tweepy Documentation

Wydanie 3.9.0

Joshua Roesslein

16 lip 2020

Spis treści

Zawartość:

ROZDZIAŁ 1

Instalacja

Użycie pip jest najprostszym sposobem na instalację najnowszej wersji z PyPI:

```
pip install tweepy
```

Możesz także użyć Git do zklonowania repozytorium z GithHub i zainstalować najnowszą wersję deweloperską:

```
git clone https://github.com/tweepy/tweepy.git
cd tweepy
pip install .
```

Możesz także zainstalować prosto z repozytorium GitHub:

```
pip install git+https://github.com/tweepy/tweepy.git
```

ROZDZIAŁ 2

Pierwsze kroki

2.1 Wprowadzenie

Jeżeli jesteś nowy to tutaj jest idealne miejsce do rozpoczęcia twojej przygody z Tweepy. Celem tego poradnika jest pomoc w ustawieniu i rozpoczęciu pracy z Tweepy. Nie znajdziesz tu zbyt szczegółowych opisów, głównie podstawowe informacje.

2.2 Hello Tweepy

```
import tweepy

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

api = tweepy.API(auth)

public_tweets = api.home_timeline()
for tweet in public_tweets:
    print(tweet.text)
```

Ten przykład pobierze tweety z twojej osi czasu i wyświetli tekst każdego z nich w konsoli. Twitter wymaga by wszystkie żądania używały OAuth do uwierzytelniania. W *Poradnik uwierzytelniania* znajdziesz więcej szczegółów na temat uwierzytelniania.

2.3 API

Klasa API dostarcza dostęp do całości metod twitter RESTful API. Każda metoda może zaakceptować różne parametry i zwrócić odpowiedzi. Po więcej informacji zajrzyj do *API Reference*.

2.4 Modele

When we invoke an API method most of the time returned back to us will be a Tweepy model class instance. This will contain the data returned from Twitter which we can then use inside our application. For example the following code returns to us a User model:

```
# Get the User object for twitter...
user = api.get_user('twitter')
```

Modele zawierają dane i metody pomocnicze, których możesz później użyć:

```
print(user.screen_name)
print(user.followers_count)
for friend in user.friends():
   print(friend.screen_name)
```

Po więcej informacji zajrzyj do ModelsReference.

ROZDZIAŁ 3

Poradnik uwierzytelniania

3.1 Wprowadzenie

Tweepy wpspiera obydwa sposoby uwierzytelniania - OAuth 1a (aplikacja-użytkownik) oraz OAuth 2 (tylko-aplikacja). Uwierzytelnianie jest obsługiwanie poprzez klasę tweepy.AuthHandler.

3.2 Uwierzytelnianie OAuth1a

Tweepy stara się ułatwić tobie korzystanie z OAuth 1a. By rozpocząć proces uwierzytelniania musisz zarejestrować swoją rejestrację klienta na Twitterze. Stwórz nową rejestrację a gdy to zrobisz powinieneś posiadać swój klucz konsumenta i sekret. Nie strać ich, będą ci potrzebne.

Następny krok to stworzenie instancji OAuthHandler. Do instancji tej przekaż swój klucz konsumenta oraz sekret, które zostały ci podane w poprzednim kroku:

```
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
```

Jeżeli posiadasz aplikacje sieciową i używasz wywołania zwrotnego URL które musi być dostarczone dynamicznie - musisz je także przekazać:

```
auth = tweepy.OAuthHandler(consumer_key, consumer_secret,
callback_url)
```

Jeżeli wywołanie zwrotne URL nie będzie zmieniane to najlepiej jest skonfigurować je statycznie na twitter.com gdy ustawiasz swój profil rejestracyjny.

W przeciwieństwie do podstawowego uwierzytelniania musisz wykonać „taniec” OAuth 1a zanim będziesz mógł zacząć używać API. By to zrobić musisz wykonać następujące kroki:

1. Zdobądź token żądania od Twittera
2. Przekieruj użytkownika do twitter.com by uwierzytelnić swoją rejestrację

3. Jeżeli używasz wywołania zwrotnego to Twitter przekieruje użytkownika do ciebie. W innym wypadku użytkownik musi ręcznie dostarczyć ci kod weryfikacyjny.
4. Wymień token uwierzytelniania na token dostępu.

Pozyskaj token żądania by rozpocząć taniec:

```
try:
    redirect_url = auth.get_authorization_url()
except tweepy.TweepError:
    print('Error! Failed to get request token.')
```

To wywołanie żąda token od twittera i zwraca tobie zuwierzytelniony URL, w którym użytkownik musi być przekierowany by być zuwierzytelnionym. Jeżeli dzieje się to w aplikacji komputerowej to możesz trzymać się swojej instancji OAuthHandler póki nie wróci użytkownik. W aplikacji sieciowej używane będzie żądanie wywołania zwrotnego. Dlatego też musisz składować w sesji token żądania, gdyż będzie on potrzebny w środku żadania wywołania zwrotnego URL. Poniżej znajduje się przykład składowania tokenu żądania w sesji:

```
session.set('request_token', auth.request_token['oauth_token'])
```

Następnie możesz przekierować użytkownika do URL, który został ci zwrócony z metody get_authorization_url().

Jeżeli jest to aplikacja komputerowa (lub każda inna aplikacja używająca wywołania zwrotnego) to konieczne jest zapytanie użytkownika o „kod weryfikacji", który twitter dostarczy mu gdy zostaniesz zuwierzytelniony. W aplikacji sieciowej wartość weryfikacyjna będzie dostarczona w żądaniu wywołania zwrotnego od twittera, w formie parametru zapytania GET w URL.

```
# Example using callback (web app)
verifier = request.GET.get('oauth_verifier')

# Example w/o callback (desktop)
verifier = raw_input('Verifier:')
```

Ostatnim krokiem jest wymiana tokenu żądania na token dostępu. Token dostępu jest „kluczem" otwierającym skarbiec Twitter API. By pozyskać ten token musisz poczynić następujące kroki:

```
# Let's say this is a web app, so we need to re-build the auth handler
# first...
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
token = session.get('request_token')
session.delete('request_token')
auth.request_token = { 'oauth_token' : token,
                       'oauth_token_secret' : verifier }

try:
    auth.get_access_token(verifier)
except tweepy.TweepError:
    print('Error! Failed to get access token.')
```

Warto jest zachować token dostępu na przyszłość. Nie musisz pozyskiwać go na nowo za każdym razem. Na tę chwilę Twitter nie wygasza ważności tokenów, tak więc stają się one nieważne tylko wtedy gdy użytkownik wycofa dostęp dla twojej aplikacji. Składowanie tokenu dostępu zależne jest od twojej aplikacji. W skrócie - musisz składować dwie wartości jako ciąg znaków: klucz i sekret:

```
auth.access_token
auth.access_token_secret
```

Możesz wrzucić je do bazy danych, pliku lub gdziekolwiek składujesz dane. By ponownie zbudować OAuthHandler z zapisanych tokenów, musisz wykonać następujące kroki:

```
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(key, secret)
```

Twój OAuthHandler jest teraz wyposażony w token dostępu - możesz zacząć pracę:

```
api = tweepy.API(auth)
api.update_status('tweepy + oauth!')
```

3.3 Uwierzytelnianie OAuth 2

Tweepy wspiera także uwierzytelnianie OAuth 2. Jest to metoda uwierzytelniania, w której aplikacja wysyła żądania API bez kontekstu użytkownika. Używaj tej metody jeżeli potrzebujesz dostępu tylko do publicznych danych typu do-odczytu.

Tak jak w przypadku OAuth 1a, na początku zarejestruj swojego klienta i zdobądź klucz konsumenta oraz sekret.

Następnie stwórz instancję AppAuthHandle, przekazując swój klucz konsumenta oraz sekret:

```
auth = tweepy.AppAuthHandler(consumer_key, consumer_secret)
```

Po otrzymaniu tokenu nosiciela możesz w końcu rozpocząć pracę:

```
api = tweepy.API(auth)
for tweet in tweepy.Cursor(api.search, q='tweepy').items(10):
    print(tweet.text)
```

ROZDZIAŁ 4

Snippety

4.1 Wprowadzenie

Tutaj znajdują się snippety, które mogą pomóc tobie w użytkowaniu Tweepy. Możesz także dodać swoje własne snippety lub usprawnić te, które się tutaj znajdują!

4.2 OAuth

```
auth = tweepy.OAuthHandler("consumer_key", "consumer_secret")

# Redirect user to Twitter to authorize
redirect_user(auth.get_authorization_url())

# Get access token
auth.get_access_token("verifier_value")

# Construct the API instance
api = tweepy.API(auth)
```

4.3 Stronnicowanie

```
# Iterate through all of the authenticated user's friends
for friend in tweepy.Cursor(api.friends).items():
    # Process the friend here
    process_friend(friend)

# Iterate through the first 200 statuses in the home timeline
for status in tweepy.Cursor(api.home_timeline).items(200):
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
    # Process the status here
    process_status(status)
```

4.4 FollowAll

Ten snippet będzie obserwował każdego kto obserwuje uwierzytelnionego użytkownika.

```
for follower in tweepy.Cursor(api.followers).items():
    follower.follow()
```

4.5 Obsługiwanie limitu wartości używając kursorów

Ponieważ kursory podnoszą `RateLimitError`w swoich metodach `next()`, obsługiwanie ich może być wykonane poprzez zapakowanie kursora jako iterator.

Uruchomienie tego snippeta wyświetli listę wszystkich użytkowników których obserwujesz, a którzy sami obserwują mniej niż 300 osób - między innymi by wykluczyć oczywiste spamboty - dodatkowo snippet będzie czekał 15 minut za każdym razem gdy osiągnie limit wartości.

```
# In this example, the handler is time.sleep(15 * 60),
# but you can of course handle it in any way you want.

def limit_handled(cursor):
    while True:
        try:
            yield cursor.next()
        except tweepy.RateLimitError:
            time.sleep(15 * 60)

for follower in limit_handled(tweepy.Cursor(api.followers).items()):
    if follower.friends_count < 300:
        print(follower.screen_name)
```

ROZDZIAŁ 5

Poradnik na temat kursorów

Ten poradnik wyjaśnia szczegóły dotyczące stronnicowania używając obiektów kursora.

5.1 Wprowadzenie

Stronnicowanie jest często używane w rozwoju Twitter API. Używa się go do iterowania osi czasu, listy użytkowników, bezpośrednich wiadomości itd. Aby wykonać stronnicowanie, musisz dostarczyć stronie lub kursorowi parametr z każdym ze swoich żądań. Problemem tego rozwiązania jest duża ilość boiler plate code wymaganego do zarządzania pętlą stronnicowania. Tweepy używa obiektu kursora by usprawnić stronnicowanie i zmniejszyć objętość kodu.

