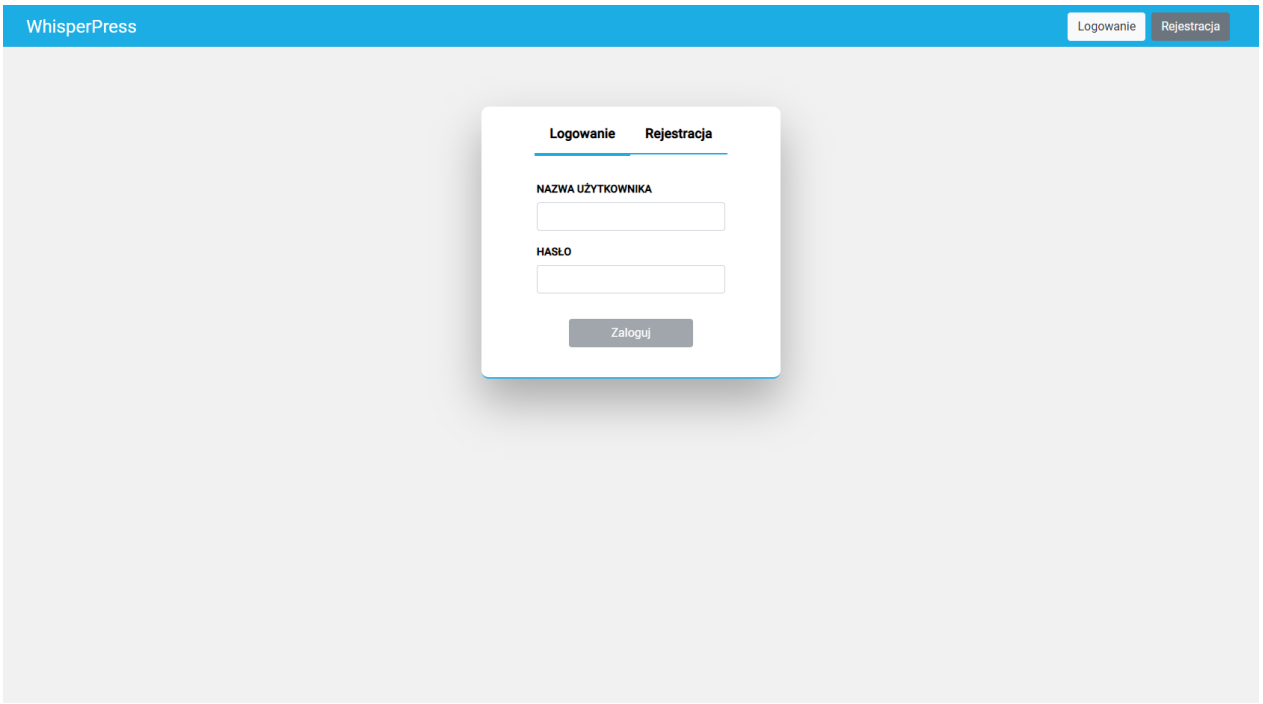
Poniższe zrzuty ekranu przedstawiają poszczególne elementy frontendu aplikacji internetowej wraz z opisem ich funkcji.

Strona główna (użytkownik niezalogowany)

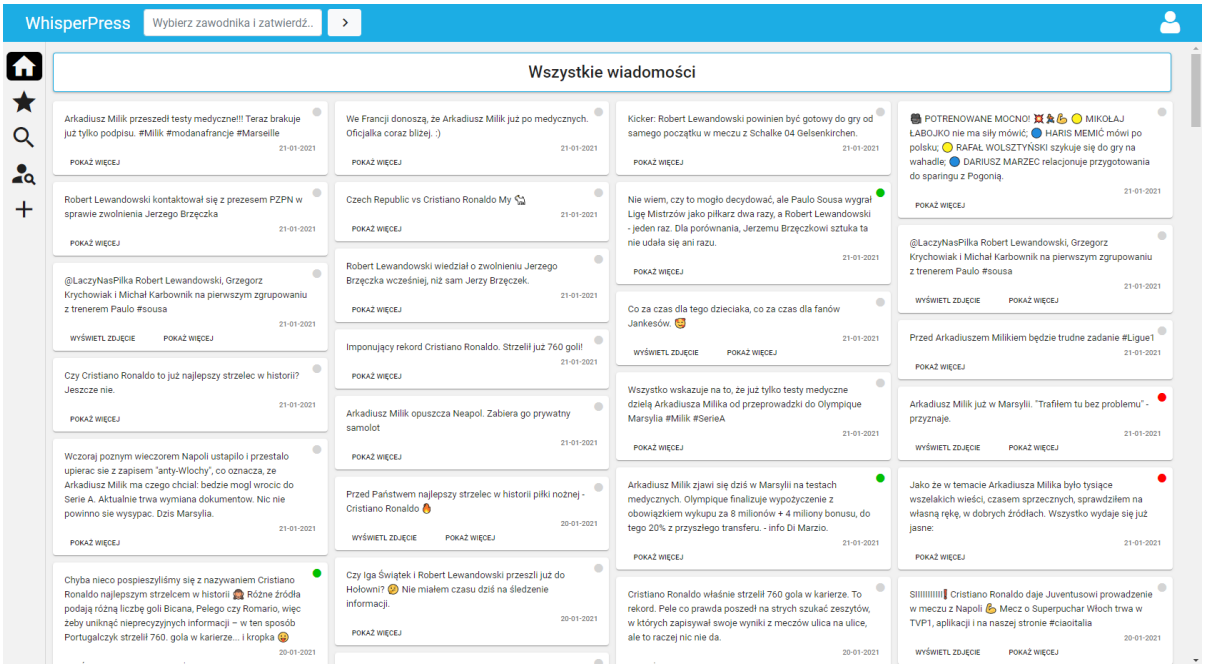
Jest to widok powitalny dla niezalogowanych użytkowników, przedstawiający informację o tym, co znajdziemy w systemie po zalogowaniu. Jest to jedyna funkcjonalność dostępna dla niezalogowanych użytkowników.



Strona rejestracji

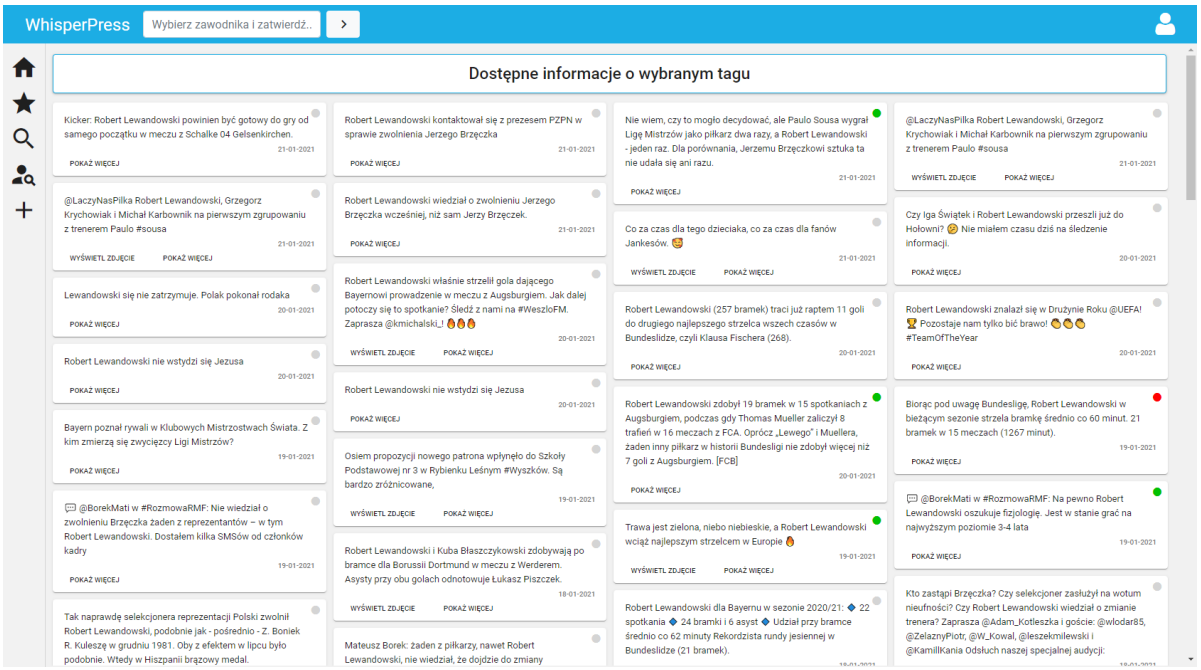


Strona logowania

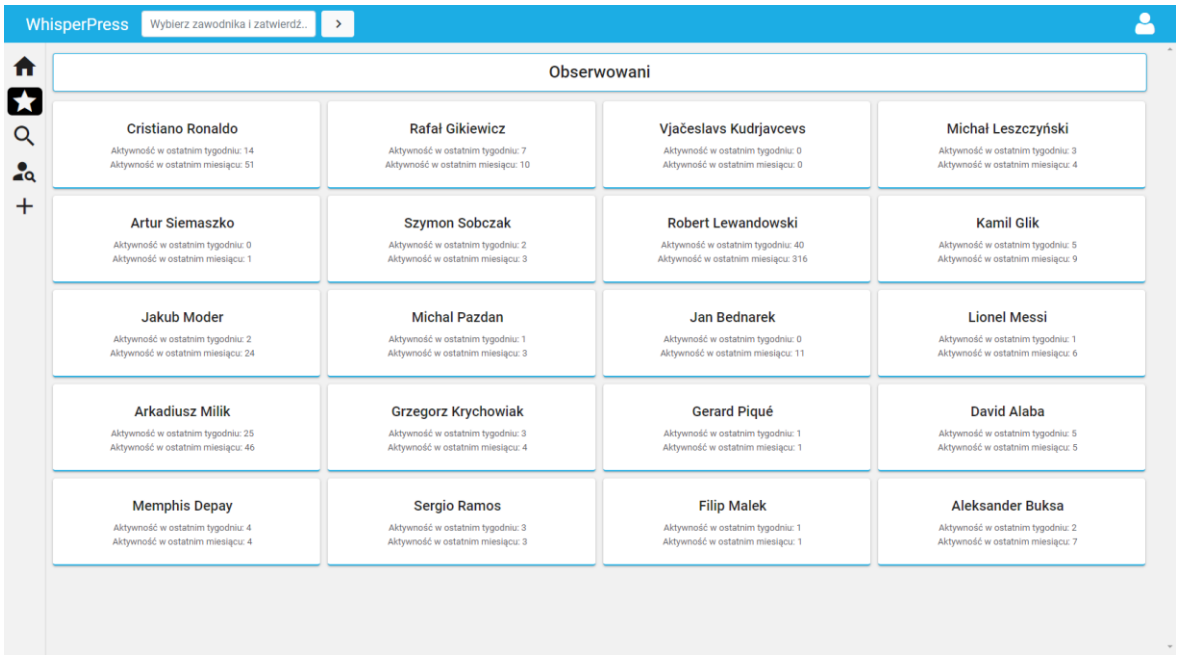


Strona główna (użytkownik zalogowany)

Panel przedstawia w jednym miejscu wszystkie newsy pobrane do systemu. Użytkownik może rozwinąć dodatkowe informacje na temat newsa, takie jak tagi, odnośnik do źródła oraz zdjęcie.

