

USER MANUAL
Getwit Python Package

Disusun oleh:
Ahmad Afiq Fitrah
1301184426



Program Studi Sarjana Informatika
Fakultas Informatika
Universitas Telkom
Bandung
2021

Daftar Isi

Daftar Isi	i
Daftar Gambar	i
1. Deskripsi	1
2. Cara Penggunaan	1
2.1 Mengunduh library	1
2.2 Generate key dan token	2
1.3 File main.py	2
1.4 File function.py	2
1.5 File model.py	9
3. Source Code	9
3.1 api_secret.py	9
3.2 model.py	9
3.3 function.py	19

Daftar Gambar

Gambar 1 perintah instal library pada terminal text editor	1
Gambar 2 tampilan api_secret.py	2
Gambar 3 tampilan main.py (contoh program)	2
Gambar 4 tampilan setelah sebuah fungsi dengan OAuth1.0 dieksekusi	6
Gambar 5 tampilan untuk meminta izin pada aplikasi yang terhubung dengan callback_url	7
Gambar 6 tampilan setelah izin diberikan dan oauth_verifier pada url halaman	7
Gambar 7 tampilan setelah menyalin oauth_verifier	8
Gambar 8 tampilan hasil akuisisi dan fungsi dengan OAuth1.0	8
Gambar 9 tampilan hasil akuisisi dari fungsi dengan OAuth2.0 App-Only	8


1. Deskripsi

Getwit python package adalah library yang terhubung dengan endpoint dengan metode 'GET' pada Twitter API v2. Library ini dibangun menggunakan Bahasa pemrograman python. Sejauh ini getwit memiliki 3 module utama yaitu function.py, model.py dan api_secret.py. Selain itu, getwit terhubung di 34 endpoint yang ada di Twitter API v2. Library dijalankan melalui sebuah module main.py diluar folder package dan memanggil tiap fungsi(endpoint) yang diinginkan. Sebelum menggunakan library tersebut, pastikan telah melakukan generate key dan token serta mengatur user authentication setting di laman developer portal dari website Twitter Developer. Seluruh endpoint yang ada telah diuji coba menggunakan dua tipe autentikasi, yaitu OAuth1.0 User-context dan OAuth2.0 App-Only dengan level akses pada developer portal ialah 'elevated'. Beberapa endpoint pada Twitter API memerlukan jenis autentikasi dan level akses tertentu dan untuk endpoint yang belum tersedia akan kami kembangkan segera.

2. Cara Penggunaan

2.1 Mengunduh library

Untuk mengunduh library, cukup mengakses di repository 'getwit package' di akun Github 'sonspyscode' dan melakukan pengunduhan. Selain itu, untuk mengunduh library tersebut dapat dilakukan melalui terminal dari text editor dengan perintah 'pip install getwit'. Pastikan telah mengunduh setiap requirement yang dibutuhkan pada file dengan nama requirement.txt atau eksekusi file python dengan nama setup.py.

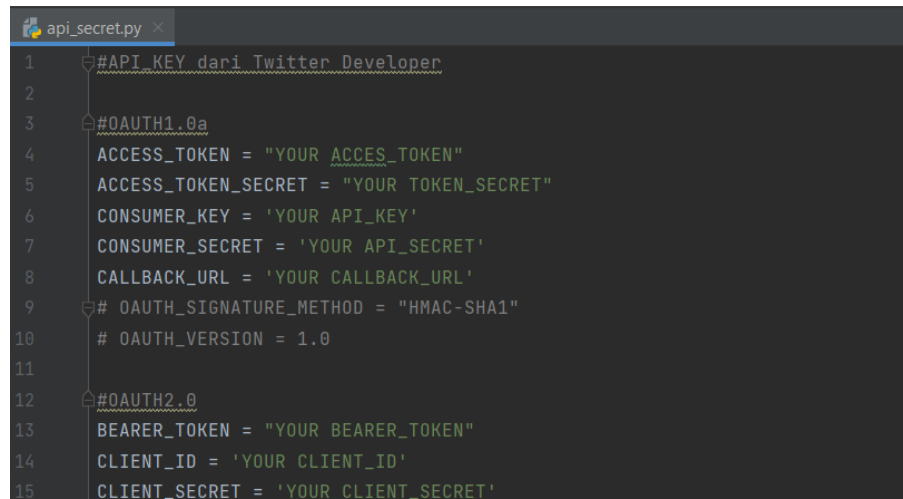


```
C:\Users\Ahmad\PycharmProjects\Aptwit> pip install getwit
```