5.2 Stary sposób vs sposób kursora

Na początku zademonstrujemy iterowanie statusów na osi czasu uwierzytelnionego użytkownika. W taki sposób robiło się to „starą metodą" zanim wprowadzony został obiekt kursora.

```
page = 1
while True:
    statuses = api.user_timeline(page=page)
    if statuses:
        for status in statuses:
            # process status here
            process_status(status)
    else:
        # All done
        break
    page += 1  # next page
```

Jak widać musimy manualnie zarządzać parametrem „page" w naszej pętli stronnicowania. A teraz zaprezentujemy wersje kodu, która używa obiektu kursora:

```
for status in tweepy.Cursor(api.user_timeline).items():
    # process status here
    process_status(status)
```

Wygląda to dużo lepiej! Kursor załatwia za nas całą sprawę stronnicowania, co pozwala nam skupić się wyłącznie na przetwarzaniu rezultatów.

5.3 Przekazywanie parametrów do metody API

Co zrobić jeżeli muszę przekazać parametry do metody API?

```
api.user_timeline(id="twitter")
```

Jako że przekazujesz kursorowi obiekt wywoływany, nie możesz przekazać parametrów prosto do metody. Zamiast tego parametry są przekazywane do metody konstruktora kursora:

```
tweepy.Cursor(api.user_timeline, id="twitter")
```

Kursor przekaże parametry do metody gdy tylko stworzy żądanie.

5.4 Obiekty czy strony

Do tej pory zademonstrowaliśmy iteracje stronnicowania dla obiektu. Co jeżeli zamiast tego chcesz przetworzyć rezultaty dla strony? Użyj do tego metody pages():

```
for page in tweepy.Cursor(api.user_timeline).pages():
    # page is a list of statuses
    process_page(page)
```

5.5 Limity

Co jeżeli chcesz by zwrócone zostało n obiektów lub stron? Musisz wtedy przekazać do metod items() lub pages() limit, który chcesz nałożyć.

```
# Only iterate through the first 200 statuses
for status in tweepy.Cursor(api.user_timeline).items(200):
    process_status(status)

# Only iterate through the first 3 pages
for page in tweepy.Cursor(api.user_timeline).pages(3):
    process_page(page)
```

ROZDZIAŁ 6

Rozszerzone Tweety

Te informacje uzupełniają Dokumentację aktualizacji tweetów Twittera.

6.1 Wprowadzenie

On May 24, 2016, Twitter announced changes to the way that replies and URLs are handled and published plans around support for these changes in the Twitter API and initial technical documentation describing the updates to Tweet objects and API options.[1] On September 26, 2017, Twitter started testing 280 characters for certain languages,[2] and on November 7, 2017, announced that the character limit was being expanded for Tweets in languages where cramming was an issue.[3]

6.2 Standardowe metody API

Każda metoda `tweepy API`, która zwraca obiekt Status akceptuje nowy parametr `tweet_mode`. Poprawne wartości dla tego parametru to `compat` oraz `extened`, które dają odpowiednio tryb kompatybilności oraz tryb rozszerzony. Domyślny tryb (gdy nie ma podanego parametru) to tryb kompatybilności.

6.2.1 Tryb kompatybilności

Domyślnie, używając trybu kompatybilności, atrybut `text` obiektu Status zwrócony przez metody `tweepy API` jest obcięty do 140 znaków, tak jak jest to wymagane. Gdy zachodzi obcinanie, atrybut `truncated` obiektu Status jest `True` i tylko jednostki, które są całkowicie zawarte w dostępnych 140 znakach będą zawarte w atrybucie `entities`. Zostanie także zaobserwowane to, że atrybut ‘‘text‘ obiektu Satus jest obcięty, ponieważ będzie on zakończony elipsą, spacją oraz skróconym permamentnym URL do tweeta.

[1] https://twittercommunity.com/t/upcoming-changes-to-simplify-replies-and-links-in-tweets/67497
[2] https://twittercommunity.com/t/testing-280-characters-for-certain-languages/94126
[3] https://twittercommunity.com/t/updating-the-character-limit-and-the-twitter-text-library/96425

6.2.2 Tryb rozszerzony

Używając trybu rozszerzonego, atrybut `text` obiektu Status zwrócony przez metody `tweepy API` jest zastąpiony przez atrybut `full_text`, który zawiera cały, nieobcięty tekst tweeta. Atrybut `truncated` obiektu Status jest `False` a atrybut `entities` zawiera wszystkie jednostki. Dodatkowo, obiekt Status będzie posiadał atrybut `display_text_range`, szyk dwóch indeksów wskaźników kodu Unicode, które identyfikują włączny start i wyłączny koniec wyświetlanej zawartości tweeta.

6.3 Przesyłanie strumieniowe

Domyśnie, obiekty Status ze strumieni mogą zawierać atrybut `extended_tweet` reprezentujący równowartość pól w nieprzetworzonych danych/właściwych danych dla tweeta. Ten atrybut/pole będzie istnieć tylko dla rozszerzonych tweeetów zawierających słownik podpól. Podpole/klucz `full_text` tego słownika będzie zawierać pełny, nieobcięty tekst tweeta a podpole/klucz `entities` będzie zawierać pełny zbiór jednostek. Jeżeli pojawią się rozszerzone jednostki to podpole/klucz `extended_entities` będzie zawierać pełen ich zbiór. Dodatkowo, podpole/klucz `display_text_range` będzie zawierać szyk dwóch indeksów wskaźników kodu Unicode, które identyfikują włączny start i wyłączny koniec wyświetlanej zawartości tweeta.

6.4 Obsługa retweetów

Używając rozszerzonego trybu dla retweetów, atrybut `full_text` obiektu Status może być skrócony poprzez elipsę zamiast zawierania całości tekstu retweeta. Jednakże, ponieważ atrybut `retweeted_status` (dla obiektu Status, który jest retweetem) jest sam w sobie obiektem Statusu, to atrybut `full_text` dla obiektu Status retweeta, może być użyty zamiennie.

Działa to podobnie dla obiektu/danych właściwych, które są retweetami ze strumieni. Słownik od atrybutu/pola `extended_tweet` zawiera podpole/klucz `full_text`, który może być obcięty elipsą. Zamiast tego może być użyty atrybut/pole `extended_tweet` Statusu retweeta (od atrybutu/pola `retweeted_status`).

6.5 Przykłady

Posiadając istniejący obiekt `tweepy.API` oraz `id` dla tweeta, można wyświetlić cały tekst tweeeta lub jeżeli jest to retweet, cały tekst retweetowanego tweeta:

```
status = api.get_status(id, tweet_mode="extended")
try:
    print(status.retweeted_status.full_text)
except AttributeError:  # Not a Retweet
    print(status.full_text)
```

Jeżeli `status` to retweet to `status.full_text` może być obcięty.

Ten odbiornik zdarzeń dla `StreamListener` wyświetla pełny tekst tweeta, lub jeżeli jest to retweet, pełny tekst retweetowanego tweeta:

```
def on_status(self, status):
    if hasattr(status, "retweeted_status"):  # Check if Retweet
        try:
            print(status.retweeted_status.extended_tweet["full_text"])
        except AttributeError:
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
            print(status.retweeted_status.text)
    else:
        try:
            print(status.extended_tweet["full_text"])
        except AttributeError:
            print(status.text)
```

Jeżeli `status` to retweet to nie będzie on posiadał atrybutu `extended_tweeet` a `status.text` moze być obcięty.

ROZDZIAŁ 7

Przesyłanie strumieniowe z Tweepy

Tweepy ułatwia używanie API Twittera do przesyłania strumieniowego poprzez obsługę uwierzytelniania i połączenia, tworzenia i niszczenia nadchodzących wiadomości oraz częściowe przekierowywanie wiadomości.

Celem tej strony jest objaśnienie pierwszych kroków na temat rozpoczęcia przesyłania strumieniowego Twittera używając Tweepy. Niektóre funkcje przesyłania strumieniowego Tweepy nie są tu wyjaśnione. Po więcej informacji sprawdź streaming.py w kodzie źródłowym Tweepy.

Uwierzytelnienie API jest wymagane by uzyskać dostęp do strumienia Twittera. Sprawdź *Poradnik uwierzytelniania* jeżeli potrzebujesz pomocy z uwierzytelnianiem.

7.1 Podsumowanie

API przesyłania strumieniowego Twittera jest używane do pobierania wiadomości z Twittera w czasie rzeczywistym. Jest to przydatne do zdobywania dużej ilości tweetów lub do tworzenia przekazu na żywo używając strumienia strony lub strumienia użytkownika. Zobacz Dokumentację API przesyłania strumieniowego Twittera.

API przesyłania strumieniowego różni się od REST API ponieważ REST API używane jest do *ściągania* danych z Twittera a API przesyłania strumieniowego *wypycha* wiadomości do trwałej sesji. Pozwala to API przesyłania strumieniowego do pobrania większej ilości danych w czasie rzeczywistym niż jest to możliwe używając REST API.

W Tweepy, instancja **tweepy.Stream** ustanawia sesję przesyłania strumieniowego i przekierowywuje wiadomości do instancji **StreamListener**. Metoda **on_data** odbiornika strumienia otrzymuje wszystkie wiadomości i nazywa funkcje według typu wiadomości. Domyślny **StreamListener** może sklasyfikować większość prostych wiadomości z Twittera i przekierować je do odpowiednio nazwanych metod, jednak metody te są tylko pniami.

Używanie API przesyłania strumieniowego zawiera się w trzech krokach.

1. Utwórz klasę dziedziczącą od **StreamListener**
2. Używając tej klasy stwórz obiekt **Stream**
3. Połącz się z Twitter API używając **Stream**.

7.2 Krok 1: Tworzenie StreamListener

Ten prosty odbiornik strumienia wyświetla tekst statusu. Metoda **on_data** używana przez **StreamListener** Tweepy wygodnie przekazuje dane ze statuu do metody **on_status**. Stwórz klasę **MyStreamListener* dziedziczącą od **StreamListener** i nadpisującą **on_status**.:

```
import tweepy
#override tweepy.StreamListener to add logic to on_status
class MyStreamListener(tweepy.StreamListener):

    def on_status(self, status):
        print(status.text)
```

7.3 Krok 2: Tworzenie Stream

Do przesyłania strumieniowego potrzebne jest API. Zajrzyj do *Poradnik uwierzytelniania* by dowiedzieć się jak zdobyć obiekt API. Gdy masz już API oraz odbiornik statusu, możesz stworzyć własny obiekt strumienia.:

```
myStreamListener = MyStreamListener()
myStream = tweepy.Stream(auth = api.auth, listener=myStreamListener)
```

7.4 Krok 3: Uruchamianie strumienia

W Tweepy dostępne są różne strumienie Twittera. W większości przypadków będą one używać filtrów user_stream lub sitestream. Po więcej informacji na temat możliwości i ograniczeń poszczególnych strumieni zajrzyj do Dokumentacji API przesyłania strumieniowego Twittera.