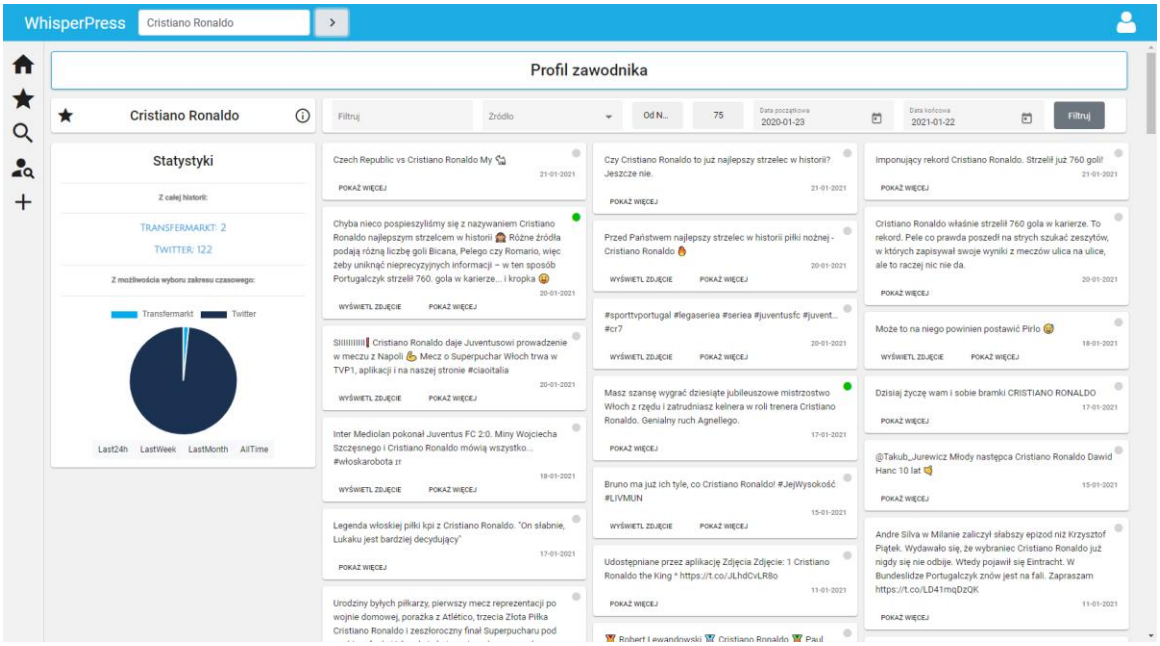


Strona z wiadomościami zawierającymi wybrany tag



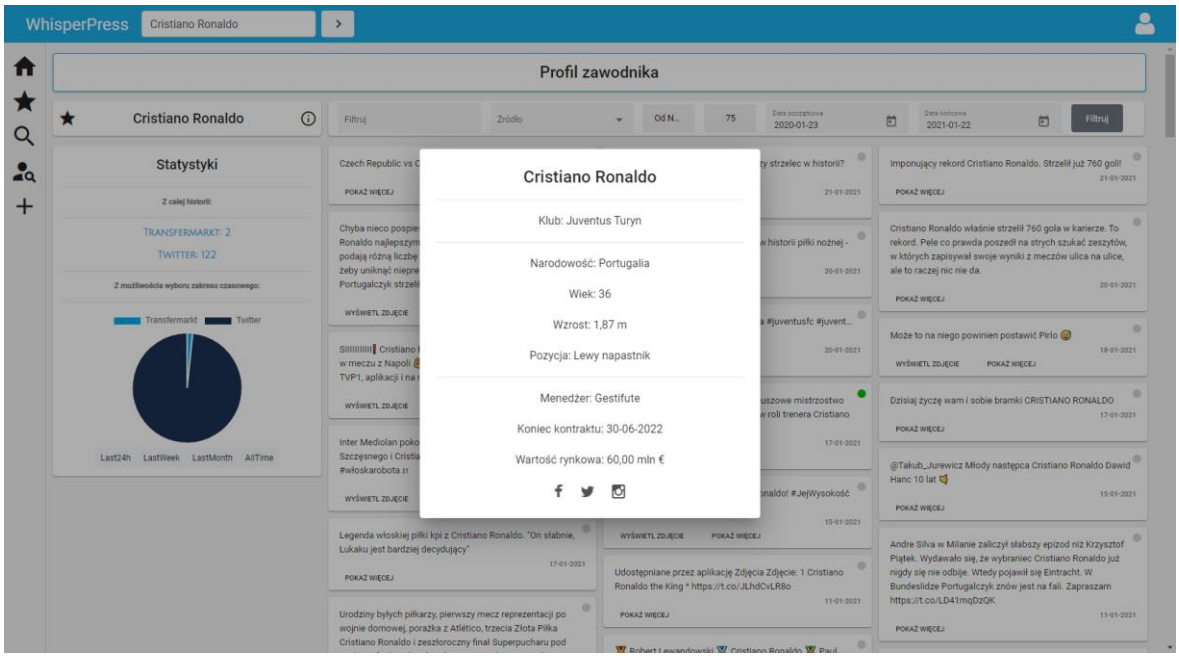
Panel obserwowanych zawodników

Zawiera listę wszystkich zawodników, którzy zostali dodani przez użytkownika do ulubionych wraz z informacją na temat postów, które pojawiły się w systemie dla danego zawodnika w ostatnim tygodniu i miesiącu.

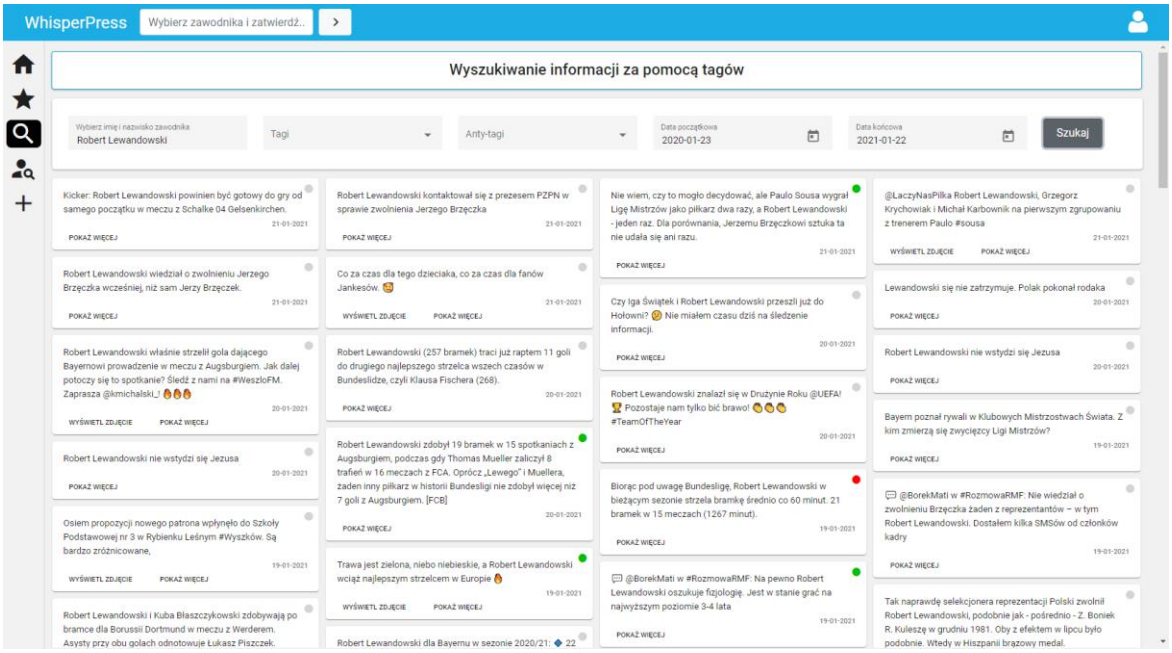


Panel zawodnika

Zawiera wszystkie newsy dotyczące danego zawodnika z możliwością ich filtrowania oraz podstawowe statystyki.