Gambar 1 perintah instal library pada terminal text editor

2.2 Generate key dan token

Sebelum mengimplementasikan getwit, pastikan telah memiliki akun developer pada website Twitter Developer. Setelah itu, generate key dan token yang ada pada



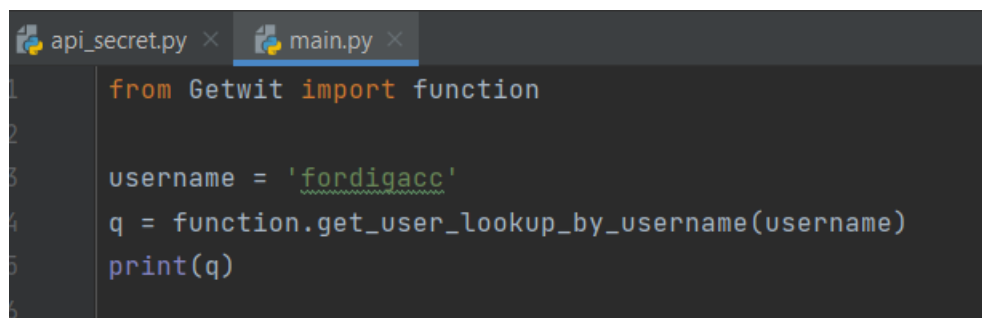
```
1 #API_KEY dari Twitter Developer
2
3 #OAUTH1.0a
4 ACCESS_TOKEN = "YOUR ACCES_TOKEN"
5 ACCESS_TOKEN_SECRET = "YOUR TOKEN_SECRET"
6 CONSUMER_KEY = 'YOUR API_KEY'
7 CONSUMER_SECRET = 'YOUR API_SECRET'
8 CALLBACK_URL = 'YOUR CALLBACK_URL'
9 # OAUTH_SIGNATURE_METHOD = "HMAC-SHA1"
10 # OAUTH_VERSION = 1.0
11
12 #OAUTH2.0
13 BEARER_TOKEN = "YOUR BEARER_TOKEN"
14 CLIENT_ID = 'YOUR CLIENT_ID'
15 CLIENT_SECRET = 'YOUR CLIENT_SECRET'
```

Gambar 2 tampilan api_secret.py

projects & apps. Key dan token tersebut disalin dan simpan pada file dengan nama api_secret.api. Perhatikan bahwa status dari aplikasi dan User Autentication Setting mempengaruhi jenis key dan token yang dapat di generate. Untuk detailnya silahkan akses <https://developer.twitter.com/en/docs/apps/overview>.

1.3 File main.py

Main.py dibuat diluar package untuk memanggil setiap fungsi yang diinginkan. Sebelum itu, lakukan import library getwit dengan perintah 'import getwit' atau lebih spesifik dengan perintah 'from getwit import function'. Selanjutnya, buat perintah untuk memanggil fungsi(endpoint) yang diinginkan.



```
1 from Getwit import function
2
3 username = 'fordigacc'
4 q = function.get_user_lookup_by_username(username)
5 print(q)
```

Gambar 3 tampilan main.py (contoh program)

1.4 File function.py

Function.py berisi fungsi yang terhubung dengan endpoint yang ada pada Twitter API v2 dengan jenis autentikasi beragam. Ada 34 endpoint yang dapat digunakan untuk

mendapatkan data atau informasi seperti bukti digital potensial. Berikut endpoint yang dimaksud,

1.4.1 Tweet lookup

a. `get_tweet_lookup`

Mengembalikan beragam informasi tentang spesifik tweet berdasarkan id atau daftar id.

b. `get_tweet_by`

Mengembalikan beragam informasi tentang spesifik tweet berdasarkan sebuah id.

1.4.2 Likes

a. `get_liked_tweets`

Mendapatkan informasi mengenai tweet yang disukai pengguna.

b. `get_likes_lookup`

Mendapatkan informasi mengenai pengguna yang menyukai sebuah tweet tertentu.

1.4.3 Timeline

a. `get_user_timeline`

Mengembalikan Tweet yang dibuat oleh satu pengguna, ditentukan oleh ID pengguna yang diminta. Secara default, sepuluh Tweet terbaru dikembalikan per permintaan. Menggunakan pagination, 3.200 Tweet terbaru dapat diambil.

b. `get_user_mention_timeline`

Mengembalikan Tweet yang menyebutkan satu pengguna yang ditentukan oleh ID pengguna yang diminta. Secara default, sepuluh Tweet terbaru dikembalikan per permintaan. Menggunakan pagination, hingga 800 Tweet terbaru dapat diambil.