W tym przykładzie użyty zostanie **filter** do przesłania strumieniowego wszystkich tweetów zawierających słowo *python*. Parametr *track* jest szykiem wyszukiwanych pojęć.

```
myStream.filter(track=['python'])
```

W tym przykładzie pokazane zostanie jak użyć **filter** do przesyłania strumieniowego tweetów wybranego użytkownika. Parametr **follow** jest szykiem zawierającym ID.

```
myStream.filter(follow=["2211149702"])
```

Prostym sposobem na znalezienie pojedyńczego ID jest użycie jednej z wielu stron internetowych do konwersji: szukaj «jakie jest moje twitter ID».

7.5 Kilka innych wskaźników

7.5.1 Przesyłanie strumieniowe asynchroniczne

Strumienie nie zatrzymują się dopóki połączenie nie zostanie zamknięte, co zablokuje nić. Tweepy oferuje wygodny parametr **is_async** w **filter** co pozwala strumieniowi działać na nowej nici. Na przykład:

```
myStream.filter(track=['python'], is_async=True)
```

7.5.2 Obsługa błędów

Używając API przesyłania strumieniowego Twittera należy być świadomym niebezpieczeństw związanych z limitowaniem współczynników. Jeżeli klient przekroczy liczbę prób połqczenia w określonym czasie z API przesyłania strumieniowego to zwrócony zostanie błąd 420. Czas oczekiwania na ponowną próbę połączenia po otrzymaniu błędu 420 będzie gwałtownie zwiększał się za każdą kolejną nieudaną próbą.

Stream Listener Tweepy przekazuje kod błędu do pnia **on_error**. Domyślne ustawienie zwraca **False** dla wszytkich kodów, jednak możliwe jest nadpisanie ustawień tak by umożliwiły one Tweepy ponowne połączenie się po otrzymaniu części lub wszystkich kodów. Możliwe jest to używając strategii rekomendowanych w Dokumentacji łączenia z API przesyłania strumieniowego Twittera.:

```
class MyStreamListener(tweepy.StreamListener):

    def on_error(self, status_code):
        if status_code == 420:
            #returning False in on_error disconnects the stream
            return False

        # returning non-False reconnects the stream, with backoff.
```

Po więcej informacji na temat kodów błędów z API Twittera zajrzyj do Dokumentacji kodów odpowiedzi Twittera.

ROZDZIAŁ 8

Odniesienia do API

Ta strona zawiera podstawową dokumentację do modułu Tweepy.

ROZDZIAŁ 9

tweepy.api — Twitter API wrapper

class API([*auth_handler=None*][, *host='api.twitter.com'*][, *search_host='search.twitter.com'*][, *cache=None*][, *api_root='/1'*][, *search_root=''*][, *retry_count=0*][, *retry_delay=0*][, *retry_errors=None*][, *timeout=60*][, *parser=ModelParser*][, *compression=False*][, *wait_on_rate_limit=False*][, *wait_on_rate_limit_notify=False*][, *proxy=None*])

Ta klasa dostarcza wrapper dla API tak jak dostarczono to przez Twitter. Funkcje dostarczone w tej klasie są zapisane poniżej.

Parametry

- **auth_handler** – program obsługi autentykacji, który zostanie użyty
- **host** – generalny host API
- **search_host** – przeszukaj hosta API
- **cache** – pamięć podręczna backend, która zostanie użyta
- **api_root** – generalna ścieżka korzenia API
- **search_root** – przeszukaj ścieżkę korzenia API
- **retry_count** – domyślna liczba powtórzeń gdy wystąpi błąd
- **retry_delay** – liczba sekund oczekiwania pomiędzy powtórzeniami
- **retry_errors** – który kod statusu HTTP zostanie użyty do powtórzenia
- **timeout** – Maksymalna ilość czasu oczekiwania na odpowiedź od Twitter
- **parser** – Obiekt, który zostanie użyty do analizy odpowiedzi od Twitter
- **compression** – Czy do żądań ma zostać użyta kompresja GZIP
- **wait_on_rate_limit** – Czy automatycznie oczekiwać na wyczerpanie limitów wskaźników
- **wait_on_rate_limit_notify** – czy wyświetlać notyfikacje, gdy Tweepy oczekuje na wyczerpanie limitów wskaźników
- **proxy** – Pełen URL proxy HTTPS, które jest użyte do połączenia z Twitter.

9.1 Metody osi czasu

API.**home_timeline**([*since_id*][, *max_id*][, *count*][, *page*])
Zwraca 20 ostatnich statusów w tym retweety zapostowane przez zuwierzytelnionego użytkownika i jego znajomych. Jest to to samo co /timeline/home.

Parametry

- **since_id** – Zwraca tylko statusy z ID większym (tzn. nowszym) niż określone ID.
- **max_id** – Zwraca tylko statusy z ID mniejszym (tzn. starszym) lub równym określonemu ID.
- **count** – Liczba wyników do pobrania na stronę.
- **page** – Określa którą stronę wyników otrzymać. Uwaga: istnieją limity stronnicowania.

Typ zwracany lista obiektów `Status`

API.**statuses_lookup**(*id_*[, *include_entities*][, *trim_user*][, *map_*][, *include_ext_alt_text*][, *include_card_uri*])
Zwraca pełne obiekty Tweet, do 100 tweetow na jedno żądanie. Sprecyzowane w parametrze `id_`.

Parametry

- **id_** – Lista ID tweetów do wyszukania, maksymalnie 100
- **include_entities** – Ustawione jako false powoduje, że węzeł jednostek nie będzie zawarty. Domyślnie False.
- **trim_user** – Boolean wskazujacy czy dostarczyć ID użytkowników zamiast kompletnych obiektów user. Domyślnie False.
- **map_** – Boolean wskazujący czy zawarte będą tweety, które nie mogą być pokazywane. Domyślnie ustawione jako False.
- **include_ext_alt_text** – Jeżeli alt tekst został dodany do którejś z dołączonych jednostek to ten parametr zwróci wartość ext_alt_text w kluczu top-level dla tej jednostki mediów.
- **include_card_uri** – Boolean wkazujący czy otrzymany tweet powinien zawierać atrybut card_uri gdy do tweeta dołączona jest karta ads oraz gdy karta jest dołączona używając wartośći card_uri.

Typ zwracany lista obiektów `Status`

API.**user_timeline**([*id/user_id/screen_name*][, *since_id*][, *max_id*][, *count*][, *page*])
Zwraca 20 ostatnich statusów zapostowanych przez zuwierzytelnionego użytkownika lub wyznaczonego użytkownika. Możliwe jest także zażądanie osi czasu innego użytkownika za pomocą parametru id.

Parametry

- **id** – Określa ID lub nazwę wyświetlaną użytkownika.
- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **since_id** – Zwraca tylko statusy z ID większym (tzn. nowszym) niż określone ID.
- **max_id** – Zwraca tylko statusy z ID mniejszym (tzn. starszym) lub równym określonemu ID.

- **count** – Liczba wyników do pobrania na stronę.
- **page** – Określa którą stronę wyników otrzymać. Uwaga: istnieją limity stronnicowania.

Typ zwracany lista obiektów `Status`

`API.`**`retweets_of_me`**(*[since_id][, max_id][, count][, page]*)
Zwraca 20 najnowszych tweetów od zuwierzytelnionego użytkownika, które zostały zretweetowane przez innych.

Parametry

- **since_id** – Zwraca tylko statusy z ID większym (tzn. nowszym) niż określone ID.
- **max_id** – Zwraca tylko statusy z ID mniejszym (tzn. starszym) lub równym określonemu ID.
- **count** – Liczba wyników do pobrania na stronę.
- **page** – Określa którą stronę wyników otrzymać. Uwaga: istnieją limity stronnicowania.

Typ zwracany lista obiektów `Status`

`API.`**`mentions_timeline`**(*[since_id][, max_id][, count]*)
Zwraca 20 najnowszych wzmianek, w tym retweety.

Parametry

- **since_id** – Zwraca tylko statusy z ID większym (tzn. nowszym) niż określone ID.
- **max_id** – Zwraca tylko statusy z ID mniejszym (tzn. starszym) lub równym określonemu ID.
- **count** – Liczba wyników do pobrania na stronę.

Typ zwracany lista obiektów `Status`

9.2 Metody statusu

`API.`**`get_status`**(*id[, trim_user][, include_my_retweet][, include_entities][, include_ext_alt_text][, include_card_uri]*)
Zwraca pojedyńczy status określony przez parametr ID.

Parametry

- **id** – Numeryczne ID statusu.
- **trim_user** – Boolean wskazujacy czy dostarczyć ID użytkowników zamiast kompletnych obiektów user. Domyślnie False.
- **include_my_retweet** – Boolean wskazujący czy tweety, które zostały zretweetowane przez zuwierzytelnionego użytkownika powinny zawierać dodatkowy węzeł current_user_retweet, który zawiera ID statusu źródłowego dla retweeta.
- **include_entities** – Ustawione jako false powoduje, że węzeł jednostek nie będzie zawarty. Domyślnie False.
- **include_ext_alt_text** – Jeżeli alt tekst został dodany do którejś z dołączonych jednostek to ten parametr zwróci wartość ext_alt_text w kluczu top-level dla tej jednostki mediów.

- **include_card_uri** – Boolean wkazujący czy otrzymany tweet powinien zawierać atrybut card_uri gdy do tweeta dołączona jest karta ads oraz gdy karta jest dołączona używając wartośći card_uri.

Typ zwracany obiekt `Status`

`API.`**`update_status`**(*status*[, *in_reply_to_status_id*][, *auto_populate_reply_metadata*][, *exclude_reply_user_ids*][, *attachment_url*][, *media_ids*][, *possibly_sensitive*][, *lat*][, *long*][, *place_id*][, *display_coordinates*][, *trim_user*][, *enable_dmcommands*][, *fail_dmcommands*][, *card_uri*])

Aktualizuje status zuwierzytelnionego użytkownika.

Dla każdej próby aktualizacji, jej tekst jest porównywany z najnowszym tweetem zuwierzytelnionego użytkownika. Każda próba, której rezultatem był by duplikat, zostanie zablokowana, co spowoduje wystąpienie błędu 403. Użytkownik nie może wysłać tego samego statusu dwa razy pod rząd.