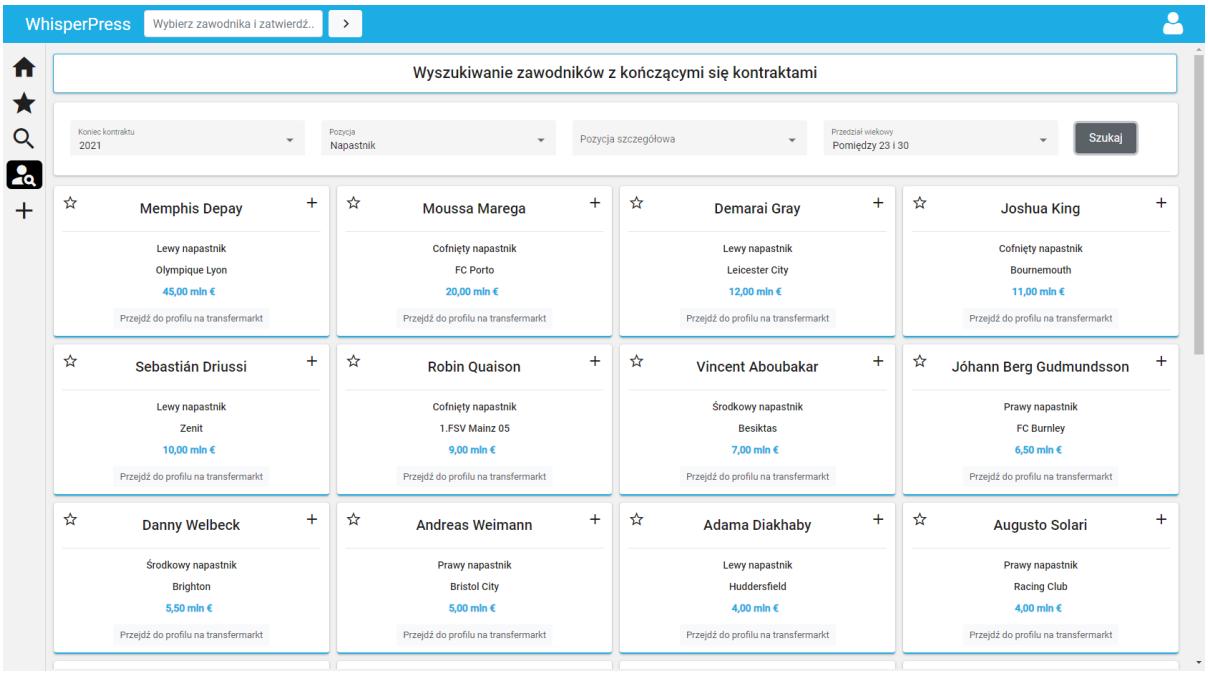


Profil zawodnika – panel szczegółowych informacji o zawodniku



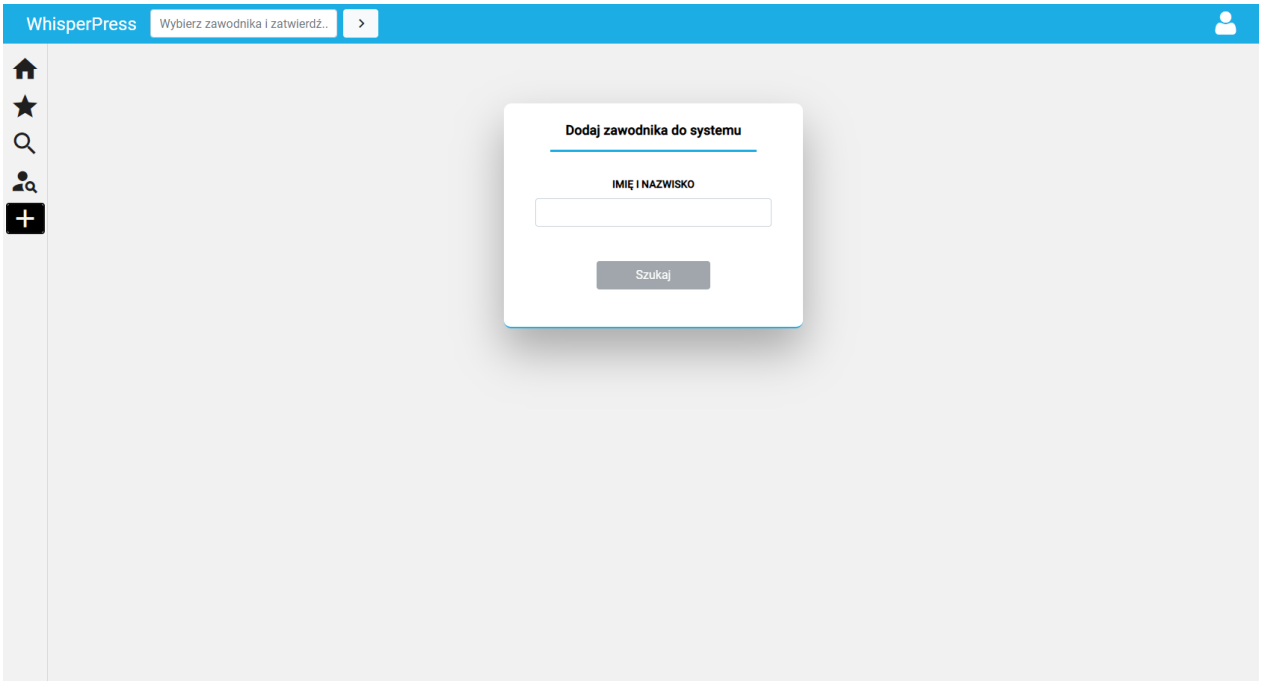
Panel wyszukiwania informacji za pomocą tagów

Strona ta umożliwia wyszukiwanie konkretnych newsów za pomocą filtrów wyszukiwania, takich jak imię i nazwisko zawodnika, tagi, antytagi oraz ramy czasowe

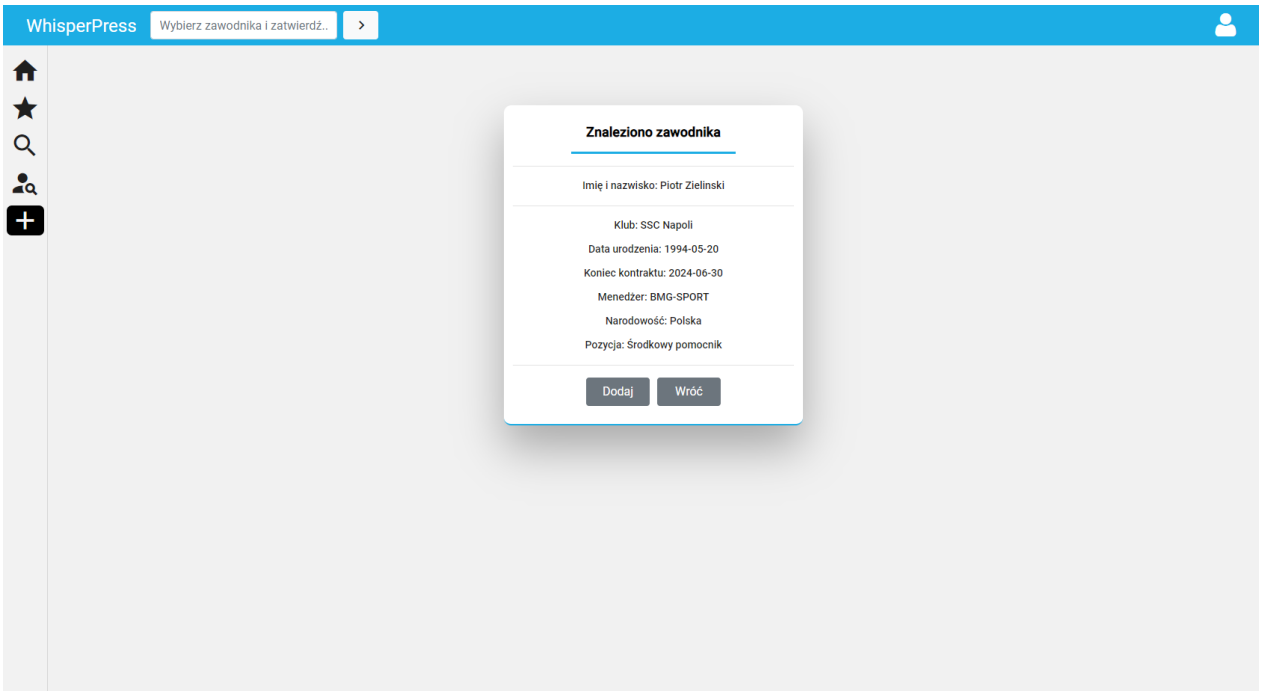


Panel wyszukiwania zawodników z kończącymi się kontraktami

Użytkownik może pobrać listę zawodników, którym kończą się kontrakty, zgodnie z konkretnymi filtrami wyszukiwania. Z tego panelu użytkownik może dodać wybranych zawodników do systemu oraz do swoich ulubionych zawodników.

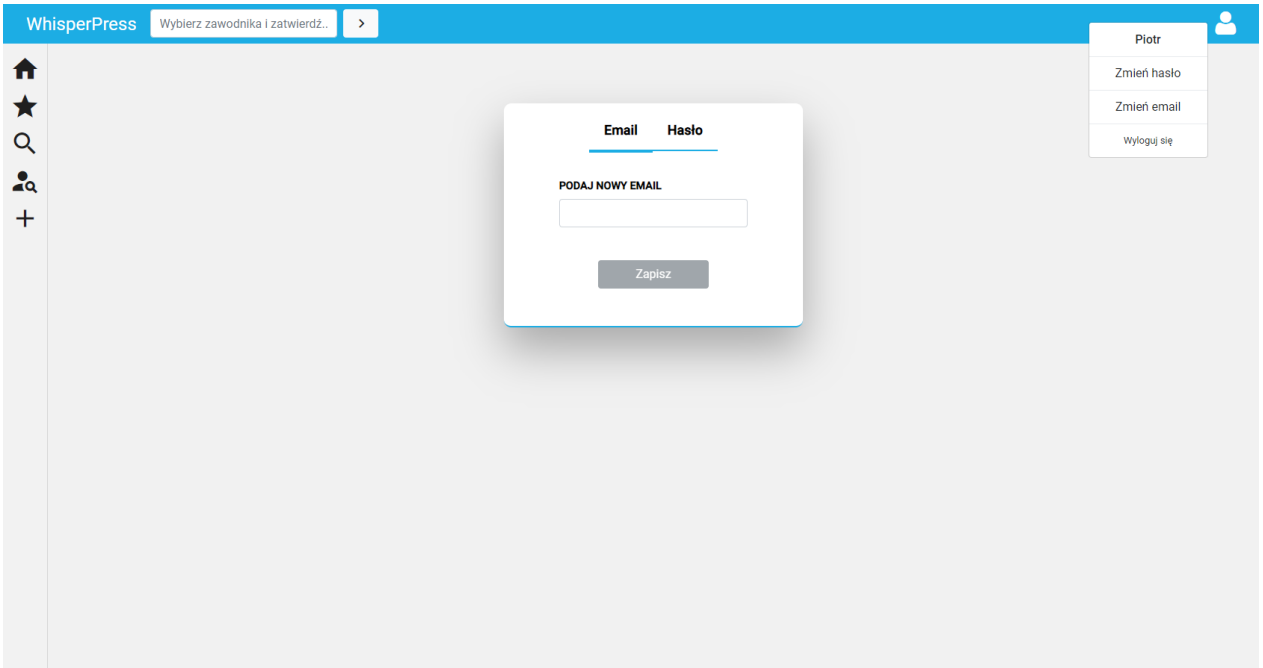


Panel dodawania zawodników do systemu (przed podaniem nazwiska)