1.4.4 Users

a. `get_user_lookup_by`

Mengembalikan berbagai informasi tentang satu atau lebih pengguna yang ditentukan oleh username mereka.

b. `get_my_profile`

Mengembalikan informasi tentang pengguna yang telah ter authorization.

c. `get_user_lookup_by_username`

Mengembalikan berbagai informasi tentang satu atau lebih pengguna yang ditentukan oleh nama pengguna mereka.

d. `get_users_lookup`

Mengembalikan berbagai informasi tentang satu atau lebih pengguna yang ditentukan oleh ID yang diminta.

e. `get_user_lookup_by_id`

Mengembalikan berbagai informasi tentang satu pengguna yang ditentukan oleh ID yang diminta.

1.4.5 Spaces

a. `get_space_lookup`

Mengembalikan berbagai informasi tentang satu pengguna yang ditentukan oleh ID yang diminta.

b. `get_search_space`

Kembalikan Spaces langsung atau terjadwal yang cocok dengan istilah pencarian yang Anda tentukan. Titik akhir ini melakukan pencarian kata kunci, yang berarti akan mengembalikan Spasi yang sama persis dengan huruf besar-kecil yang cocok dengan istilah pencarian yang ditentukan. Istilah pencarian akan cocok dengan judul asli Space.

c. `get_space`

Mengembalikan detail tentang beberapa Spasi. Hingga 100 ID Spasi yang dipisahkan koma dapat dicari menggunakan titik akhir ini.

d. `get_spaces_by_creator_ids`

Mengembalikan Spaces langsung atau terjadwal yang dibuat oleh ID pengguna yang ditentukan. Hingga 100 ID yang dipisahkan koma dapat dicari menggunakan titik akhir ini.

1.4.6 Volume Streams

a. `get_sample_stream`

Mendapatkan sekitar 1% dari semua Tweet secara real-time. Jika Anda memiliki akses '*Academic Research*', Anda dapat menghubungkan hingga dua koneksi redundan untuk memaksimalkan waktu streaming Anda.

1.4.7 Retweets

a. `get_retweets_lookup`

Memungkinkan Anda mendapatkan informasi tentang siapa yang telah me-Retweet Tweet.

1.4.8 Search Tweets

a. `get_recent_tweet`

End point ini mengembalikan Tweet dari tujuh hari terakhir yang cocok dengan permintaan pencarian.

1.4.9 Tweet Count

a. `get_recent_tweet_count`

End point ini mengembalikan jumlah Tweet dari tujuh hari terakhir yang cocok dengan `str_query` yang dimasukkan.

1.4.10 Quote Tweets

a. `get_quote_tweet`

Mengembalikan Tweet Kutipan untuk Tweet yang ditentukan oleh ID Tweet yang diminta

1.4.11 Mutes

a. `get_mutes_lookup`

Mengembalikan daftar pengguna yang dibisukan oleh ID pengguna yang ditentukan.

1.4.12 Lists Lookup

a. `get_user_owned_list`

Mengembalikan semua Daftar yang dimiliki oleh pengguna yang ditentukan.

b. `get_list_lookup`

Mengembalikan detail dari Daftar yang ditentukan.

1.4.13 List Tweets Lookup

a. `get_list_tweets`

Mengembalikan daftar Tweet dari Daftar yang ditentukan.

1.4.14 Pinned Lists

a. `get_pinned_list`

Mengembalikan Daftar yang disematkan oleh pengguna tertentu.

1.4.15 List Member

a. `get_user_list_membership`

Mengembalikan semua List yang menjadi anggota pengguna tertentu.

b. `get_user_list_member`

Mengembalikan daftar pengguna yang menjadi anggota List yang ditentukan.

1.4.16 List Follows

a. `get_user_list_followed`

Mengembalikan daftar pengguna yang diikuti dari List yang ditentukan.

b. `get_list_followers`

Mengembalikan daftar pengguna yang merupakan pengikut Daftar yang ditentukan.

1.4.17 Follows

a. `get_following_lookup`

Mengembalikan daftar pengguna yang diikuti oleh ID pengguna tertentu.

b. `get_followers_lookup`

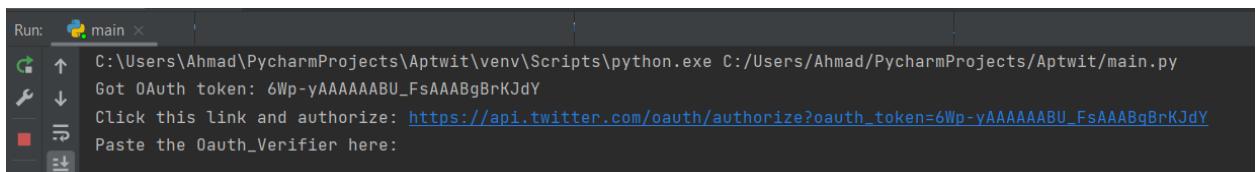
Mengembalikan daftar pengguna yang merupakan pengikut ID pengguna yang ditentukan.