Mimo, że nie jest to limitowane przez API, użytkownik może stworzyć maksymalnie określoną liczbę tweetów za jednym razem. Jeżeli liczba aktualizacji zapostowana przez użytkownika osiągnie aktualny limit, metoda tha zwróci błąd HTTP 403.

Parametry

- **status** – Tekst statusu aktualizacji.
- **in_reply_to_status_id** – ID istniejącego statusu dla którego odpowiada aktualizacja. Uwaga: Ten parametr zostanie zignorowany, chyba, że autor tweeta, do którego się odnosi jest wspomniany w ramach tekstu statusu. Tak więc w aktualizacji musisz zawrzeć @username, gdzie username jest autorem wspomnianego tweeta.
- **auto_populate_reply_metadata** – Jeżeli ustawione jako true i użyte w in_reply_to_status_id, głowne @mentions będą wyszukane w oryginalnym tweecie i dodane do nowego tweeta. To dołączy @mentions do metaaty istniejącego tweeta wraz z rozwojem łańcucha tweetów, póki nie zostanie osiągnięty limit @mentions. Odpowiedź nie powiedzie się, w przypadkach gdzie oryginalny tweet został usunięty.
- **exclude_reply_user_ids** – Gdy użyte z auto_populate_reply_metadata, lista ID użytkowników oddzielona przecinkami zostanie usunięta z wygenerowanego przez serwer prefiksu @mentions na rozszerzonym tweecie. Zauważ, że główne @mention nie mogą być usunięte ponieważ zepsuło by to semantykę in-reply-to-status-id. Próby ich usunięcia będą dyskretnie zignorowane.
- **attachment_url** – Aby URL nie był policzony w treść statusu rozszerzonego tweeta, dołacz go jako załącznik. URL ten musi być permalinkiem tweeta lub linkiem Wiadomości Bezpośredniej. Inne nie-twitterowe URL muszą pozostać w tekście statusu. URL przekazane do parametru attachment_url, które nie są przyporządkowane do permalinku tweeta lub Wiadomości Bezpośredniej, nie stworzą tweeta i spowodują wystąpienie wyjątku.
- **media_ids** – Lista media_ids powiązanych z tweetem. Jeden tweet może zawierać maksymalnie 4 zdjęcia, 1 animowany GIF lub 1 video
- **possibly_sensitive** – Jeżeli prześlesz media tweeta, które mogą zawierać wrażliwy kontent taki jak nagość lub procedury medyczne to ta wartość musi być ustawiona jako true.
- **lat** – Szerokość geograficzna lokacji do której odnosi się tweet. Ten parametr zostanie zignorowany, chyba, że znajduje się pomiędzy -90.0 a +90.0 (północ to wartość dodatnia). Zostanie on także zignorowany jeżeli nie ma on dopasowanego parametru długości geograficznej.
- **long** – Długość geograficzna do której odnosi się tweet. Poprawne wartości zawieraja się między -180.0 a +180.0 (wschód to wartość dodatnia). Ten parametr zostanie zignorowany

jeżeli przekracza ten zakres, nie jest liczbą, gdy geo_enabled jest wyłączone oraz jeżeli nie ma dopasowanego parametru szerokości geograficznej.

- **place_id** – Miejsce gdzieś na świecie.
- **display_coordinates** – Czy wbić szpilkę w miejsce o dokładnych koordynatach, z których został wysłany Tweet.
- **trim_user** – Boolean wskazujacy czy dostarczyć ID użytkowników zamiast kompletnych obiektów user. Domyślnie False.
- **enable_dmcommands** – Gdy ustawione jako true, pozwala używać krótkich komend do wysyłania Wiadomości Bezpośrednich jako część tekstu statusu do wysłania do użytkownika. Jeżeli ustawione jako false, powyższe rozwiązanie zostanie wyłączone i zawarte zostaną głowne znaki zapostowanego tekstu statusu.
- **fail_dmcommands** – Gdy ustawione jako true, powoduje, że jakikolwiek tekst statusu rozpoczęty komendą krótka wywołuje błąd API. Gdy ustawione jako true, pozwala krótkim komendom na zostanie wysłanym w tekście statusu i bycie użytym przez API
- **card_uri** – Powiązuje karte reklamową z tweetem używając wartości card_uri z jakiejkolwiek odpowiedzi od karty reklamowej.

Typ zwracany obiekt `Status`

API.**update_with_media**(*filename*[, *status*][, *in_reply_to_status_id*][, *auto_populate_reply_metadata*][, *lat*][, *long*][, *source*][, *place_id*][, *file*])

Nierekomendowane: używa *API.media_upload()*. Zaktualizuj status zuwierzytelnionego użytkownika. Statusy ktore są duplikatami lub są za długie będą dyskretnie ignorowane.

Parametry

- **filename** – Nazwa pliku obrazu do wrzucenia. Będzie ona automatycznie otwarta, chyba, że *file* zostanie określone.
- **status** – Tekst statusu aktualizacji.
- **in_reply_to_status_id** – ID istniejącego już statusu, dla ktorego aktualizacja jest odpowiedzią.
- **auto_populate_reply_metadata** – Czy automatycznie zawierać @wzmianki w metadacie statusu.
- **lat** – Szerokość geograficzna do której odnosi się tweet.
- **long** – Długość geograficzna do której odnosi się tweet.
- **source** – Źródło aktualizacji. Wspierane tylko przez Identi.ca. Twitter ignoruje ten parametr.
- **place_id** – Twitter ID lokacji, która jest wymieniona w tweecie gdy użytkownik ma włączoną geolokację.
- **file** – Plik, który zostanie użyty zamiast otwierania *filename*. *filename* jest nadal wymagany dla detekcji typu MIME oraz do używania pola formularzu w danych POST.

Typ zwracany obiekt `Status`

API.**destroy_status**(*id*)

Niszczy status określony przez parametr ID. Zuwierzytelniony użytkownik musi być autorem statusu by go zniszczyć.

Parametry **id** – Numeryczne ID statusu.

Typ zwracany obiekt `Status`

API.**retweet**(*id*)
Retweetuje tweet. Wymaga id tweeta, który retweetujesz.

Parametry **id** – Numeryczne ID statusu.

Typ zwracany obiekt `Status`

API.**retweeters**(*id*[, *cursor*][, *stringify_ids*])
Zwraca do maksymalnie 100 ID użytkowników, należących do użytkowników, którzy zretweetowali tweeta określonego przez paremetr ID.

Parametry

- **id** – Numeryczne ID statusu.
- **cursor** – Dzieli wyniki na strony. Ustaw wartość jako -1 by rozpocząć stronnicowanie. Dostarcz wartości tak jak są zwracane w tekście opowiedzi. Atrybuty next_cursor i previous_cursor używane są do przechoznenia na przód i w tył.
- **stringify_ids** – ID będą zwracane jago ciąg znaków.

Typ zwracany list of Integers

API.**retweets**(*id*[, *count*])
Zwraca do maksymalnie 100 pierwszych retweetów wybranego tweeta.

Parametry

- **id** – Numeryczne ID statusu.
- **count** – Określa liczbę retweetów do pozyskania.

Typ zwracany lista obiektów `Status`

API.**unretweet**(*id*)
Odtweetowuje zretweetowany status. Wymaga id retweeta.

Parametry **id** – Numeryczne ID statusu.

Typ zwracany obiekt `Status`

9.3 Metody użytkownika

API.**get_user**(*id/user_id/screen_name*)
Zwraca informacje o wybranym użytkowniku.

Parametry

- **id** – Określa ID lub nazwę wyświetlaną użytkownika.
- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.

Typ zwracany obiekt `User`

API.**me**()
Zwraca informacje o zuwierzytelnionym użytkowniku.

Typ zwracany obiekt `User`

API.**friends**([*id/user_id/screen_name*][, *cursor*][, *skip_status*][, *include_user_entities*])
Zwraca znajomych użytkownika po 100 na raz, posortowanych według chronologii dodania do znajomych. Jeżeli użytkownik nie jest określony to domyślnie będzie użyty zuwierzytelniony użytkownik.

Parametry

- **id** – Określa ID lub nazwę wyświetlaną użytkownika.
- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **cursor** – Dzieli wyniki na strony. Ustaw wartość jako -1 by rozpocząć stronnicowanie. Dostarcz wartości tak jak są zwracane w tekście opowiedzi. Atrybuty next_cursor i previous_cursor używane są do przechoznenia na przód i w tył.
- **count** – Liczba wyników do pobrania na stronę.
- **skip_status** – Boolean wskazujący czy będą zawarte w zwróconych obiektach user. Domyślnie False.
- **include_user_entities** – Gdy ustawione jako False to jednostki węzła obiektu user nie będą zawarte. Domyślnie True.

Typ zwracany lista obiektów `User`

API.**followers**([*id/screen_name/user_id*][, *cursor*])
Zwraca obserwujących użytkownika według chronologii rozpoczęcia przez nich obserwowania. Jeżeli użytkownik nie jest określony to domyślnie będzie użyty zuwierzytelniony użytkownik.

Parametry

- **id** – Określa ID lub nazwę wyświetlaną użytkownika.
- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **cursor** – Dzieli wyniki na strony. Ustaw wartość jako -1 by rozpocząć stronnicowanie. Dostarcz wartości tak jak są zwracane w tekście opowiedzi. Atrybuty next_cursor i previous_cursor używane są do przechoznenia na przód i w tył.
- **count** – Liczba wyników do pobrania na stronę.
- **skip_status** – Boolean wskazujący czy będą zawarte w zwróconych obiektach user. Domyślnie False.
- **include_user_entities** – Gdy ustawione jako False to jednostki węzła obiektu user nie będą zawarte. Domyślnie True.

Typ zwracany lista obiektów `User`

API.**lookup_users**([*user_ids*][, *screen_names*][, *include_entities*][, *tweet_mode*])
Zwraca fully-hydrated obiekt użytkownika, maksymalnie 100 użytkowników na jedno żądanie.

Należy mieć na uwadze kilka kwestii używając tej metody.

- Musisz obserwować chronionego użytkownika by móc zobaczyć ich najnowszą zmianę statusu. Jeżeli nie obserwujesz go jego status zostanie usunięty.

- Porządek ID użytkowników lub nazw wyświetlanych może nie być rownoważny z porządkiem użytkowników w zwróconym szyku.
- Jeżeli żądany użytkownik jest nieznany, zablokowany lub usunięty to nie zostanie on zwrócony do listy wyników.
- Jeżeli żadne z twoich wyszukiwań nie spełnia wymagań poprzez zwrócenie obiektu użytkownika, to wystąpi wtedy błąd HTTP 404.