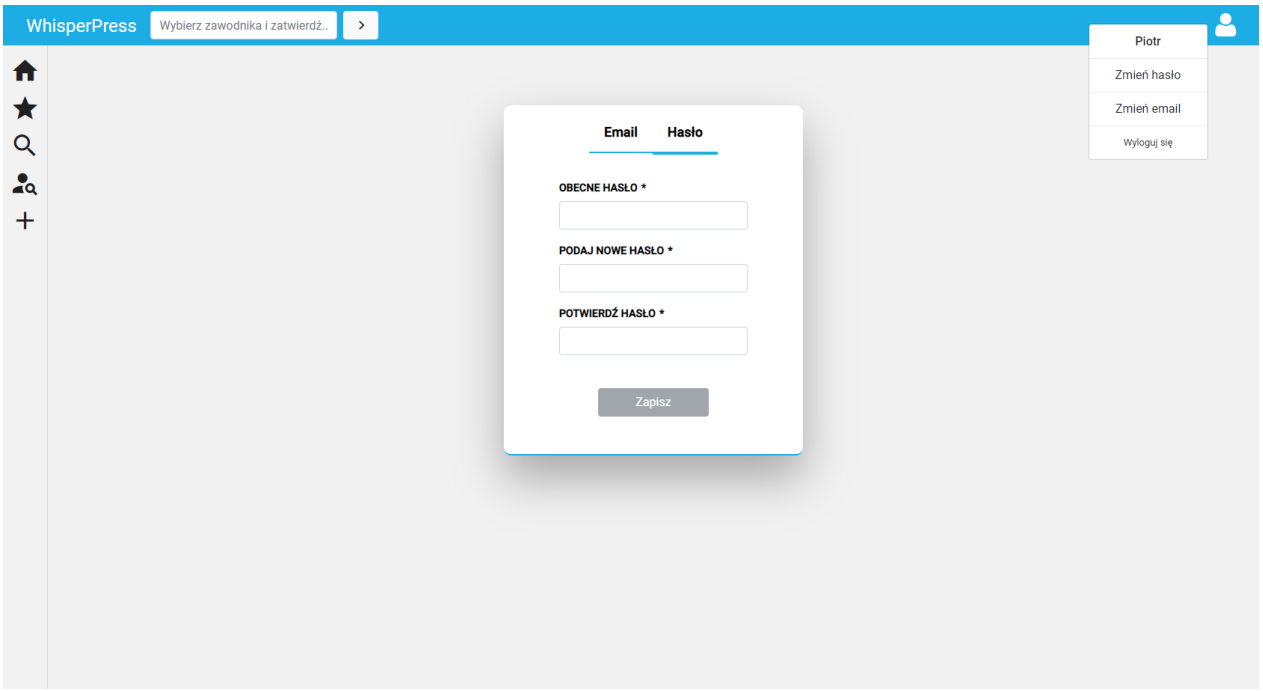


Panel dodawania zawodników do systemu (po podaniu nazwiska)

Po wpisaniu odpowiedniego nazwiska, wyświetlają się szczegółowe informacje na temat zawodnika. Użytkownik może zdecydować, czy chce dodać zawodnika do systemu.



Panel zmiany adresu e-mail użytkownika



Panel zmiany hasła

Backend aplikacji

Autor: Kacper Tomczak

Backend aplikacji intenetowej został stworzony przy użyciu technologii .NET core 3.1, Entity Framework Core oraz LINQ.

Endpointy

Poniższa tabela przedstawia wszystkie endpointy wykorzystane w backendzie aplikacji:

ENDPOINT	METODA	OPIS

/api/Account /Login	POST	Request body:
		{
		"userName": "string",
		"password": "string"
		}
		
		Endpoint umożliwia zalogowanie, zwraca token autoryzujący

/api/Account /Register	POST	Request body:
		
		{
		"userName": "string",
		"email": "user@example.com",
		"password": "string",
		"confirmPassword": "string"
		}
		
		Endpoint umożliwia rejestracje użytkownika, zwraca kod 200 oraz informacje potwierdzającą rejestracje, lub liste errorów

/api/Commu nication/Expi ringContract s	GET	Wymagany query param to “Year”, więc prawidłowo wykorzystany endpoint wygląda tak: /api/Communication/ExpiringContracts?Year=2020 Pozostałe query param to: PlayerPosition, AgeGroup, PlayerPositionDetails, Nationality, wymaga niemieckich nazw analogicznie do wyszukiwania na portalu Transfermarkt. Np. dla wszystkich pozycji zawodnika będzie to: GET: /api/Communication/ExpiringContracts?Year=2020&PlayerPosition= alle Zwraca listę do 25zawodników pasujących do parametrów wyszukiwania.

/api/Communication/PlayerTM/{playerFullName}	GET	{playerFullName} to imie I nazwisko zawodnika np. GET: /api/Communication/PlayerTM/Robert lewandowski Zwraca informacje: {"Name":"Robert","Surname":"Lewandowski","DateOfBirth":"1988-08-21T00:00:00+02:00","PlaceOfBirth":"Warszawa ","Nationality":"Polska","Position":"Środkowy napastnik ","ContractUntil":"2023-06-30","Height":"1,85 m ","Manager":"Pinhas Zahavi ","MarketValue":"60,00 mln €","Club":"Bayern Monachium ","SocialMedia":[{"Title":"Twitter","URL":"http://twitter.com/lewy_official"},{"Title":"Facebook","URL":"http://www.facebook.com/rl9official"},{"Title":"Instagram","URL":"http://www.instagram.com/_rl9/"},{"Title":"","URL":"http://www.lewandowskiofficial.com"}]}
/api/Tag	GET	Zwraca listę wszystkich tagów w aplikacji

Poniższe endpointy wymagają zalogowania / wysłania w headerze Tokenu:

Parametr: Authorization: Bearer {TOKEN}

ENDPOINT	METODA	OPIS

/api/Account/ ChangePassword	PUT	Request body:
		
		{
		"oldPassword": "string",
		"newPassword": "string",
		"confirmPassword": "string"
		}
		
		Zwraca kod 200 z informacją o pomyślności operacji lub listę błędów

/api/Account/ ChangeEmail	PUT	Request body:
		
		{
		"email": "user@example.com"
		}
		
		Umożliwia zmianę maila, zwraca kod 200 z informacją operacji lub błąd walidacji.

/api/News/ NewsAssignedToPlayer	GET	Zwraca stronicową liste newsów dla pojedynczego zawodnika. Wymagane QueryParams to: “PageNumber”, “PageSize” I “PlayerId”. Pozostałe QueryParam to string “TextInput”, date “StartDate”, date “EndDate”, int “SortOrder”, IEnumerable<string> “Sources”
/api/News/FullSearch	GET	Zwraca stronicową listę newsów, wymagane QueryParams to: “PageNumber”, “PageSize”. Dodatkowe możliwe to: “PlayerId”.

		Pozostałe QueryParam to string “TextInput”, date “StartDate”, date “EndDate”, int “SortOrder”, IEnumerable<string> “Sources”, IEnumerable<string> “Tags”, IEnumerable<string> “AntiTags”
/api/News/ NewsForSingleTag	GET	Zwraca stronicową liste newsów dla pojedynczego tagu. Wymagane QueryParams to PageNumber, PageSize. Pozostałe QueryParam to TextInput, StartDate, EndDate, SortOrder I Tag. Gdy w query stringu nie znajdzie się parametr Tag, zwróci pustą listę.
/api/News/ById/{id}	GET	Zwraca News po danym id, jeżeli nie ma w bazie to zwraca 404.
/api/News/NewsStats/ All	GET	Zwraca informacje dot. ilości newsów z danych źródeł dla wszystkich newsów w bazie danych.
/api/News/ TimeCategorizedStats	GET	Zwraca ilość newsów z danego źródła w kategoriach czasowych: ostatnie 24h, tydzień, miesiąc i allTime. Można jako query param dodać: playerId(/api/News/TimeCategorizedStats?playerId=1), wtedy zwróci to samo dla newsów dot. danego zawodnika.
/api/News/ DayByDayNewsStats ForPlayer	GET	Zwraca ilość newsów z danego źródła dzień po dniu dla danego zawodnika, wymagane queryParams to: PayerId, StartDate i EndDate
/api/News/ PagedAllNews	GET	Zwraca wszystkie newsy w postaci stronicowej listy, wymagane query Params to: PageNumber(od 1), i PageSize(od 1). Dodatkowo można dodać parametry: StartDate, EndDate w postaci mm-dd-yyyy, oraz SortOrder(0(dla Ascending) lub 1(dla Descending),) domyślne sort order to Ascending.

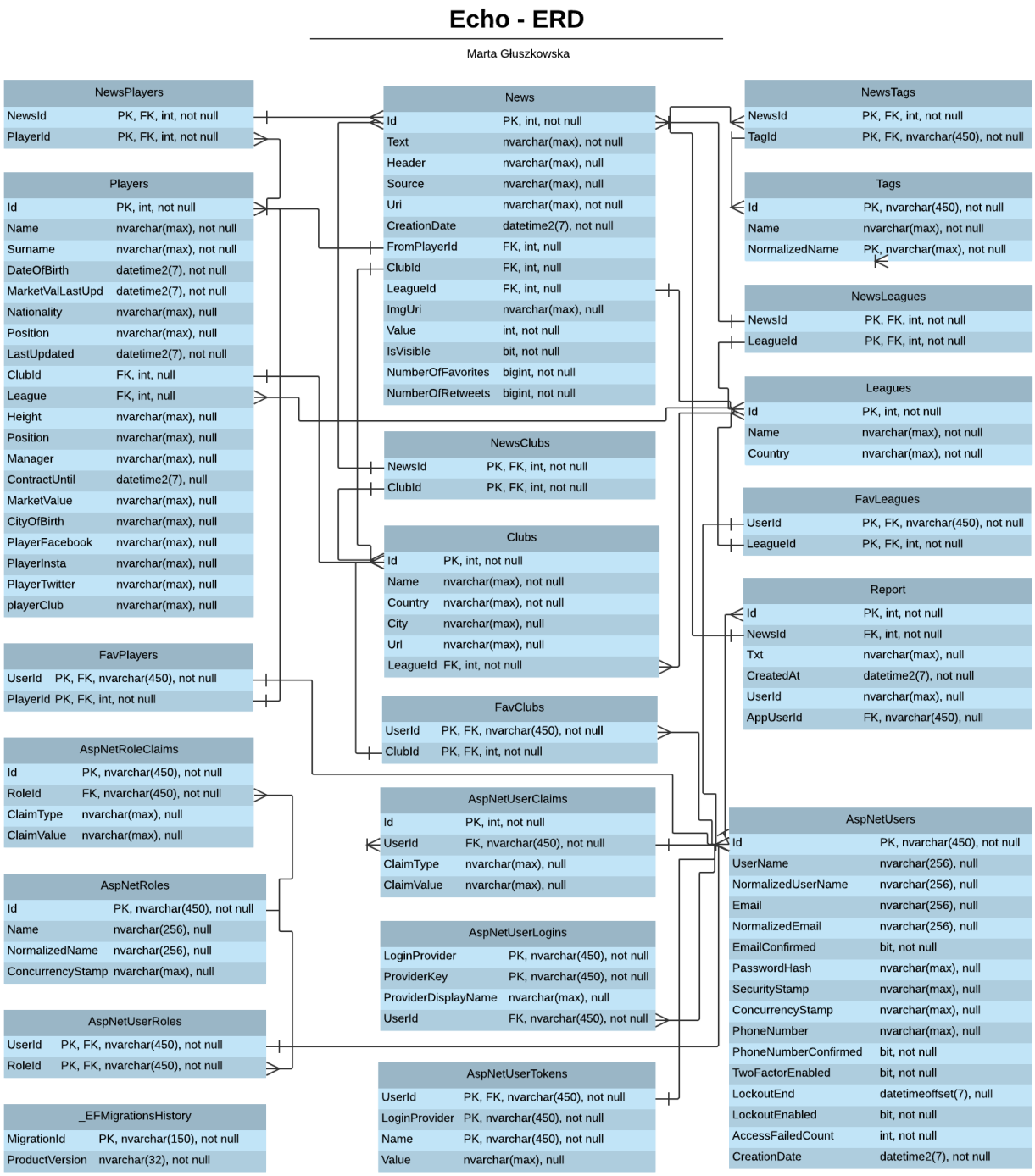
/api/Player	GET	Zwraca listę wszystkich zawodników z bazy w postaci:
		[
		{
		„playerId”: 0,
		“playerName”: “string”
		}
		]