1.4.18 Blocks

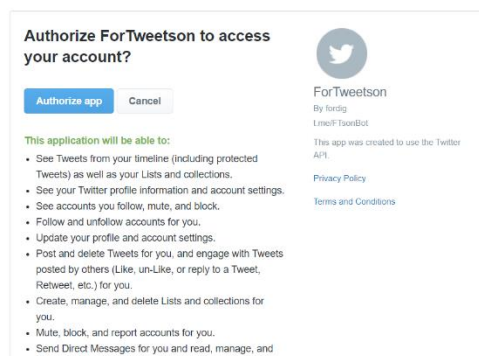
a. `get_block_lookup`

Mengembalikan daftar pengguna yang diblokir oleh ID pengguna yang ditentukan.

Ada dua jenis autentikasi yang dilakukan, pertama autentikasi OAuth1.0a yang memerlukan 'consumer_key' dan 'consumer_secret'. Disaat fungsi(endpoint) yang memerlukan OAuth1.0a dipanggil, maka akan otomatis menjalankan proses autentikasi tersebut. Pada OAuth1.0a setelah program dieksekusi, akan diminta untuk mengakses sebuah link autentikasi untuk meminta izin pada aplikasi yang terhubung dengan akun developer.

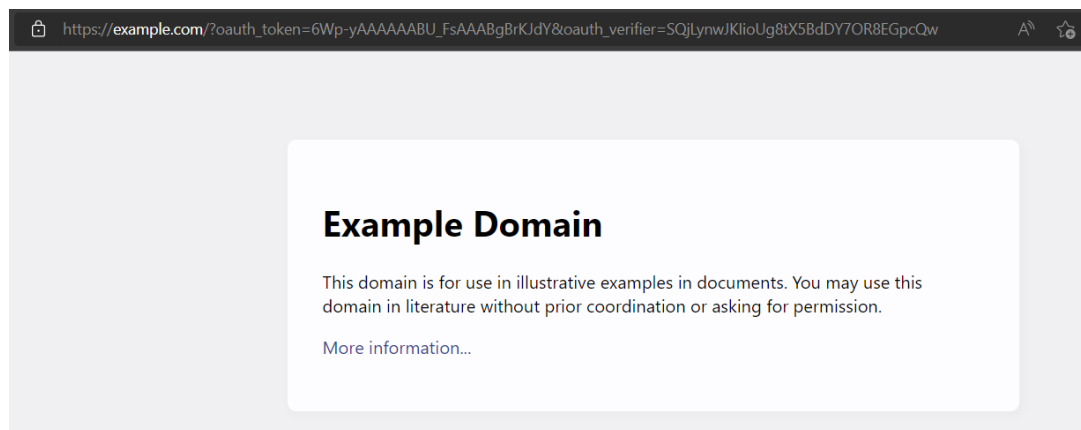


Gambar 4 tampilan setelah sebuah fungsi dengan OAuth1.0 dieksekusi



Gambar 5 tampilan untuk meminta izin pada aplikasi yang terhubung dengan callback_url