Parametry

- **user_ids** – Lista ID użytkowników, masymalnie 100 może być zawartych w pojedyńczym żądaniu.
- **screen_names** – Lista nazw wyświetlanych, masymalnie 100 może być zawartych w pojedyńczym żądaniu.
- **include_entities** – Ustawione jako false powoduje, że węzeł jednostek nie będzie zawarty. Domyślnie False.
- **tweet_mode** – Poprawne wartości żądań są kompatybilne i rozszerzone, co nadaje im odpowiednio tryb kompatybilności lub trybowi rozszerzony, dla tweetów zawierających ponad 140 znaków.

Typ zwracany lista obiektów `User`

`API.`**`search_users`**(*q*[, *count*][, *page*])
Uruchom wyszukiwanie dla użytkowników podobne do przycisku Znajdź Ludzi na Twitter.com; używając teego API zostaną zwrócone te same rezultaty co w przypadku wyszukiwania ludzi. Używając tego API możliwe jest uzyskanie makymalnie 1000 pierwszych wyników.

Parametry

- **q** – Zapytanie, które jest uruchomione przeciwko wyszukiwaniom ludzi.
- **count** – Określa liczbę statusów do uzyskania. Nie może być większe niz 20.
- **page** – Określa którą stronę wyników otrzymać. Uwaga: istnieją limity stronnicowania.

Typ zwracany lista obiektów `User`

9.4 Metody wiadomości bezpośrednich

`API.`**`get_direct_message`**([*id*][, *full_text*])
Zwraca wybraną wiadomość bezpośrednią.

Parametry

- **id** – The id of the Direct Message event that should be returned.
- **full_text** – Boolean wkazujący czy powinna być zwrócona całość tekstu wiadomości. Ustawione jako False powoduje, że wiadomość zostanie obcięta do 140 znaków. Domyślnie False.

Typ zwracany obiekt `DirectMessage`

`API.`**`list_direct_messages`**([*count*][, *cursor*])
Zwraca wszystkie zdarzenia Wiadomości Bezpośrednich (otrzymane i wysłane) w ostatnich 30 dniach. Posortowane w odwrotnym porządku chronologicznym.

Parametry

- **count** – Liczba wyników do pobrania na stronę.
- **cursor** – Dzieli wyniki na strony. Ustaw wartość jako -1 by rozpocząć stronnicowanie. Dostarcz wartości tak jak są zwracane w tekście opowiedzi. Atrybuty next_cursor i previous_cursor używane są do przechoznenia na przód i w tył.

Typ zwracany lista obiektów `DirectMessage`

`API.`**`send_direct_message`**(*recipient_id*, *text*[, *quick_reply_type*][, *attachment_type*][, *attachment_media_id*])

Wysyła nową wiadomość bezpośrednią od zuwierzytelniongo użytkownika do wybranego użytkownika.

Parametry

- **recipient_id** – ID użytkownika, który ma otrzymać wiadomość.
- **text** – Tekst twojej Wiadomości Bezpośrednij. Maksymalna długość: 10000 znaków.
- **quick_reply_type** – Typ Szybkiej Odpowiedzi do pokazania użytkownikowi * options - szyk obiektów Opcji (maks 20). * text_input - tekst obiektu Input. * location - obiekt Lokacji.
- **attachment_type** – Typ załącznika. Może być mediami lub lokacją.
- **attachment_media_id** – ID mediów do powiązania z wiadomością. Wiadomość bezpośrednia może odnosić si tylko do pojedyńczeego media_id.

Typ zwracany obiekt `DirectMessage`

`API.`**`destroy_direct_message`**(*id*)

Usuwa wiadomość bezpośrednią określona w wymaganym parametrze ID. Zuwierzytelniony użytkownik musi być odbiorcą tej konkretnej wiadomości bezpośredniej. Wiadomości bezpośrednie mogą być usunięte tylko z intefejsu dostarczonego kontektu użytkownika. Inni członkowie konwersacji nadal mają dostęp do Wiadomości Bezpośrednich.

Parametry **id** – ID Wiadomości Bezpośredniej która ma zostać usunięta.

Typ zwracany None

9.5 Metody Znajomych

`API.`**`create_friendship`**(*id/screen_name/user_id*[, *follow*])

Stwórz nową znajomość z wybranym użytkownikiem (obserwuj).

Parametry

- **id** – Określa ID lub nazwę wyświetlaną użytkownika.
- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **follow** – Włącz notyfikacje dla wybranego użytkownika wraz z dodaniem go do znajomych.

Typ zwracany obiekt `User`

`API.`**`destroy_friendship`**(*id/screen_name/user_id*)

Zerwij znajomość z wybranym użytkownikim (przestań obserwować).

Parametry

- **id** – Określa ID lub nazwę wyświetlaną użytkownika.
- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.

Typ zwracany obiekt `User`

`API.`**show_friendship**(*source_id/source_screen_name*, *target_id/target_screen_name*)
Zwraca szczegółowe informacje na temat znajomości pomiędzy dwoma użytkownikami.

Parametry

- **source_id** – user_id podanego użytkownika.
- **source_screen_name** – screen_name podanego użytkownika.
- **target_id** – user_id wybranego użytkownika.
- **target_screen_name** – screen_name wybranego użytkownika.

Typ zwracany obiekt `Friendship`

`API.`**lookup_friendships**(*user_ids/screen_names*)
Returns the relationships of the authenticated user to the list of up to 100 screen_names or user_ids provided.

Parametry

- **user_ids** – Lista ID użytkowników, masymalnie 100 może być zawartych w pojedyńczym żądaniu.
- **screen_names** – Lista nazw wyświetlanych, masymalnie 100 może być zawartych w pojedyńczym żądaniu.

Typ zwracany `Relationship` object

`API.`**friends_ids**(*id/screen_name/user_id*[, *cursor*])
Zwraca szyk zawierający ID użytkowników obserwowanych przez wybranego użytkownika.

Parametry

- **id** – Określa ID lub nazwę wyświetlaną użytkownika.
- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **cursor** – Dzieli wyniki na strony. Ustaw wartość jako -1 by rozpocząć stronnicowanie. Dostarcz wartości tak jak są zwracane w tekście opowiedzi. Atrybuty next_cursor i previous_cursor używane są do przechoznenia na przód i w tył.

Typ zwracany list of Integers

`API.`**followers_ids**(*id/screen_name/user_id*)
Zwraca szyk zawierający ID użytkowników obserwujących wybranego użytkownika.

Parametry

- **id** – Określa ID lub nazwę wyświetlaną użytkownika.
- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.

- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **cursor** – Dzieli wyniki na strony. Ustaw wartość jako -1 by rozpocząć stronnicowanie. Dostarcz wartości tak jak są zwracane w tekście opowiedzi. Atrybuty next_cursor i previous_cursor używane są do przechoznenia na przód i w tył.

Typ zwracany list of Integers

9.6 Metody konta

API.**verify_credentials**([*include_entities*][, *skip_status*][, *include_email*])
Potwierdza, że dane podanego użytkownika są prawidłowe.

Parametry

- **include_entities** – Ustawione jako false powoduje, że węzeł jednostek nie będzie zawarty. Domyślnie False.
- **skip_status** – Boolean wskazujący czy będą zawarte w zwróconych obiektach user. Domyślnie False.
- **include_email** – Gdy ustawione jako true, e-mail będzie zwrócony w obiekcie użytkownika jako ciąg znaków.

Typ zwracany obiekt `User` jeżeli dane są prawidłowe, w innym przypadku False

API.**rate_limit_status**()
Zwraca aktualne limity wartości dla metod należących do wybranej rodziny zasobów. Używając tego uwierzytelniania (tylko dla aplikacji), odpowiedź tej metody wskaże limit kontekstu dla tego uwierzytelniania.

Parametry **resources** – Odseparowana przecinkami lista rodziny zasobów. Powinieneś znać aktualny limit wartośći dyspozycji dla.

Typ zwracany obiekt `JSON`

API.**update_profile_image**(*filename*)
Aktualizuje zdjęcie profilowe zuwierzytelnionego użytkownika. Poprawne formaty: GIF, JPG oraz PNG

Parametry **filename** – ścieżka lokalna dla obrazu, który ma być wrzucony. Nie jest to remote URL!

Typ zwracany obiekt `User`

API.**update_profile_background_image**(*filename*)
Aktualizuje zdjęcie w tle użytkownika. Poprawne formaty: GIF, JPG oraz PNG

Parametry **filename** – ścieżka lokalna dla obrazu, który ma być wrzucony. Nie jest to remote URL!

Typ zwracany obiekt `User`

API.**update_profile**([*name*][, *url*][, *location*][, *description*])
Ustawia wartości, które użytkownicy mogą ustawić pod zakładką „Konto” w ustawieniach swojego konta.

Parametry

- **name** – Maksimum 20 znaków.
- **url** – Maximum 100 znaków. Będzie poprzedzone „http://” jeżeli jeszcze nie jest.
- **location** – Maximum 30 znaków.

- **description** – Maximum 160 znaków.

Typ zwracany obiekt `User`

9.7 Ulubione metody

`API.`**`favorites`**(*[id][, page]*)
Zwraca ulubione statusy zuwierzytelnionego użytkownika lub użytkownika określonego przez parametr ID.

Parametry

- **id** – ID lub nazwa wyświetlana użytkownika od którego żądane sa ulubione
- **page** – Określa którą stronę wyników otrzymać. Uwaga: istnieją limity stronnicowania.

Typ zwracany lista obiektów `Status`

`API.`**`create_favorite`**(*id*)
Ustawia jako ulubione statusy określone przez parametr ID jako zuwierzytelniony użytkownik.

Parametry **id** – Numeryczne ID statusu.

Typ zwracany obiekt `Status`

`API.`**`destroy_favorite`**(*id*)
Usuwa z ulubionych statusy określone przez parametr ID jako zuwierzytelniony użytkownik.

Parametry **id** – Numeryczne ID statusu.

Typ zwracany obiekt `Status`

9.8 Metody Blokowania

`API.`**`create_block`**(*id/screen_name/user_id*)
Blokuje użytkownika określonego przez parametr ID jako zuwierzytelniony użytkownik. Przerywa znajomość jeżeli taka istniała.

Parametry

- **id** – Określa ID lub nazwę wyświetlaną użytkownika.
- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.

Typ zwracany obiekt `User`

`API.`**`destroy_block`**(*id/screen_name/user_id*)
Odblokowywuje użytkownika określonego przez parametr ID jako zuwierzytelniony użytkownik.

Parametry

- **id** – Określa ID lub nazwę wyświetlaną użytkownika.
- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.

Typ zwracany obiekt `User`

`API.`**`blocks`**(*[page]*)
Zwraca szyk obiektów użytkownika, który blokowany jest przez zuwierzytelnionego użytkownika.