/api/Player/{playerId}	GET	{playerId} to liczba odpowiadająca Id zawodnika w bazie danych, endpoint zwraca kod 200 i informacje szczegółowe na temat zawodnika znajdujące się w bazie danych aplikacji lub kod 404, jeżeli nie ma zawodnika o danym Id.
/api/Player/ AddFavourite/ {playerId}	POST	Dodanie zawodnika o podanym Id do listy ulubionych. Jeżeli zawodnika o podanym ID nie ma to zwraca 404, a jeżeli już jest w ulubionych to zwraca 400.
/api/Player/ DeleteFavourite/ {playerId}	DELETE	Usuwa zawodnika o podanym Id do listy ulubionych. Jeżeli zawodnika o podanym ID nie ma w bazie to zwraca 404, a jeżeli jest w bazie i nie ma w ulubionych to zwraca 400.
/api/Player/ FavouritePlayers	GET	Pobiera listę ulubionych zawodników.
/api/Player/ AddFromContractList	POST	{ “playerName”: “string”, “addToFavourite”: true

		} Dodaje zawodnika po pełnym imieniu i nazwisku do bazy danych, najpierw szukając w Transfermarkt. Po sukcesie dodania do bazy, zaczyna szukać newsów w trzech źródłach. Może to trwać około 30 sekund. Może jednocześnie zapisać zawodnika do listy ulubionych. Jeżeli zawodnik nie zostanie odnaleziony w Transfermarkt to zwróci 404. Jeżeli zawodnik już jest w bazie to zwróci 400.
/api/Player/ AddFromAddPlayer	POST	Dodanie zawodnika do bazy wg. modelu zwracanego przez /api/Communication/PlayerTM/{playerFullName} Gdy zawodnik już jest w bazie zwraca 400.

Relacyjna baza danych

Wszystkie dane aplikacji WhisperPress są przechowywane w relacyjnej bazie danych MSSQL. Poniższa grafika przedstawia diagram bazy danych:



Aplikacja do pobierania i analizowania danych

Autorzy: Marta Głuszkowska, Emilia Kłobukowska

Aplikacja napisana przy użyciu technologii Django. Służy do pobierania danych ze źródeł internetowych i nadawania im odpowiednich tagów. Dane te są przekazywane aplikacji serwerowej za pomocą serwisów REST.

Najważniejsze pliki

Opis najważniejszych plików znajdujących się w aplikacji:

/webapp/AllWords.txt

Plik zawierający listę wszystkich słów, jakie kiedykolwiek były analizowane wraz z liczbą ich wystąpień. Używany w tagowaniu emocjonalnym w pliku *views.py*.

/webapp/BannedWords.txt

Plik zawierający listę słów, które nie powinny zostać brane pod uwagę jako tagi. Używany w tagowaniu emocjonalnym w pliku *views.py*.

/webapp/SpecialWords.txt

Plik zawierający listę słów, które powinny mieć większe prawdopodobieństwo wystąpienia jako tagi ze względu na związek z piłką nożną. Używany w tagowaniu emocjonalnym w pliku *views.py*.

/webapp/views.py

W pliku views.py znajdują się wszystkie widoki udostępniające dane za pomocą zapytań http oraz algorytmy odpowiadające za pobieranie newsów z różnych źródeł internetowych i ich tagowanie.

changeDateFormat

Pozwala to na przekonwertowanie daty z formatu „14 lip 2020” na „2020-07-14”.
Input: *date* - string odzwierciedlający datę w formacie zawierającym co najmniej trzy pierwsze litery miesiąca (np. „14 lip 2020”).
Output: String zawierający przekonwertowaną datę (np. „2020-07-14”).

removeAccents

Wszystkie litery i symbole posiadające znaki diakratyczne zostają zamienione na litery symbole bez znaków diakratycznych
Input: input_text - string dowolnej długości (np. „Zażółć gęślą jaźń”)
Output: string nie zawierający znaków diakratycznych (np. „Zazolc gesla jaźń”)

search_Twitter

Wyszukuje wszystkie tweety (posty na portalu Twitter) dotyczące danej osoby, które są nowsze niż podana data. Następujące informacje zostają zapisane w formacie JSON

Input:

name – string składający się z co najmniej dwóch wyrazów będących imieniem i nazwiskiem (np. „Arkadiusz Milik").

lastCheckDateTime – data i godzina zapisane jako typ datetime

Output1: lista informacji na temat tweetów zapisana w formacie JSON:

{'News': [
{'CreationDate': 'data utworzenia tweeta',
'Text': 'tekst tweeta',
'Source': 'Twitter',
'Uri': 'odnośnik do oryginalnego tweeta',
'NumberOfRetweets': int,
'NumberOfFavourites': int,
'SentimentTag': int (wartość emocjonalna danego tekstu),
'Media': 'URL zdjęcia umieszczonego w tweecie'},
(…)]}

Funkcja oparta jest na zapytaniach API do portalu Twitter. Istnieje limit 180 zapytań, resetujący się co 15 minut. W przypadku, gdy pozostały limit wynosi 5 zapytań, zwracany jest czas pozostały do resetu limitu.

Output2: liczba oznaczająca pozostały czas do resetu limitu

W przypadku przekroczenia limitu zapytań, zwracany zostaje error 429.

Output3: *'error 429'*

search_Twitter_While_Limited

Odpowiada funkcji *search_Twitter*, posiada jednak inny zestaw kluczy i tokenów do autoryzacji połączenia z portalem Twitter. Może być wykorzystywana, gdy limit dla podstawowej funkcji został wyczerpany.

Input:

name – string składający się z co najmniej dwóch wyrazów będących imieniem i nazwiskiem

lastCheckDateTime – data i godzina zapisane jako typ datetime

Output1: lista informacji na temat tweetów zapisana w formacie JSON:

{'News': [
{'CreationDate': 'data utworzenia tweeta',
'Text': 'tekst tweeta',
'Source': 'Twitter',
'Uri': 'odnośnik do oryginalnego tweeta',
'NumberOfRetweets': int,
'NumberOfFavourites': int,
'SentimentTag': int (wartość emocjonalna danego tekstu),
'Media': 'URL zdjęcia umieszczonego w tweecie'},
(…)]}

Output2: słownik zawierający informacje o rodzaju błędu i pozostałym limicie
Output3: *'error 429'*

search_90minut

Wyszukuje wiadomości i wydarzenia ze strony głównej portalu 90minut i zapisuje informacje na temat artykułów dowolnej osoby, które zostały opublikowane później niż podana data. Zostają one zwracane w formacie JSON.

Input:

name – string składający się z co najmniej dwóch wyrazów będących imieniem i nazwiskiem

lastCheckDateTime – data i godzina zapisane jako typ datetime

Output: lista informacji na temat tweetów zapisana w formacie JSON:

{'News': [
{'Text': 'tekst tweeta',
'Header': 'nagłówek artykułu,
'Source': '90minut.pl',
'Uri': 'odnośnik do oryginalnego tweeta',
'CreationDate': 'data utworzenia tweeta'
'SentimentTag': int (wartość emocjonalna danego tekstu) },
(…)]}

search_Transfermarkt_profile

Wyszukuje profil dowolnej osoby na portalu Transfermarkt i zwraca jego dane profilowe w formacie JSON:

{'Players': [
{'Name': 'imię',
'Surname': 'nazwisko',
'DateOfBirth': 'data urodzenia',
'PlaceOfBirth': 'miejsce urodzenia',
'Nationality': 'narodowość',
'Position': 'pozycja, na której gra zawodnik',
'ContractUntil': 'data ważności obecnego kontraktu',
'Height: 'wzrost',
'Manager: 'menadżer,
'MarketValue': 'aktualna wartość rynkowa',
'SocialMedia': ['lista oficjalnych profili społecznościowych zawodnika']
'Club': 'klub, do którego obecnie należy zawodnik' }]}

Input: *name* – string składający się z co najmniej dwóch wyrazów będących imieniem i nazwiskiem

Output1: lista informacji profilowych zapisana w formacie JSON

W przypadku nie znalezienia profilu danego zawodnika zwracany jest komunikat
Output2: *'player not found'*

Transfermarkt_profile_from_URL

Pobiera informacje z profilu Transfermarkt, do którego został podany link. Zwraca listę informacji profilowych zapisanych w formacie JSON.
Input: *url* – URL do profilu Transfermarkt
Output: lista informacji profilowych zapisana w formacie JSON:

{'Players': [
{'Name': 'imię',
'Surname': 'nazwisko',
'DateOfBirth': 'data urodzenia',
'PlaceOfBirth': 'miejsce urodzenia',
'Nationality': 'narodowość',
'Position': 'pozycja, na której gra zawodnik',
'ContractUntil': 'data ważności obecnego kontraktu',
'Height: 'wzrost',
'Manager: 'menadżer,
'MarketValue': 'aktualna wartość rynkowa',
'SocialMedia': ['lista oficjalnych profili społecznościowych zawodnika']
'Club': 'klub, do którego obecnie należy zawodnik' }]}

search_Transfermarkt

Wyszukuje wiadomości i pogłoski transferowe z portalu Transfermarkt, które dotyczą dowolnej osoby i zostały opublikowane później niż podana data. Zwracana zostaje lista w formacie JSON, zawierająca informacje:
Input:
name – string składający się z co najmniej dwóch wyrazów będących imieniem i nazwiskiem
lastCheckDateTime – data i godzina zapisane jako typ datetime
Output: lista informacji profilowych zapisana w formacie JSON:
{'News': [
{'Header': 'nagłówek artykułu,
'Text': 'tekst tweeta',
'Source': 'Transfermarkt',
'Uri': 'odnośnik do oryginalnego artykułu',
'CreationDate': 'data opublikowania artykułu,
'SentimentTag': int (wartość emocjonalna danego tekstu) },
(…)]}

searchExpiringContracts

Wyszukuje na portalu Transfermarkt zawodników z kończącymi się kontraktami, które są zgodne z podanymi filtrami.