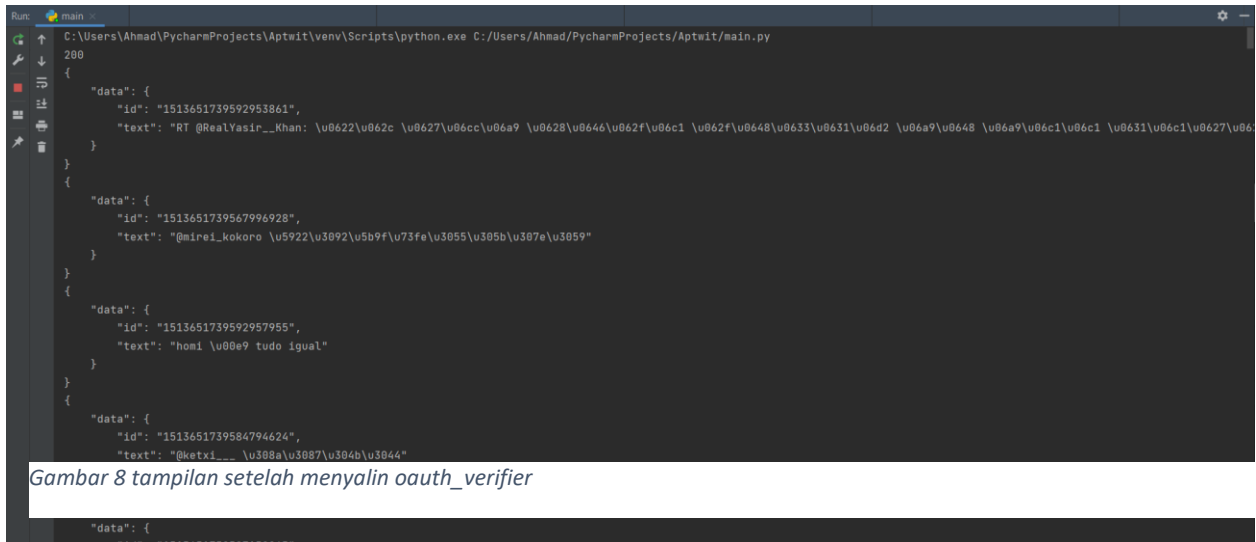
Setelah itu, akan dialihkan ke halaman sesuai 'CALLBACK_URL' yang telah diisi pada User Authentication Setting. Pada link tersebut, salin token yang memiliki nama variable `oauth_verifier`. Tempel hasil salin tersebut ke dalam terminal text editor dan jalankan program. Untuk tampilannya dapat dilihat pada gambar berikut.



Gambar 6 tampilan setelah izin diberikan dan `oauth_verifier` pada url halaman

Jika program telah dieksekusi dan response codenya 200, maka program, berhasil dieksekusi dan tampilannya seperti berikut:

Selanjutnya pada fungsi(endpoint) dengan OAuth2.0 App-Only memerlukan bearer_token dari developer portal. Selanjutnya, hanya perlu eksekusi program dan hasilnya akan ditampilkan.



```
Run: main
C:\Users\Ahmad\PycharmProjects\Aptwit\venv\Scripts\python.exe C:/Users/Ahmad/PycharmProjects/Aptwit/main.py
200
{
  "data": {
    "id": "1513651739592953861",
    "text": "RT @RealYasir_Khan: \u0622\u062c \u06cc\u06e9 \u0628\u0646\u062f\u06c1 \u062f\u0648\u0633\u0631\u0648\u0622 \u06e9\u0648 \u06e9\u06c1 \u0633\u0631\u0648\u0622\u0648"
  }
}
{
  "data": {
    "id": "1513651739592957928",
    "text": "@mirei_kokoro \u0622\u062c\u062f\u06c1 \u062f\u0648\u0633\u0631\u0648\u0622 \u06e9\u0648 \u06e9\u06c1 \u0633\u0631\u0648\u0622\u0648"
  }
}
{
  "data": {
    "id": "1513651739592957955",
    "text": "hml \u06e9 tudo igual"
  }
}
{
  "data": {
    "id": "1513651739584794624",
    "text": "@keti_... \u0628\u062f\u06c1 \u062f\u0648\u0633\u0631\u0648\u0622"
  }
}
```

Gambar 8 tampilan setelah menyalin oauth_verifier

Gambar 9 tampilan hasil akuisisi dari fungsi dengan OAuth2.0 App-Only



```
Click this link and authorize: https://api.twitter.com/oauth/authorize?oauth_token=6Wp-yAAAAA8U_FsAAA8q8r-KJdY
Paste the OAuth_Verifier here: 1451023923907158024
Response code: 200
{
  "data": {
    "created_at": "2021-10-21T03:14:43.000Z",
    "description": "beta tester",
    "id": "1451023923907158024",
    "location": "Makassar, Indonesia",
    "name": "Fordig Acc",
    "pinned_tweet_id": "1513385758572236800",
    "profile_image_url": "https://pbs.twimg.com/profile_images/1484391550188929026/NizRFq49_normal.jpg",
    "protected": false,
    "public_metrics": {
      "followers_count": 0,
      "following_count": 9,
      "listed_count": 1,
      "tweet_count": 10
    },
    "url": "",
    "username": "fordigacc",
    "verified": false
  }
}
```

Gambar 7 tampilan hasil akuisisi dan fungsi dengan OAuth1.0

Perhatikan bahwa, kode program pada function.py agar tidak diubah karena akan mempengaruhi eksekusi endpoint. Setiap objek yang berhasil diakuisisi, semuanya diatur didalam file model.py.