Parametry **`page`** – Określa którą stronę wyników otrzymać. Uwaga: istnieją limity stronnicowania.

Typ zwracany lista obiektów `User`

`API.`**`blocks_ids`**(*[cursor]*)
Zwraca szyk numerycznych ID użytkowników, którzy blokowani są przez zuwierzytelnionego użytkownika.

Parametry **`cursor`** – Dzieli wyniki na strony. Ustaw wartość jako -1 by rozpocząć stronnicowanie. Dostarcz wartości tak jak są zwracane w tekście opowiedzi. Atrybuty next_cursor i previous_cursor używane są do przechoznenia na przód i w tył.

Typ zwracany list of Integers

9.9 Metody Wyciszania

`API.`**`create_mute`**(*id/screen_name/user_id*)
Wycisza użytkownika określonego przez parametr ID dla zuwierzytelnionego użytkownika.

Parametry

- **`id`** – Określa ID lub nazwę wyświetlaną użytkownika.
- **`screen_name`** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **`user_id`** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.

Typ zwracany obiekt `User`

`API.`**`destroy_mute`**(*id/screen_name/user_id*)
Wyłącza wyciszenie użytkownika określonego przez parametr ID dla zuwierzytelnionego użytkownika.

Parametry

- **`id`** – Określa ID lub nazwę wyświetlaną użytkownika.
- **`screen_name`** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **`user_id`** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.

Typ zwracany obiekt `User`

`API.`**`mutes`**(*[cursor][, include_entities][, skip_status]*)
Zwraca szyk obiektów użytkownika, którego wyciszył zuwierzytelniony użytkownik.

Parametry

- **`cursor`** – Dzieli wyniki na strony. Ustaw wartość jako -1 by rozpocząć stronnicowanie. Dostarcz wartości tak jak są zwracane w tekście opowiedzi. Atrybuty next_cursor i previous_cursor używane są do przechoznenia na przód i w tył.
- **`include_entities`** – Ustawione jako false powoduje, że węzeł jednostek nie będzie zawarty. Domyślnie False.

- **skip_status** – Boolean wskazujący czy będą zawarte w zwróconych obiektach user. Domyślnie False.

Typ zwracany lista obiektów `User`

`API.`**`mutes_ids`**(*[cursor]*)
Zwraca szyk numerycznych id użytkowników, których wyciszył zuwierzytelniony użytkownik.

Parametry **cursor** – Dzieli wyniki na strony. Ustaw wartość jako -1 by rozpocząć stronnicowanie. Dostarcz wartości tak jak są zwracane w tekście opowiedzi. Atrybuty next_cursor i previous_cursor używane są do przechoznenia na przód i w tył.

Typ zwracany list of Integers

9.10 Metody Reportowania Spamu

`API.`**`report_spam`**(*id/screen_name/user_id[, perform_block]*)
Użytkownik określony przez ID zostaje zablokowany przez zuwierzytelniongo użytkownika oraz zgłoszony jako spammer.

Parametry

- **id** – Określa ID lub nazwę wyświetlaną użytkownika.
- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **perform_block** – Boolean wskazujący czy zgłoszone konto ma być zablokowane. Domyślnie true.

Typ zwracany obiekt `User`

9.11 Metody Zapisanych wyszukań

`API.`**`saved_searches`**()
Zwraca zapisane zapytania zuwierzytelnionego użytkownika.

Typ zwracany lista obiektów `SavedSearch`

`API.`**`get_saved_search`**(*id*)
Pobiera dane dla zapisanych wyszukiwań należących o zuwierzytelnionego użytkownika, określone przez podane ID.

Parametry **id** – ID zapisanego wyszukania które ma zostać pobrane.

Typ zwracany obiekt `SavedSearch`

`API.`**`create_saved_search`**(*query*)
Tworzy zapisane wyszukani dla zuwierzytelnionego użytkownika.

Parametry **query** – Zapytanie dla wyszukania, które użytkownik chciałby zapisać.

Typ zwracany obiekt `SavedSearch`

API.**destroy_saved_search**(*id*)
Niszczy zapisane wyszukianie dla zuwierzytelnionego użytkownika. Wyszukanie określone przez ID musi należeć do zuwierzytelnionego użytkownika.

Parametry **id** – ID zapisanego wyszukania które ma zostać usunięte.

Typ zwracany obiekt `SavedSearch`

9.12 Metody Pomocy

API.**search**(*q*[, *geocode*][, *lang*][, *locale*][, *result_type*][, *count*][, *until*][, *since_id*][, *max_id*][, *include_entities*])
Zwraca zbiór odpowiednich tweetów pasujących do określonego zapytania.

Proszę miej na uwadze, że usługa wyszukiwania Twittera oraz Search API nie są w założeniu pełnym źródłem tweetów. Nie wszystkie tweety będą zindeksowane lub udostępnione przez wyszukiwarkę.

W API v1.1 format odpowiedzi dla Search API został udoskonalony tak by zwracał obiekty tweetów podobnie jak obiekty, które możesz znaleśść w REST API oraz platformie. Jednakże, atrybuty perspektywiczne (pola które odnoszą się do perspektywy zuwierzytelnionego użytkownika) nie są na tą chwilę wspierane w tym punkcie końcowym.[1][2]

Parametry

- **q** – ciąg znaków zapytania wyszukiwania dla maksimum 500 znaków, wliczając w to operatory. Zapytania mogą być także ograniczone przez ich zawiłość.
- **geocode** – Zwraca tweety w oparciu o lokalizacje użytkownika wewnątrz promienia podanej szerokości/długości geograficznej. Lokacja domyślnie jest pobierana z Geotagging API, lecz zostanie cofnięta do profilu Twitter. Wartość parametru jest określona przez „szerokość,długość,promień" gdzie jednostki promienia muszą być określone jako „mi" (mile) lub „km" (kilometry). Uwaga: nie możesz użyć pobliskiego operatora poprzez API by zgeokodyfikować przypadkowe lokacje; jenakże możesz użyć tego parametru geocode do wyszukania geokodów bezpośrednio. Maksymalnie 1000 różnych „podregionów" będzie brane pod uwagę używajac modyfikatora promienia.
- **lang** – Ogranicza tweey to podanego języka, nadanego przez kod ISO 639-1. Detekcja języka jest best-effort.
- **locale** – Określa język zapytania, które będzie wysłane (na tę chwię tylko „ja"). Jesto skierowane do konsumentów posiadających szczególne wymagania językowe. Domyślnie działa w większości przypadków.
- **result_type** – Określa jaki typ wyników wyszukiwania chciałbyś otrzymywać. Domyślnie „mixed". Poprawne wartości zawierają: * mixed : zawiera wyniki popularne oraz w czasie rzeczywistym *recent : zwraca tylko najnowsze wyniki * popular : zwraca tylko popularne wyniki
- **count** – Liczba wyników do pobrania na stronę.
- **until** – Zwraca tweety stworzone przed określoną datą. Data powinna być w formacie YYYY-MM-DD. Miej na uwadze, że indeks wyszukiwania ma 7-dniowy limit. Innymi słowami, nie zostaną znalezione żadne tweety starsze niż tydzień.

[1] https://web.archive.org/web/20170829051949/https://dev.twitter.com/rest/reference/get/search/tweets
[2] https://twittercommunity.com/t/favorited-reports-as-false-even-if-status-is-already-favorited-by-the-user/11145

- **since_id** – Zwraca tylko statusy z ID większym (tzn. nowszym) niż określone ID. Liczba tweetów, które są dostępne w API jst limitowana. Jeżeli limit tweetów wydarzył się od since_id to since_id będzie najstarzym dostępnym ID.
- **max_id** – Zwraca tylko statusy z ID mniejszym (tzn. starszym) lub równym określonemu ID.
- **include_entities** – Ustawione jako false powoduje, że węzeł jednostek nie będzie zawarty. Domyślnie False.

Typ zwracany obiekt `SearchResults`

9.13 Metoda Listy

`API.`**`create_list`**(*name*[, *mode*][, *description*])
Tworzy nową listę dla zuwierzytelnionego użytkownia. Miej na uwadzę, że możesz stworzyć maksymalnie 1000 list dla jednego konta.

Parametry

- **name** – Nazwa nowej listy.
- **mode** – Czy lista jest publiczna czy prywatna. Wartości mogą być publiczne i prwyatne. Domyślnie listy są publiczne jeżeli nie jest ustawiony żaden tryb.
- **description** – Opis listy, którą tworzysz.

Typ zwracany obiekt `List`

`API.`**`destroy_list`**([*owner_screen_name/owner_id*], *list_id/slug*)
Usuwa określoną listę. Zuwierzytelniony użytkownik musi być właścicielem listy by ją usunąć.

Parametry

- **owner_screen_name** – Nazwa wyświetlana użytkownika, który jest właścicielem listy żądanej przez żeton.
- **owner_id** – ID użytkownika, który jest właścicielem listy żadanej przez żeton.
- **list_id** – Numeryczna lista ID.
- **slug** – Możesz zidentyfikować listę używając jej żetona zamiast numerycznego ID. Jeżeli się na to zdecydujesz to będziesz musiał określić właściciela listy używając parametrów owner_id lub owner_screen_name.

Typ zwracany obiekt `List`

`API.`**`update_list`**(*list_id/slug*[, *name*][, *mode*][, *description*][, *owner_screen_name/owner_id*])
Aktualizuje wybraną listę. Zuwierzytelniony użytkownik musi być właścicielem listy by ją zaktualizować.

Parametry

- **list_id** – Numeryczna lista ID.
- **slug** – Możesz zidentyfikować listę używając jej żetona zamiast numerycznego ID. Jeżeli się na to zdecydujesz to będziesz musiał określić właściciela listy używając parametrów owner_id lub owner_screen_name.
- **name** – Nazwa listy.
- **mode** – Czy lista jest publiczna czy prywatna. Wartości mogą być publiczne i prwyatne. Domyślnie listy są publiczne jeżeli nie jest ustawiony żaden tryb.

- **description** – Opis nadany liście.
- **owner_screen_name** – Nazwa wyświetlana użytkownika, który jest właścicielem listy żądanej przez żeton.
- **owner_id** – ID użytkownika, który jest właścicielem listy żadanej przez żeton.

Typ zwracany obiekt `List`

API.**lists_all**(*[screen_name][, user_id][, reverse]*)

Zwraca wszystkie listy, które subskrybuje zuwierzytelniony lub określony użytkownik, w tym ich własne. Użytkownik określony jest poprzez parametry `user_id` lub `screen name`. Jeżeli użytkownik nie jest podany to zostanie wtedy użyty zuwierzytelniony użytkownik.