Input:

year – rok wygasania kontraktu,
playerPosition – pozycja ogólna, na której gra zawodnik,
playerPositionDetails – pozycja szczegółowa, na której gra zawodnik,
ageGroup – zakres wiekowy,
nationality – narodowość

Output: lista informacji o zawodnikach spełniających podane wymagania, zapisana w formacie JSON z danymi informacjami:

[{'name': 'imię i nazwisko',
'position': 'pozycja, na której gra zawodnik',
'currentClub': 'klub, do którego obecnie należy zawodnik',
'currentMarketValue': 'aktualna wartość rynkowa',
'playerProfileURL': 'URL profilu Transfermarkt zawodnika'},
(…)]

AddSentimentTag

Przygotowuje tekst do analizy wydźwięku, wywołuje funkcję *checkWordInDicts* oraz zwraca wynik analizy.

Input: *text* – dowolny tekst w języku polskim
Output1: 0 – wydźwięk negatywny
Output2: 1 – wydźwięk neutralny
Output3: 2 – wydźwięk pozytywny

checkWordInDicts

Sprawdza wartość wydźwięku wszystkich słów zawartych w liście *sentence* na podstawie danych zawartych w pliku *„zbior slow.csv".*

Input: *sentence* – lista zawierająca pojedyncze wyrazy
Output: liczba float oznaczająca sumę wartości wydźwięku wszystkich słów

getPlayerProfile

Wywołuje funkcję *search_Transfermarkt_profile* oraz zwraca jej wynik

Input: *name* – string składający się z co najmniej dwóch wyrazów będących imieniem i nazwiskiem

Output: lista w formacie JSON:

{'Players': [
{'Name': 'imię',
'Surname': 'nazwisko',
'DateOfBirth': 'data urodzenia',
'PlaceOfBirth': 'miejsce urodzenia',
'Nationality': 'narodowość',
'Position': 'pozycja, na której gra zawodnik',
'ContractUntil': 'data ważności obecnego kontraktu',
'Height: 'wzrost',

'Manager: 'menadżer,
'MarketValue': 'aktualna wartość rynkowa',
'SocialMedia': ['lista oficjalnych profili społecznościowych zawodnika']
'Club': 'klub, do którego obecnie należy zawodnik' }]}

getPlayerProfileFromURL

Wywołuje funkcję *search_Transfermarkt_profile_from_URL* oraz zwraca jej wynik
Input: *url* - URL do profilu Transfermarkt
Output: lista w formacie JSON:

{'Players': [
{'Name': 'imię',
'Surname': 'nazwisko',
'DateOfBirth': 'data urodzenia',
'PlaceOfBirth': 'miejsce urodzenia',
'Nationality': 'narodowość',
'Position': 'pozycja, na której gra zawodnik',
'ContractUntil': 'data ważności obecnego kontraktu',
'Height: 'wzrost',
'Manager: 'menadżer,
'MarketValue': 'aktualna wartość rynkowa',
'SocialMedia': ['lista oficjalnych profili społecznościowych zawodnika']
'Club': 'klub, do którego obecnie należy zawodnik' }]}

getAllInfo

Wywołuje funkcje *search_Twitter*, *search_90minut* i *search_Transfermarkt* oraz zwraca wszystkie wyniki jako jeden plik JSON.
Input:

name - string składający się z co najmniej dwóch wyrazów będących imieniem i nazwiskiem
date - data i godzina zapisane jako string

Output: lista w formacie JSON:

{'Dane': [
{'News': [
{'CreationDate': 'data utworzenia tweeta',
'Text': 'tekst tweeta',
'Source': 'Twitter',
'Uri': 'odnośnik do oryginalnego tweeta',
'NumberOfRetweets': int,
'NumberOfFavourites': int,
'SentimentTag': int (wartość emocjonalna danego tekstu),
'Media': 'URL zdjęcia umieszczonego w tweecie'},
(…)]}
{'News': [

{'Text': 'tekst tweeta',
'Header': 'nagłówek artykułu,
'Source': '90minut.pl',
'Uri': 'odnośnik do oryginalnego tweeta',
'CreationDate': 'data utworzenia tweeta'
'SentimentTag': int (wartość emocjonalna danego tekstu) },
(…)]},
{'News': [
{'Header': 'nagłówek artykułu,
'Text': 'tekst tweeta',
'Source': 'Transfermarkt',
'Uri': 'odnośnik do oryginalnego artykułu',
'CreationDate': 'data opublikowania artykułu,
'SentimentTag': int (wartość emocjonalna danego tekstu) },
(…)]}]}

getTwitterNews

Wywołuje funkcję *search_Twitter* i zwraca wynik jako plik JSON.

Input:

name - string składający się z co najmniej dwóch wyrazów będących imieniem i nazwiskiem

date - data i godzina zapisane jako string

Output: lista w formacie JSON:

{'News': [
{'CreationDate': 'data utworzenia tweeta',
'Text': 'tekst tweeta',
'Source': 'Twitter',
'Uri': 'odnośnik do oryginalnego tweeta',
'NumberOfRetweets': int,
'NumberOfFavourites': int,
'SentimentTag': int (wartość emocjonalna danego tekstu),
'Media': 'URL zdjęcia umieszczonego w tweecie'},
(…)]}

getTwitterNewsWhileLimited

Wywołuje funkcję *search_Twitter_While_Limited* i zwraca wynik jako plik JSON.

Input:

name - string składający się z co najmniej dwóch wyrazów będących imieniem i nazwiskiem

date - data i godzina zapisane jako string

Output: lista w formacie JSON:

{'News': [
{'CreationDate': 'data utworzenia tweeta',

'Text': 'tekst tweeta',
'Source': 'Twitter',
'Uri': 'odnośnik do oryginalnego tweeta',
'NumberOfRetweets': int,
'NumberOfFavourites': int,
'SentimentTag': int (wartość emocjonalna danego tekstu),
'Media': 'URL zdjęcia umieszczonego w tweecie'},
(…)]}

getTransfermarktNews

Wywołuje funkcję *search_Transfermarkt* i zwraca wynik jako plik JSON.

Input:

name - string składający się z co najmniej dwóch wyrazów będących imieniem i nazwiskiem

date - data i godzina zapisane jako string

Output: lista w formacie JSON:

{'News': [
{'Header': 'nagłówek artykułu,
'Text': 'tekst tweeta',
'Source': 'Transfermarkt',
'Uri': 'odnośnik do oryginalnego artykułu',
'CreationDate': 'data opublikowania artykułu,
'SentimentTag': int (wartość emocjonalna danego tekstu) },
(…)]}

get90MinutNews

Wywołuje funkcję *search_90minut* i zwraca wynik jako plik JSON.

Input:

name - string składający się z co najmniej dwóch wyrazów będących imieniem i nazwiskiem

date - data i godzina zapisane jako string

Output: lista w formacie JSON:

{'News': [
{'Text': 'tekst tweeta',
'Header': 'nagłówek artykułu,
'Source': '90minut.pl',
'Uri': 'odnośnik do oryginalnego tweeta',
'CreationDate': 'data utworzenia tweeta'
'SentimentTag': int (wartość emocjonalna danego tekstu) },
(…)]}

addSentimentTagForMultipleTexts

Dla każdego tekstu z listy *teksts* wywołuje funkcję *AddSentimentTag* i zwraca listę par złożonych z tekstu i jego wartości wydźwięku

Input: *texts* – lista różnych tekstów

Output: lista złożona z par (*text*, *sentimentValue*), gdzie *sentimentValue* to wartość wydźwięku tekstu:

[(text1, sentimentValue1), (tet2, sentimentValue2), (…)]

extract_names1

Jedna z dwóch funkcji odpowiadających za znajdowanie w tekście nazw własnych oraz imion i nazwisk. Na wejściu funkcja przyjmuje tekst w formacie typu string, a zwraca listę znalezionych nazw własnych. Korzysta z biblioteki NLTK. Funkcja *extract_names1* wykorzystywana jest w funkcji *add_tags* do znajdowania nazw własnych w tekście i dodawania ich do listy tagów.

extract_names2

Druga z funkcji odpowiadających za znajdowanie w tekście imion i nazwisk. Korzysta z nieco innych metod wyszukiwania imion i nazwisk i jest uzupełnieniem funkcji *extract_names1*. Na wejściu funkcja przyjmuje tekst w formacie typu string, a zwraca listę znalezionych nazw własnych. Korzysta z biblioteki NLTK. Funkcja *extract_names2* wykorzystywana jest w funkcji add_tags do znajdowania nazw własnych w tekście i dodawania ich do listy tagów.

extract_hashtags

Funkcja służąca do znajdowania w tekście tagów poprzedzonych znakiem „#”. Na wejściu przyjmuje tekst typu string i zwraca listę wszystkich hashtagów znalezionych w danym tekście. Wykorzystywana jest w funkcji *add_tags*.

remove_punctuation_marks

Funkcja usuwająca z tekstu interpunkcję i dzieląca go na listę wyrazów. Na wejściu przyjmuje tekst typu string i zwraca listę wszystkich słów z danego tekstu. Wykorzystywana jest w funkcjach *prepare* oraz *AddSentimentTag* w celu przygotowania tekstów do tagowania.

remove_polish_stop_words

Funkcja usuwająca z listy słów tzw. „stop words”, czyli słowa występujące bardzo często w tekstach w języku polskim, które nie powinny być brane pod uwagę jako potencjalni kandydaci na tagi dla danego tekstu. Na wejściu przyjmuje listę wszystkich słów w tekście i zwraca przefiltrowaną listę słów. Wykorzystywana jest w funkcjach *prepare* oraz *AddSentimentTag* w celu przygotowania tekstów do tagowania.