1.5 File model.py

File dengan nama model.py berisi query parameter setiap endpointnya. Maka jika ingin mendapatkan lebih banyak objek pada suatu endpoint maka dapat mempelajari setiap objek yang dibutuhkan pada <https://developer.twitter.com/en/docs>. Jenis autentikasi dengan key dan token yang berbeda dapat mempengaruhi objek yang dapat diakuisisi. Model.py juga berpengaruh pada parameter yang akan digunakan dalam memanggil endpoint pada file main.py. Isi dari model.py dapat dimodifikasi dan sesuaikan dengan kebutuhan dan variasi objek yang diinginkan.

3. Source Code

3.1 api_secret.py

```
#API_KEY dari Twitter Developer

#OAUTH1.0a
ACCESS_TOKEN = "YOUR_ACCESS_TOKEN"
ACCESS_TOKEN_SECRET = "YOUR_ACCESS_TOKEN_SECRET"
CONSUMER_KEY = 'YOUR_CONSUMER_KEY'
CONSUMER_SECRET = 'YOUR_CONSUMER_SECRET'

#OAUTH2.0
BEARER_TOKEN = "YOUR_BEARER_TOKEN"
```

3.2 model.py

```
#QUERY PARAMETER
#TWEETS
def get_tweet_par(tweet_ids):
    return {
        'ids': tweet_ids,
        'tweet.fields':
            'attachments,author_id,context_annotations,conversation_id,created_at,entities,geo,id,in_reply_to_user_id,lang,possibly_sensitive,non_public_metrics,referenced_tweets,reply_settings,source,text,withheld',
        'user.fields':
            'created_at,description,entities,id,location,name,pinned_tweet_id,profile_image_url,protected,public_metrics,url,username,verified,withheld',
        'expansions':
            'author_id,referenced_tweets.id,referenced_tweets.id.author_id,entities.mentions.username,attachments.poll_ids,attachments.media_keys,in_reply_to_user_id,geo.place_id',
        'poll.fields':
            'duration_minutes,end_datetime,id,options,voting_status',
        'place.fields':
            'contained_within,country,country_code,full_name,geo,id,name,place_type',
        'media.fields':
            'alt_text,duration_ms,height,media_key,non_public_metrics,organic_metrics,preview_image_url,promoted_metrics,public_metrics,type,url,width'
    }

def get_tweet_by_par():
    return {
        'tweet.fields':
            'attachments,author_id,context_annotations,conversation_id,created_at,entities
```

```

s,geo,id,in_reply_to_user_id,lang,non_public_metrics,public_metrics,organic_m
etrics,possibly_sensitive,referenced_tweets,reply_settings,source,text,withhe
ld',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld',
        'expansions':
'attachments.poll_ids,attachments.media_keys,author_id,entities.mentions.user
name,geo.place_id,in_reply_to_user_id,referenced_tweets.id,referenced_tweets.
id.author_id',
        'poll.fields':
'duration_minutes,end_datetime,id,options,voting_status',
        'place.fields':
'contained_within,country,country_code,full_name,geo,id,name,place_type',
        'media.fields':
'duration_ms,height,media_key,preview_image_url,type,url,width,public_metrics
,non_public_metrics,organic_metrics,promoted_metrics,alt_text'
    }

#LIKES
def get_likes_lookup_par(max_results, pagination_token):
    return {
        'expansions': 'pinned_tweet_id',
        'max_results': max_results,
        # 'media.fields': 'duration_ms, height, media_key, preview_image_url,
type, url, width, public_metrics, non_public_metrics, organic_metrics,
promoted_metrics, alt_text',
        'pagination_token': pagination_token,
        # 'place.fields': 'contained_within, country, country_code,
full_name, geo, id, name, place_type',
        # 'poll.fields': 'duration_minutes, end_datetime, id, options,
voting_status',
        'tweet.fields':
'attachments,author_id,context_annotations,conversation_id,created_at,entitie
s,geo,id,in_reply_to_user_id,lang,non_public_metrics,organic_metrics,possibly
_sensitive,promoted_metrics,public_metrics,referenced_tweets,reply_settings,s
ource,text,withheld',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld',
    }

def get_liked_tweets_par(max_results, pagination_token):
    return {
        'expansions':
'author_id,referenced_tweets.id,referenced_tweets.id.author_id,entities.menti
ons.username,attachments.poll_ids,attachments.media_keys,in_reply_to_user_id,
geo.place_id',
        'max_results': max_results,
        'media.fields':
'alt_text,duration_ms,height,media_key,preview_image_url,public_metrics,type,
url,width',
        'pagination_token': pagination_token,
        'place.fields':
'contained_within,country,country_code,full_name,geo,id,name,place_type',
        'poll.fields':
'duration_minutes,end_datetime,id,options,voting_status',