Maksymalnie 100 wyników może być zwrócone przez to wywołanie. Listy subskrybowane są zwrócone jako pirwsze, tuż za nimi listy posiadane na własność. To oznacza, że jeżeli użytkownik subsrybuje 90 list i posiada 20 list to ta metoa zwróci 90 list subskrybowanych i 10 list posiadanych. Metoda `reverse` zwraca posiadane listy jako pierwsze, tak więc z ustawieniem `reverse=true` zwrócone będzie 20 list posiadanych i 80 subskrybowanych.

Parametry

- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **reverse** – Boolean wskazujący czy chiałbyś by posiadane na własność listy zostały zwrócone jako pierwsze. Zobacz opis powyżej po więcej informacji na temat tego parametru.

Typ zwracany lista obiektów `List`

API.**lists_memberships**(*[screen_name][, user_id][, filter_to_owned_lists][, cursor][, count]*)

Zwraca listy, do których został dodany wybrany użytkownik. Jeżeli `user_id` lub `screen_name` nie są podane to zwracane jest członkostwo dla zuwierzytelnionego użytkownika.

Parametry

- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **filter_to_owned_lists** – Boolean wskazujący czy zwrócić tylko listy, które posiada zuwierzytelniony użytkownik oraz użytkownik reprezentowany przez `user_id` lub `screen_name`.
- **cursor** – Dzieli wyniki na strony. Ustaw wartość jako -1 by rozpocząć stronnicowanie. Dostarcz wartości tak jak są zwracane w tekście opowiedzi. Atrybuty next_cursor i previous_cursor używane są do przechoznenia na przód i w tył.
- **count** – Liczba wyników do pobrania na stronę.

Typ zwracany lista obiektów `List`

API.**lists_subscriptions**(*[screen_name][, user_id][, cursor][, count]*)

Zdobywa kolekcję list do którye subskrybuje wybrany użytkownnik, domyślnie 20 list na jedną stronę. Nie zawiera list posiadanych przez użytkownika.

Parametry

- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **cursor** – Dzieli wyniki na strony. Ustaw wartość jako -1 by rozpocząć stronnicowanie. Dostarcz wartości tak jak są zwracane w tekście opowiedzi. Atrybuty next_cursor i previous_cursor używane są do przechoznenia na przód i w tył.
- **count** – Liczba wyników do pobrania na stronę.

Typ zwracany lista obiektów `List`

API.**list_timeline**(*list_id/slug*[, *owner_id/owner_screen_name*][, *since_id*][, *max_id*][, *count*][, *include_entities*][, *include_rts*])

Zwraca oś czasu tweetów, których autorami są członkowie wybranej listy. Domyślnie zawiera retweety. Użyj parametru `include_rts=false` by pominąć retweety.

Parametry

- **list_id** – Numeryczna lista ID.
- **slug** – Możesz zidentyfikować listę używając jej żetona zamiast numerycznego ID. Jeżeli się na to zdecydujesz to będziesz musiał określić właściciela listy używając parametrów owner_id lub owner_screen_name.
- **owner_id** – ID użytkownika, który jest właścicielem listy żadanej przez żeton.
- **owner_screen_name** – Nazwa wyświetlana użytkownika, który jest właścicielem listy żądanej przez żeton.
- **since_id** – Zwraca tylko statusy z ID większym (tzn. nowszym) niż określone ID.
- **max_id** – Zwraca tylko statusy z ID mniejszym (tzn. starszym) lub równym określonemu ID.
- **count** – Liczba wyników do pobrania na stronę.
- **include_entities** – Ustawione jako false powoduje, że węzeł jednostek nie będzie zawarty. Domyślnie False.
- **include_rts** – Boolean wskazujący czy oś czasu listy będzie zawierać natywne retweety (jeżeli istnieją) w dodatku do standardowego strumienia tweetów. Format wyjścia reteetowanych tweetów jest identyczny do tego widocznego w home_timeline.

Typ zwracany lista obiektów `Status`

API.**get_list**(*list_id/slug*[, *owner_id/owner_screen_name*])

Zwraca wybraną listę. Prywate listy będą pokazane tylko jeżeli zuwierzytelniony użytkownik jest ich właścicielem.

Parametry

- **list_id** – Numeryczna lista ID.
- **slug** – Możesz zidentyfikować listę używając jej żetona zamiast numerycznego ID. Jeżeli się na to zdecydujesz to będziesz musiał określić właściciela listy używając parametrów owner_id lub owner_screen_name.
- **owner_id** – ID użytkownika, który jest właścicielem listy żadanej przez żeton.
- **owner_screen_name** – Nazwa wyświetlana użytkownika, który jest właścicielem listy żądanej przez żeton.

Typ zwracany obiekt `List`

API.**add_list_member**(*list_id/slug*, *screen_name/user_id*[, *owner_id/owner_screen_name*])
Dodaje nowego członka do listy. Zuwierzytelniony użytkownik musi być właścicielem listy by dodawać do niej członków. Lista może zawierać maksymalnie 5000 członków.

Parametry

- **list_id** – Numeryczna lista ID.
- **slug** – Możesz zidentyfikować listę używając jej żetona zamiast numerycznego ID. Jeżeli się na to zdecydujesz to będziesz musiał określić właściciela listy używając parametrów owner_id lub owner_screen_name.
- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **owner_id** – ID użytkownika, który jest właścicielem listy żadanej przez żeton.
- **owner_screen_name** – Nazwa wyświetlana użytkownika, który jest właścicielem listy żądanej przez żeton.

Typ zwracany obiekt `List`

API.**add_list_members**(*list_id/slug*, *screen_name/user_id*[, *owner_id/owner_screen_name*])
Dodaje członków do listy, maksymalnie 100. Zuwierzytelniony użytkownik musi być właścicielem listy by dodawać do niej członków. Lista może zawierać maksymalnie 5000 członków.

Parametry

- **list_id** – Numeryczna lista ID.
- **slug** – Możesz zidentyfikować listę używając jej żetona zamiast numerycznego ID. Jeżeli się na to zdecydujesz to będziesz musiał określić właściciela listy używając parametrów owner_id lub owner_screen_name.
- **screen_name** – Oddzielona przecinkami lista nazw, maksymalnie 100 na jedno żądanie.
- **user_id** – Oddzielona przecinkami lista ID użytkowników, maksymalnie 100 na jedno żądanie.
- **owner_id** – ID użytkownika, który jest właścicielem listy żadanej przez żeton.
- **owner_screen_name** – Nazwa wyświetlana użytkownika, który jest właścicielem listy żądanej przez żeton.

Typ zwracany obiekt `List`

API.**remove_list_member**(*list_id/slug*, *screen_name/user_id*[, *owner_id/owner_screen_name*])
Usuwa wybranego użytkownika z listy. Zuwierzytelniony użytkownik musi być właścicielem listy by usuwać z niej użytkowników.

Parametry

- **list_id** – Numeryczna lista ID.
- **slug** – Możesz zidentyfikować listę używając jej żetona zamiast numerycznego ID. Jeżeli się na to zdecydujesz to będziesz musiał określić właściciela listy używając parametrów owner_id lub owner_screen_name.
- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.

- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **owner_id** – ID użytkownika, który jest właścicielem listy żadanej przez żeton.
- **owner_screen_name** – Nazwa wyświetlana użytkownika, który jest właścicielem listy żądanej przez żeton.

Typ zwracany obiekt `List`

API.**remove_list_members**(*list_id/slug*, *screen_name/user_id*[, *owner_id/owner_screen_name*])
Usuwa użytkowników z listy, maksymalnie 100. Zuwierzytelniony użytkownik musi być właścicielem listy by usuwać z niej użytkowników. Listy mogą zawierać maksymalnie 5000 użytkowników.

Parametry

- **list_id** – Numeryczna lista ID.
- **slug** – Możesz zidentyfikować listę używając jej żetona zamiast numerycznego ID. Jeżeli się na to zdecydujesz to będziesz musiał określić właściciela listy używając parametrów owner_id lub owner_screen_name.
- **screen_name** – Oddzielona przecinkami lista nazw, maksymalnie 100 na jedno żądanie.
- **user_id** – Oddzielona przecinkami lista ID użytkowników, maksymalnie 100 na jedno żądanie.
- **owner_id** – ID użytkownika, który jest właścicielem listy żadanej przez żeton.
- **owner_screen_name** – Nazwa wyświetlana użytkownika, który jest właścicielem listy żądanej przez żeton.

Typ zwracany obiekt `List`

API.**list_members**(*list_id/slug*[, *owner_id/owner_screen_name*][, *cursor*])
Zwraca użytkowników z określonej listy.

Parametry

- **list_id** – Numeryczna lista ID.
- **slug** – Możesz zidentyfikować listę używając jej żetona zamiast numerycznego ID. Jeżeli się na to zdecydujesz to będziesz musiał określić właściciela listy używając parametrów owner_id lub owner_screen_name.
- **owner_id** – ID użytkownika, który jest właścicielem listy żadanej przez żeton.
- **owner_screen_name** – Nazwa wyświetlana użytkownika, który jest właścicielem listy żądanej przez żeton.
- **cursor** – Dzieli wyniki na strony. Ustaw wartość jako -1 by rozpocząć stronnicowanie. Dostarcz wartości tak jak są zwracane w tekście opowiedzi. Atrybuty next_cursor i previous_cursor używane są do przechoznenia na przód i w tył.

Typ zwracany lista obiektów `User`

API.**show_list_member**(*list_id/slug*, *screen_name/user_id*[, *owner_id/owner_screen_name*])
Sprawdza czy określony użytkownik jest członkiem określonej listy.

Parametry

- **list_id** – Numeryczna lista ID.
- **slug** – Możesz zidentyfikować listę używając jej żetona zamiast numerycznego ID. Jeżeli się na to zdecydujesz to będziesz musiał określić właściciela listy używając parametrów owner_id lub owner_screen_name.

- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **owner_id** – ID użytkownika, który jest właścicielem listy żadanej przez żeton.
- **owner_screen_name** – Nazwa wyświetlana użytkownika, który jest właścicielem listy żądanej przez żeton.

Typ zwracany obiekt `User` jeżeli użytkownik jest członkiem listy

`API.`**`subscribe_list`**(*list_id/slug*[, *owner_id/owner_screen_name*])
Subskrybuje zuwierzytelnionego użytkownika do określonej listy.

Parametry

- **list_id** – Numeryczna lista ID.
- **slug** – Możesz zidentyfikować listę używając jej żetona zamiast numerycznego ID. Jeżeli się na to zdecydujesz to będziesz musiał określić właściciela listy używając parametrów owner_id lub owner_screen_name.
- **owner_id** – ID użytkownika, który jest właścicielem listy żadanej przez żeton.
- **owner_screen_name** – Nazwa wyświetlana użytkownika, który jest właścicielem listy żądanej przez żeton.