remove_numbers

Funkcja usuwająca z listy słów wszystkie liczby. Na wejściu przyjmuje listę wszystkich słów w tekście i zwraca przefiltrowaną listę słów. Wykorzystywana jest w funkcjach *prepare* oraz *AddSentimentTag* w celu przygotowania tekstów do tagowania.

computeTF

Funkcja przygotowująca dla danego zestawu słów częstotliwość występowania danego słowa w konkretnym tekście. Jest to więc informacja, jaka jest liczba wystąpień każdego słowa w danym tekście w stosunku do liczby wszystkich słów w danym tekście. Na wejściu funkcja przyjmuje słownik, w którym kluczami są poszczególne słowa z danego tekstu, a wartościami liczby całkowite oznaczające liczbę wystąpień danego słowa w tekście. Drugim argumentem funkcji jest lista wszystkich słów występujących w danym tekście. Funkcja zwraca słownik, gdzie kluczami są pojedyncze słowa, a wartościami liczba oznaczająca stosunek liczby wystąpień słowa do liczby wszystkich słów w tekście.

computeIDF

Funkcja obliczająca, jak często dane słowo ze zbioru słów podanych na wejściu występuje w stosunku do wszystkich tekstów. Na wejściu podany jest słownik, gdzie kluczami są poszczególne teksty, a wartościami słowniki zawierające wszystkie słowa z danego tekstu i liczbę ich wystąpień.

Funkcja pobiera z pliku *AllWords.txt* słownik zawierający wszystkie słowa, które kiedykolwiek pojawiły się w analizowanych tekstach wraz z liczbą ich wystąpień. Następnie tworzony jest pusty słownik *idfDict*, w którym przechowywane będą wyniki i kluczami w tym słowniku są wszystkie słowa, jakie wystąpiły w wejściowym słowniku.

Następnie dla każdego dokumentu w wejściowym słowniku i dla każdego słowa w tym dokumencie sprawdzane jest, czy słowo pojawiło się już na liście wszystkich słów zapisanych w pliku *AllWords.txt*. Jeśli nie, słowo zostaje dodane do listy. Następnie liczba wystąpień tego słowa jest zwiększana o jeden i zapisana w pliku *AllWords.txt*.

Na końcu dla każdego słowa, które pojawiło się na wejściu, obliczana jest wartość IDF, czyli odwrotna częstotliwość słowa w stosunku do wszystkich dokumentów. Jest obliczana na podstawie wzoru: *log(N/v),* gdzie *N* to liczba wszystkich dokumentów, a *v* to liczba mówiąca, w ilu dokumentach wystąpiło dane słowo. Wynik jest zapisywany dla każdego słowa w słowniku *idfDict* i na końcu zwracany jest ten wynikowy słownik.

computeTFIDF

Funkcja obliczająca dla każdego słowa wartość TF-IDF, czyli iloczyn TF i IDF. Na wejściu przyjmuje słownik ze wszystkimi słowami występującymi w tekście i ich wartością TF, oraz zmienną oznaczającą wartość IDF dla wszystkich słów w tekście. Wartości te są mnożone i zapisywane do słownika, a później zwracane jako wynik.

lemat_from_word

Funkcja przyjmująca jako argument pojedyncze słowo i zwracająca jego lemat. Korzysta z analizatora morfologicznego Morfeusz. Wykorzystywana w funkcjach *prepare* oraz *prepare_word* do przygotowywania lematu dla słów występujących w pobranych tekstach.

prepare

Funkcja przygotowująca dany tekst do analizy tagowania. Na wejściu przyjmuje tekst typu string i zwraca listę wszystkich słów przygotowanych do analizy. Funkcja ta ma za zadanie zamienić wszystkie wielkie litery na małe, usunąć linki oraz oznaczenia kont

społecznościowych rozpoczynających się od znaku „@”. Później przy użyciu funkcji *remove_punctuation_marks*, *lemat_from_word* oraz *remove_numbers* przygotowuje odpowiednią listę słów. Wykorzystywana w funcji *tag* do przygotowywania tekstów do tagowania po słowach kluczowych.

prepare_word

Funkcja przygotowująca odpowiednio słowo, które może zostać wyznaczone jako tag. Na wejściu przyjmuje pojedyncze słowo, a na wyjściu zwraca to samo słowo w odpowiedniej formie lub wartość *„delete word”*, jeśli słowo nie powinno znaleźć się na liście tagów.

Funkcja zamienia wielkie litery w słowie na małe i analizuje słowo za pomocą funkcji *analyse* z modułu Morfefusz. Jeśli w analizie danego słowa wystąpił znacznik *„ign”, „imię”* lub *„nazwisko*”, wówczas prygotowywany jest lemat danego słowa i słowo jest zwracane na wyjściu. Pozostałe słowa nie są zwracane, a zamiast nich zwracana jest wartość *„delete word”*.

tag

Funkcja wyznaczająca dla zbioru tekstów listę słów kluczowych, które potencjalnie mogą zostać tagami dla danych tekstów. Na wejściu przyjmuje listę tekstów w postaci pliku JSON, string reprezentujący imię i nazwisko zawodnika, którego dotyczą dane teksty, oraz zmienną logiczną *all* mówiącą o tym, czy zbiór newsów pochodzi ze wszystkich źródeł zebranych razem, czy z pojedynczego źródła. Domyślnie flaga *all* jest ustawiona na wartość False. Wówczas funkcja działa dla wejściowej zmiennej typu JSON o formacie:

```
{
    „News”: {
        „Text”: „Przykładowy tekst”,
        „Uri”: „Przykładowy uri”,
        …
    }
}
```

Jeśli flaga *all* jest równa True, wówczas algorytm działa dla wejściowego pliku JSON o formacie:

```
{
    „Dane”: {
        „News”: {
            „Text”: „Przykładowy tekst”,
            „Uri”: „Przykładowy uri”,
            …
        },
        „News”: {
            „Text”: „Przykładowy tekst”,
            „Uri”: „Przykładowy uri”,
            …
        },
        …
```

```
        }
    }
```

Funkcja pobiera z pliku *AllWords.txt* słownik z listą wszystkich słów, jakie kiedykolwiek pojawiły się w analizowanych tekstach wraz z informacją, ile razy wystąpiły. Lista tych słów, czyli kluczy słownika, jest zapisywana do zmiennej *uniqueWords*. Później dla każdego tekstu pobranego na wejściu jest przygotowywana lista słów, które będą rozpatrywane jako tagi. Lista ta jest przygotowywana za pomocą funkcji *prepare* i zapisana w słowniku *tagging_results*. Słowa te są również dodawane do listy wszystkich unikatowych słów w zmiennej *uniqueWords*.

Następnie dla każdego dokumentu tworzony jest słownik *numOfWords_bag*, gdzie dla każdego słowa obliczana jest liczba wystąpień słowa w danym dokumencie. Każdy taki słownik dla pojedynczego dokumentu jest dodawany do słownika *numOfWordsAll*, który przechowuje informację o wszystkich słowach ze wszystkich tekstów pobranych na wejściu. Na końcu dla każdego tekstu wywoływana jest funkcja *computeTF* z parametrami *numOfWords_bag* oraz *bagOfWords*. Zwraca ona słownik, gdzie kluczami są słowa występujące w danym tekście (odpowiadające kluczom w słowniku *numOfWords_bag*), a wartościami są liczby oznaczające stosunek liczby wystąpień słowa w danym tekście do liczby wszystkich słów z danego tekstu. Do obliczania „TF" brane są więc pod uwagę tylko słowa występujące w jednym konkretnym tekście. Słownik ten jest później zapisywany dla każdego dokumentu do słownika *tagging_results*.

Następnie algorytm sprawdza, czy zmienna *numOfWordsAll* jest pustą listą, co oznacza, że wejściowy plik JSON nie zawierał żadnych tekstów. Wówczas zwracana jest wartość False oraz zmienna *name*.

Kolejnym krokiem jest obliczenie „IDF" dla wszystkich słów, które pojawiły się w wejściowych tekstach. Obliczane jest to za pomocą funkcji *computeIDF*, której jako parametr przekazujemy słownik ze wszystkimi słowami występującymi w wejściowych tekstach wraz z liczbą ich wystąpień. Wynik tej funkcji zapisywany jest w zmiennej *idf*.

Następnie z plików *SpecialWords.txt* zapisywana jest lista słów, które powinny mieć zwiększoną wartość TD-IDF, aby miały one większą szansę na wystąpienie jako tagi. Są to słowa kluczowe dla piłki nożnej. Tak samo pobierana jest lista słów, które nigdy nie powinny pojawić się jako tagi. Lista ta jest pobierana z pliku *BannedWords.txt*.

Kolejnym elementem algorytmu jest obliczenie dla każdego dokumentu wejściowego wartości TF-IDF i ewentualne podwyższenie jej lub zaniżenie dla słów występujących na liście słów specjalnych i słów zakazanych. Wartość TF-IDF jest obliczana dla każdego dokumentu za pomocą funkcji *computeTFIDF*. Jej wynik jest zapisywany dla każdego tekstu w słowniku *tagging_results*. Jeśli jakieś wejściowe słowo należy do zbioru słów specjalnych, wartość TF-IDF jest dla niego mnożona razy 10, natomiast jeśli słowo należy do słów zakazanych, wówczas jego wartość jest ustawiana na 0. Dla każdego dokumentu słowa są sortowane malejąco, zaczynając od słowa o najwyższej wartości TF-IDF. Wynik algorytmu przechowywany w zmiennej *tagging_results* wraz ze zmienną *name* jest zwracany jako wynik funkcji.

add_tags

Funkcja przypisująca listę tagów dla każdego tekstu. Wynik jest zwracany jako zmienna typu JSON.

Na wejściu funkcja przyjmuje wynik funkcji *tag*, czyli słownik ze wszystkimi pobranymi tekstami, wszystkimi słowami w nich zawartymi, oraz wartością TF-IDF obliczoną dla każdego z tych słów. Drugim parametrem funkcji jest zmienna typu string reprezentująca imię i nazwisko szukanego zawodnika. Trzecim argumentem funkcji jest zmienna typu JSON zawierająca wszystkie pobrane teksty. Ostatni argument to flaga wskazująca, czy wejściowy zbiór newsów pochodzi ze wszystkich źródeł zebranych razem, czy z pojedynczego źródła. Flaga ta działa analogicznie do flagi *all* w funkcji *tag.*

Algorytm jest wykonywany po kolei dla każdego tekstu z wejściowego słownika *tagging_results*.