```

```

        'tweet.fields':
'attachments,author_id,context_annotations,conversation_id,created_at,entitie
s,geo,id,in_reply_to_user_id,lang,possibly_sensitive,public_metrics,reference
d_tweets,reply_settings,source,text,withheld',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld',
    }

#TIMELINE
def get_user_timeline_par():
    return {
        'exclude': 'replies,retweets',
        'expansions' :
'author_id,referenced_tweets.id,referenced_tweets.id.author_id,entities.menti
ons.username,attachments.poll_ids,attachments.media_keys,in_reply_to_user_id,
geo.place_id',
        'place.fields':
'contained_within,country,country_code,full_name,geo,id,name,place_type',
        'poll.fields':
'duration_minutes,end_datetime,id,options,voting_status',
        'media.fields':
'alt_text,duration_ms,height,media_key,non_public_metrics,organic_metrics,pre
view_image_url,public_metrics,type,url,width',
        'tweet.fields' :
'attachments,author_id,context_annotations,conversation_id,created_at,entitie
s,geo,id,in_reply_to_user_id,lang,possibly_sensitive,public_metrics,reference
d_tweets,reply_settings,source,text,withheld',
        'user.fields' :
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld'
    }

def get_user_mention_timeline_par():
    return {
        'expansions' :
'author_id,referenced_tweets.id,referenced_tweets.id.author_id,entities.menti
ons.username,attachments.poll_ids,attachments.media_keys,in_reply_to_user_id,
geo.place_id',
        'place.fields':
'contained_within,country,country_code,full_name,geo,id,name,place_type',
        'poll.fields':
'duration_minutes,end_datetime,id,options,voting_status',
        'media.fields':
'alt_text,duration_ms,height,media_key,organic_metrics,preview_image_url,publ
ic_metrics,type,url,width',
        'tweet.fields' :
'attachments,author_id,context_annotations,conversation_id,created_at,entitie
s,geo,id,in_reply_to_user_id,lang,possibly_sensitive,public_metrics,reference
d_tweets,reply_settings,source,text,withheld',
        'user.fields' :
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld'
    }

#USERS
def get_user_lookup_by_par(usernames):

```

```

    return {
        'usernames': usernames,
        'expansions': 'pinned_tweet_id',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_image_url,protected,public_metrics,url,username,verified,withheld',
        'tweet.fields':
'attachments,author_id,context_annotations,conversation_id,created_at,entities,geo,id,in_reply_to_user_id,lang,non_public_metrics,organic_metrics,possibly_sensitive,promoted_metrics,public_metrics,referenced_tweets,reply_settings,source,text,withheld'
    }

def get_user_lookup_by_username_par():
    return {
        'expansions': 'pinned_tweet_id',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_image_url,protected,public_metrics,url,username,verified,withheld',
        'tweet.fields':
'attachments,author_id,context_annotations,conversation_id,created_at,entities,geo,id,in_reply_to_user_id,lang,non_public_metrics,organic_metrics,possibly_sensitive,public_metrics,referenced_tweets,reply_settings,source,text,withheld'
    }

def get_users_lookup_par(user_ids):
    return {
        'ids': user_ids,
        'expansions': 'pinned_tweet_id',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_image_url,protected,public_metrics,url,username,verified,withheld',
        'tweet.fields':
'attachments,author_id,context_annotations,conversation_id,created_at,entities,geo,id,in_reply_to_user_id,lang,non_public_metrics,organic_metrics,possibly_sensitive,promoted_metrics,public_metrics,referenced_tweets,reply_settings,source,text,withheld'
    }

def get_user_lookup_by_id_par():
    return {
        'expansions': 'pinned_tweet_id',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_image_url,protected,public_metrics,url,username,verified,withheld',
        'tweet.fields':
'attachments,author_id,context_annotations,conversation_id,created_at,entities,geo,id,in_reply_to_user_id,lang,non_public_metrics,organic_metrics,possibly_sensitive,promoted_metrics,public_metrics,referenced_tweets,reply_settings,source,text,withheld'
    }

def get_my_par():
    return {
        'expansions': 'pinned_tweet_id',
        'user.fields':