Typ zwracany obiekt `List`

`API.`**`unsubscribe_list`**(*list_id/slug*[, *owner_id/owner_screen_name*])
Anuluje subskrypcję zuwierzytelnionego użytkownika do określonej listy.

Parametry

- **list_id** – Numeryczna lista ID.
- **slug** – Możesz zidentyfikować listę używając jej żetona zamiast numerycznego ID. Jeżeli się na to zdecydujesz to będziesz musiał określić właściciela listy używając parametrów owner_id lub owner_screen_name.
- **owner_id** – ID użytkownika, który jest właścicielem listy żadanej przez żeton.
- **owner_screen_name** – Nazwa wyświetlana użytkownika, który jest właścicielem listy żądanej przez żeton.

Typ zwracany obiekt `List`

`API.`**`list_subscribers`**(*list_id/slug*[, *owner_id/owner_screen_name*][, *cursor*][, *count*][, *include_entities*][, *skip_status*])
Zwraca subskrybentów wybranej listy. Subskrybenci prywatnych list będą wyświetleni tylko jeżeli zuwierzytelniony użytkownik jest właścicielem listy.

Parametry

- **list_id** – Numeryczna lista ID.
- **slug** – Możesz zidentyfikować listę używając jej żetona zamiast numerycznego ID. Jeżeli się na to zdecydujesz to będziesz musiał określić właściciela listy używając parametrów owner_id lub owner_screen_name.
- **owner_id** – ID użytkownika, który jest właścicielem listy żadanej przez żeton.
- **owner_screen_name** – Nazwa wyświetlana użytkownika, który jest właścicielem listy żądanej przez żeton.

- **cursor** – Dzieli wyniki na strony. Ustaw wartość jako -1 by rozpocząć stronnicowanie. Dostarcz wartości tak jak są zwracane w tekście opowiedzi. Atrybuty next_cursor i previous_cursor używane są do przechoznenia na przód i w tył.
- **count** – Liczba wyników do pobrania na stronę.
- **include_entities** – Ustawione jako false powoduje, że węzeł jednostek nie będzie zawarty. Domyślnie False.
- **skip_status** – Boolean wskazujący czy będą zawarte w zwróconych obiektach user. Domyślnie False.

Typ zwracany lista obiektów `User`

`API.`**show_list_subscriber**(*list_id/slug*, *screen_name/user_id*[, *owner_id/owner_screen_name*])
Sprawdza czy określony użytkownik jest subskrybentem określonej listy.

Parametry

- **list_id** – Numeryczna lista ID.
- **slug** – Możesz zidentyfikować listę używając jej żetona zamiast numerycznego ID. Jeżeli się na to zdecydujesz to będziesz musiał określić właściciela listy używając parametrów owner_id lub owner_screen_name.
- **screen_name** – Określa nazwę wyświetlaną użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **user_id** – Określa ID użytkownika. Przydatne do rozróżnienia czy nazwa wyświetlana jest taka sama jak ID użytkownika.
- **owner_id** – ID użytkownika, który jest właścicielem listy żadanej przez żeton.
- **owner_screen_name** – Nazwa wyświetlana użytkownika, który jest właścicielem listy żądanej przez żeton.

Typ zwracany obiekt `User` jeżeli użytkownik subskrybuje listę

9.14 Metody Trendów

`API.`**trends_available**()
Zwraca lokację, dla której Twitter posiada informacje o trendujących tematach. Odpowiedź będzie w formie szyku ‘”locations”, który szyfruje WOEID (Yahoo! Where On Earth ID) lokacji i inne odczytywalne przez człowieka informacje takie jak autentyczna nazwa oraz państwo, w którym znajduje się lokacja.

Typ zwracany obiekt `JSON`

`API.`**trends_place**(*id*[, *exclude*])
Zwraca top 50 trendujących tematów dla wybranego WOEID jeżeli są dla nich dostępne informacje.

Ta odpowiedź jest szykiem obiektów „trend” które szyfrują nazwę trendującego tematu, parametr zapytania, który może być użyty do wyszukania tematu na Twitter Search a także Twitter Search URL.

Ta informacja zostaje zmagazynowana na 5 minut. Zażądanie większej częstotliwości nie zwróci większej ilości danych i nie będzie liczyć się jako wspólczynnik limitu używania.

tweet_volume dla ostatnich 24 godzin jest także zwracane dla wielu trendów jeżeli jest to możliwe.

Parametry

- **id** – ID Yahoo! Where On Earth lokacji, dla której mają być zwrócone informacje o trenach. Globalne informacje są dostępne używając 1 jako WOEID.

- **exclude** – Ustawienie tego jako jednakowe z hashtagami, spowoduje usunięcie hashtagów z listy trendów.

Typ zwracany obiekt JSON

API.**trends_closest**(*lat*, *long*)
Zwraca lokację, dla której Twitter ma trendujące tematy najbardziej zbliżone do określonej lokacji.

Ta odpowiedź jest szykiem „locations" który szyfruje WOEID lokacji oraz inne odczytywalne przz człowieka informacje takie jak autentyczna nazwa oraz państwo w którym lokacja się znajduje.

WOEID jest to Yahoo! Where On Earth ID.

Parametry

- **lat** – Jeżeli podana jest długość gograficzna to dostępne trendy lokacji będa ustawione wg. dystansu od najbliższej do najdalszej, w parach co-ordinate. Poprawne wartości długości gograficznej to wyłącznie wartości od -180.0 do +180.0 (zachód to wartość negatywna, wschód to pozytywna).
- **long** – Jeżeli podana jest szerokość gograficzna to dostępne trendy lokacji będa ustawione wg. dystansu od najbliższej do najdalszej, w parach co-ordinate. Poprawne wartości długości gograficznej to wyłącznie wartości od -180.0 do +180.0 (zachód to wartość negatywna, wschód to pozytywna).

Typ zwracany obiekt JSON

9.15 Metody Geo

API.**reverse_geocode**([*lat*][, *long*][, *accuracy*][, *granularity*][, *max_results*])
Posiadając szerokość i długość geograficzną, wyszukiwane będą miejsca (miasta i dzielnice), których ID mogą być określone w wywołaniu dla *update_status()* tak by wyglądały jako nazwa lokacji. To wywołanie dostarcza szczegółowe informacje na temat lokacji; funkcja nearby_places() powinna być używana do zdobywania list miejscw okolicy, bez szczegółowych informacji.

Parametry

- **lat** – Szerokość geograficzna lokacji.
- **long** – Długość geograficzna lokacji.
- **accuracy** – Określa „region" do wyszukiwania, między innymi numer (wtedy jest to zasięg w metrach, ale może być to też ciąg znaków zakończonych jako stopy). Jeżeli nie jest to przekazane, założony będzie zasięg 0m.
- **granularity** – Assumed to be neighborhood by default; can also be city.
- **max_results** – Wskazówka co do maksymalnej liczby rezultatów, które zostaną zwrócone. Jest to tylko wskazanie, nie musisz się do niego stosować.

API.**geo_id**(*id*)
Posiadając *id* lokacji podaje więcej informacji na jej temat.

Parametry **id** – Poprawny ID Twittera lokacji.

9.16 Metody Użyteczności

API.**configuration**()
Zwraca aktualną konfigurację używaną przez Twitter, w tym żetony twitter.cm, które nie są nazwami użytkowników, maksymalne rozmiary zdjęcia oraz długość skróconego URL t.co. Zaleca się by aplikacje żądały tego punktu końcowego gdy są obłaowane, jednak nie więcej niż raz dziennie.

9.17 Metody Mediów

API.**media_upload**(*filename*[, *file*])
Użyj tego punktu końcowego by wrzucić obraz na Twitter.

Parametry

- **filename** – Nazwa pliku obrazu do wrzucenia. Jest automatycznie otwarte, chyba, że `file` jest określone.
- **file** – Obiekt pliku, który musi być użyty zamiast otwierania `filename`. `filename` jest nadal wymagane dla detekcji typu MIME oraz do użytwania pól formy w danych POST.

Typ zwracany obiekt `Media`

API.**create_media_metadata**(*media_id*, *alt_text*)
Ten punkt końcowy może być użyty do dostarczenia dodatkowych informacji na temat wrzuconego media_id. Ta funkcja na tę chwilę jest wspierana tylko przez obrazy i GIFy. Wywołaj ten punkt końcowy by dodać dodatkowe metadane takie jak alt text.

Parametry

- **media_id** – ID mediów, do których dodany jest alt text.
- **alt_text** – Alt tekst który zostanie dodany do obrazu.

ROZDZIAŁ 10

tweepy.error — Wyjątki

Wyjątki są dostępne bezpośrednio w module tweepy co oznacza, że tweepy.error sam w sobie nie musi zostać zimportowany. Na przykład: tweepy.error.TweepError jest dostępny jako tweepy.TweepError.

exception TweepError

Główny wyjątek używany przez Tweepy. Pojawia się w różnych przypadkach.

Gdy podniesiony jest TweepError z powodu błedu, którym odpowiedział Twitter to kod błedu (tak jak jest to określone w dokumentacji API) może być znaleziony w TweepError.response.text.. Uwaga: TweepErrors może być także podniesiony wraz z innymi rzeczami jako wiadomość (na przykład jako prosty błąd ciągów znaków powodu).

exception RateLimitError

Jest podniesione gdy metoda API nie działa ze względu na osiągnięcie limitu współczynników określonej przez Twitter. Ułatwia to obsługę limitu współczynników.

Dziedziczy od *TweepError* więc except TweepError też złapie RateLimitError.

ROZDZIAŁ 11

Uruchamianie testów

Te kroki wyjaśniają jak uruchamiać testy dla Tweepy:

1. Pobierz kod źródłowy Tweepy do katalogu.
2. Zainstaluj z pobranego źródła z `test` extra, np. `pip install .[test]`. Opcjonalnie możesz zainstalować też `dev` extra, dla `tox` i `coverage` np. `pip install .[dev,test]`.
3. Uruchom `python setup.py nosetests` lub po prostu `nosetests` w katalogu źródłowym. Z `dev` extra będzie wyświetlone sprawozdanie a `tox` może być także użyty do uruchomienia testu z inną wersją Python.

By nagrywać nowe kasety możesz użyć niżej wymienionych zmiennych środowiskowych:

`TWITTER_USERNAME` `CONSUMER_KEY` `CONSUMER_SECRET` `ACCESS_KEY` `ACCESS_SECRET` `USE_REPLAY`

Po prostu ustaw `USE_REPLAY` jako `FALSE` i dostarcz aplikacji dane logowania oraz nazwę użytkownika.

ROZDZIAŁ 12

Indeksy i tabele

- genindex
- search

Indeks

T

U

V