Najpierw spośród wszystkich słów z tekstu wybierane jest pierwsze 5, które otrzymały najwyższą wartość w algorytmie TF-IDF i dla których ta wartość była większa od 0. Jeśli wszystkich słów było mniej niż 5, wówczas brane są wszystkie słowa, których wartość TF-IDF była większa od 0. Następnie dla każdego tekstu wywoływane są dwie funkcje wyszukujące imiona, nazwiska i nazwy własne w tekstach: *extract_names1* i *extract_names2.* Tak samo wyszukiwane są wszystkie hashtagi z tekstu za pomocą funkcji *extract_hashtags*. Wszystkie znalezione wartości są odpowiednio przygotowywane za pomocą funkcji *prepare_word* i sprawdzane jest, czy słowo nie należy do listy polskich „stop words" oraz czy nie pojawiło się już wcześniej na liście tagów. Jeśli wszystkie warunki zostały spełnione, słowo zostaje dodane do listy tagów.

Do ostatecznej listy tagów zostaje dodane również imię i nazwisko szukanego zawodnika przekazane w parametrze *player*. Wszystkie wielkie litery w tagach zamieniane są na małe litery. Ostateczne listy tagów dla każdego tekstu są dodawane do zmiennej typu JSON, która została przekazana na wejściu i zmienna ta jest zwracana na wyjściu.

searchExpiringContractsView

Widok wywołujący funkcję *searchExpiringContracts* i udostępniający wynik w postaci pliku JSON za pomocą metody *HttpResponse*. Do wywołania funkcji *searchExpiringContracts* wykorzystuje parametry przekazane w żądaniu HTTP: *year*, *playerPosition*, *playerPositionDetails*, *ageGroup* oraz *nationality*. Jeśli któryś z parametrów nie został podany w żądaniu, domyślnie jest ustawiana wartość „*alle*", odpowiadająca w portalu Transfermarkt opcji „wszystkie". Jeśli funkcja *searchExpiringContracts* nie mogła się wykonać poprawnie, zwracany jest odpowiedni komunikat o błędzie. Jeśli cały widok nie wykonał się poprawnie, zwracany jest status 404 wraz z komunikatem „Page not found".

playerInfoView

Widok wywołujący funkcję *getPlayerProfile* i udostępniający wynik w postaci pliku JSON za pomocą metody *HttpResponse*. Do wywołania funkcji *getPlayerProfile* wykorzystuje parametr *name* przekazany w żądaniu http. Jeśli widok nie wykonał się poprawnie, zwracany jest status 404 wraz z komunikatem „Page not found".

allNewsForPlayerView

Widok wywołujący funkcję *getAllInfo*, która pobiera newsy ze wszystkich źródeł, oraz funkcje *tag* i *add_tags*, które dodają do każdego newsa odpowiednie tagi. Widok udostępnia wynik w postaci pliku JSON za pomocą metody *HttpResponse*. Do wywołania funkcji *getAllInfo* wykorzystuje parametry przekazane w żądaniu HTTP: *name*, *date*. Rezultat funkcji *getAllInfo* jest przekazywany do funkcji *tag*, która ma za zadanie przygotować potencjalną listę tagów dla tekstu. Jeśli funkcja *tag* nie mogła się wykonać poprawnie, zwracana jest jedynie lista pobranych newsów bez tagów. Jeśli funkcja *tag* zwróciła poprawnie potencjalną listę tagów, jej rezultat jest przekazywany do funkcji *add_tags*, która ma za zadanie przydzielić dla danego tekstu odpowiednie tagi z proponowanej listy. Jeśli cały widok nie wykonał się poprawnie, zwracany jest status 404 wraz z komunikatem „Page not found".

twitterView

Widok wywołujący funkcję *getTwitterNews*, która pobiera newsy portalu Twitter, oraz funkcje *tag* i *add_tags*, które dodają do każdego newsa odpowiednie tagi. Widok udostępnia wynik w postaci pliku JSON za pomocą metody *HttpResponse*. Do wywołania funkcji *getTwitterNews* wykorzystuje parametry przekazane w żądaniu HTTP: *name*, *date*. Jeśli funkcja *getTwitterNews* nie mogła się wykonać poprawnie, zwracany jest status 400. Jeśli funkcja *getTwitterNews* nie mogła się wykonać ze względu na wyczerpany limit zapytań do API Twittera, zwracana jest liczba całkowita oznaczająca timestamp zakończenia tego limitu. Jeśli funkcja *getTwitterNews* zwróciła niepustą listę newsów, rezultat jest przekazywany do funkcji *tag*, która ma za zadanie przygotować potencjalną listę tagów dla tekstu. Jej wynik jest przekazywany do funkcji *add_tags*, która ma za zadanie przydzielić dla danego tekstu odpowiednie tagi z proponowanej listy. Jeśli cały widok nie wykonał się poprawnie, zwracany jest status 404 wraz z komunikatem „Page not found".

newsFrom90minutView

Widok wywołujący funkcję *get90MinutNews*, która pobiera newsy portalu 90minut.pl, oraz funkcje *tag* i *add_tags,* które dodają do każdego newsa odpowiednie tagi. Widok udostępnia wynik w postaci pliku JSON za pomocą metody *HttpResponse*. Do wywołania funkcji *get90MinutNews* wykorzystuje parametry przekazane w żądaniu HTTP: *name*, *date*. Jeśli funkcja *get90MinutNews* zwróciła niepustą listę newsów, rezultat jest przekazywany do funkcji *tag*, która ma za zadanie przygotować potencjalną listę tagów dla tekstu. Jej wynik jest przekazywany do funkcji *add_tags,* która ma za zadanie przydzielić dla danego tekstu odpowiednie tagi z proponowanej listy. Jeśli cały widok nie wykonał się poprawnie, zwracany jest status 404 wraz z komunikatem „Page not found".

transfermarktView

Widok wywołujący funkcję *getTransfermarktNews*, która pobiera newsy portalu Transfermarkt, oraz funkcje *tag* i *add_tags*, które dodają do każdego newsa odpowiednie tagi. Widok udostępnia wynik w postaci pliku JSON za pomocą metody *HttpResponse*. Do wywołania funkcji *getTransfermarktNews* wykorzystuje parametry przekazane w żądaniu HTTP: *name*, *date*. Jeśli funkcja *getTransfermarktNews* zwróciła niepustą listę newsów, rezultat jest przekazywany do funkcji *tag*, która ma za zadanie przygotować potencjalną

listę tagów dla tekstu. Jej wynik jest przekazywany do funkcji *add_tags*, która ma za zadanie przydzielić dla danego tekstu odpowiednie tagi z proponowanej listy. Jeśli cały widok nie wykonał się poprawnie, zwracany jest status 404 wraz z komunikatem „Page not found".

transfermarktProfileFromUrlView

Widok wywołujący funkcję *getPlayerProfileFromURL*. Udostępnia wynik w postaci pliku JSON za pomocą metody HttpResponse. Do wywołania funkcji *getPlayerProfileFromURL* wykorzystuje parametr url przekazany w żądaniu HTTP. Jeśli widok nie wykonał się poprawnie, zwracany jest status 404 wraz z komunikatem „Page not found".

tagNews

Widok wywołujący funkcje *tag* i *add_tags*, które dodają odpowiednie tagi do pojedynczego tekstu. Widok udostępnia wynik w postaci pliku JSON za pomocą metody *HttpResponse*. Do wywołania funkcji *tag* wykorzystuje parametry przekazane w żądaniu HTTP: *text*, *name*. Jej wynik jest przekazywany do funkcji *add_tags*, która ma za zadanie przydzielić dla danego tekstu odpowiednie tagi z proponowanej listy. Jeśli cały widok nie wykonał się poprawnie, zwracany jest status 404 wraz z komunikatem „Page not found".

twitterLimitedView

Widok działający analogicznie do widoku *twitterView*, z tą różnicą, że zamiast funkcji *gettTwitterNews* wykorzystuje funkcję *getTwitterNewsWhileLimited* i jest używany w sytuacji, kiedy widok *twitterView* nie może działać poprawnie ze względu na wyczerpane limity zapytań.

sentimentTagNews

Widok wywołujący funkcję *AddSentimentTag*, która nadaje danemu tekstowi tag emocjonalny. Udostępnia wynik w postaci liczby całkowitej za pomocą metody *HttpResponse*. Do wywołania funkcji *AddSentimentTag* wykorzystuje parametr *text* przekazany w żądaniu HTTP. Jeśli widok nie wykonał się poprawnie, zwracany jest status 404 wraz z komunikatem „Page not found".

/webapp/zbior slow.csv

Plik ze zbiorem wszystkich słów, wykorzystywany w pliku *views.py* do tagowania emocjonalnego.

/webapp/access_keys_and_tokens.txt

Plik zawierający tokeny do API Twittera, używane do pobierania danych z Twittera w pliku views.py

/webapp/access_keys_and_tokens_while_limited.txt

Plik zawierający drugi zestaw tokenów do API Twittera, używany do pobierania danych z Twittera w pliku views.py

Endpointy

Poniższa tabela przedstawia wszystkie endpointy udostępnione przez aplikację wraz z ich opisem:

ENDPOINT	PARAMETRY	OPIS
/expiringContracts/	year, playerPosition, playerPositionDetails, ageGroup, nationality	Zwraca listę zawodników z kończącymi się kontraktami
/player/	name	Zwraca informacje o zawodniku
/allNews/	name, date	Pobiera newsy z wszystkich źródeł dla danego zawodnika, począwszy od określonej daty
/twitter/	name, date	Pobiera newsy z portalu Twitter dla danego zawodnika, począwszy od określonej daty
/twitterLimited/	name, date	Pobiera newsy z portalu Twitter dla danego zawodnika, począwszy od określonej daty
/90minut/	name, date	Pobiera newsy z portalu 90minut dla danego zawodnika, począwszy od określonej daty
/transfermarkt/	name, date	Pobiera newsy z portalu Transfermarkt dla danego zawodnika, począwszy od określonej daty
/transfermarktProfileFromUrl/	url	Pobiera informacje o zawodniku na podstawie adresu URL jego profilu na portalu Transfermarkt
/tagNews/	text, name	Zwraca listę tagów dla podanego tekstu i nazwiska zawodnika
/sentimentTag/	text	Zwraca tag emocjonalny dla danego tekstu

Wszystkie endpointy korzystają z metody GET.