```

```

'created_at,description,entities,id,location,name,pinned_tweet_id,profile_image_url,protected,public_metrics,url,username,verified,withheld',
    'tweet.fields':
'attachments,author_id,context_annotations,conversation_id,created_at,entities,geo,id,in_reply_to_user_id,lang,non_public_metrics,organic_metrics,possibly_sensitive,promoted_metrics,public_metrics,referenced_tweets,reply_settings,source,text,withheld'
    }

#SPACES
def get_spaces_by_id_par(space_id):
    return {
        'ids': space_id,
        'expansions':
'host_ids,creator_id,invited_user_ids,speaker_ids,topic_ids',
        'topic.fields': 'description,id,name',
        'space.fields':
'created_at,creator_id,ended_at,host_ids,id,invited_user_ids,is_ticketed,lang,participant_count,scheduled_start,speaker_ids,started_at,state,subscriber_count,title,topic_ids,updated_at',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_image_url,protected,public_metrics,url,username,verified,withheld'
    }

def get_search_space_par(str_query):
    return {
        'query': str_query,
        'expansions':
'host_ids,creator_id,invited_user_ids,speaker_ids,topic_ids',
        'state': 'all',
        #live,scheduled,all'
        'topic.fields': 'description,id,name',
        'space.fields':
'created_at,creator_id,ended_at,host_ids,id,invited_user_ids,is_ticketed,lang,participant_count,scheduled_start,speaker_ids,started_at,state,subscriber_count,title,topic_ids,updated_at',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_image_url,protected,public_metrics,url,username,verified,withheld',
    }

def get_spaces_par(space_ids):
    return {
        'ids': space_ids,
        'expansions':
'host_ids,creator_id,invited_user_ids,speaker_ids,topic_ids',
        'topic.fields': 'id,name,description',
        'space.fields':
'host_ids,created_at,creator_id,id,lang,invited_user_ids,participant_count,speaker_ids,started_at,ended_at,subscriber_count,topic_ids,state,title,updated_at,scheduled_start,is_ticketed',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_image_url,protected,public_metrics,url,username,verified,withheld',
    }

```

```

def get_spaces_by_creator_ids_par(user_ids):
    return {
        'user_ids': user_ids,
        'expansions': 'invited_user_ids,speaker_ids,creator_id,host_ids',
        'topic.fields': 'id,name,description',
        'space.fields':
'created_at,creator_id,ended_at,host_ids,id,invited_user_ids,is_ticketed,lang
,participant_count,scheduled_start,speaker_ids,started_at,state,subscriber_co
unt,title,topic_ids,updated_at',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld',
    }

def get_spaces_buyers():
    return {
        'expansions': 'pinned_tweet_id',
        'media.fields':
'duration_ms,height,media_key,preview_image_url,type,url,width,public_metrics
,non_public_metrics,organic_metrics,promoted_metrics,alt_text',
        'place.fields':
'contained_within,country,country_code,full_name,geo,id,name,place_type',
        'poll.fields':
'duration_minutes,end_datetime,id,options,voting_status',
        'tweet.fields':
'attachments,author_id,context_annotations,conversation_id,created_at,entitie
s,geo,id,in_reply_to_user_id,lang,non_public_metrics,public_metrics,organic_m
etrics,promoted_metrics,possibly_sensitive,referenced_tweets,reply_settings,s
ource,text,withheld',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld',
    }

def get_spaces_tweet():
    return {
        'expansions':
'attachments.poll_ids,attachments.media_keys,author_id,entities.mentions.user
name,geo.place_id,in_reply_to_user_id,referenced_tweets.id,referenced_tweets.
id.author_id',
        'media.fields':
'duration_ms,height,media_key,preview_image_url,type,url,width,public_metrics
,non_public_metrics,organic_metrics,promoted_metrics,alt_text',
        'place.fields':
'contained_within,country,country_code,full_name,geo,id,name,place_type',
        'poll.fields':
'duration_minutes,end_datetime,id,options,voting_status',
        'tweet.fields':
'attachments,author_id,context_annotations,conversation_id,created_at,entitie
s,geo,id,in_reply_to_user_id,lang,non_public_metrics,public_metrics,organic_m
etrics,promoted_metrics,possibly_sensitive,referenced_tweets,reply_settings,s
ource,text,withheld',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld',
    }

```



```

#RETWEETS
def get_retweets_lookup_par(pagination_token):
    return {
        'expansions': 'pinned_tweet_id',
        # 'media.fields': 'duration_ms, height, media_key, preview_image_url,
type, url, width, public_metrics, non_public_metrics, organic_metrics,
promoted_metrics, alt_text',
        'pagination_token': pagination_token,
        # 'place.fields':
'id,max_results,pagination_token,expansions,tweet.fields,user.fields',
        # 'poll.fields' :
'id,max_results,pagination_token,expansions,tweet.fields,user.fields',
        'tweet.fields' :
'attachments,author_id,context_annotations,conversation_id,created_at,entitie
s,geo,id,in_reply_to_user_id,lang,non_public_metrics,organic_metrics,possibly
_sensitive,promoted_metrics,public_metrics,referenced_tweets,reply_settings,s
ource,text,withheld',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld'
    }

#SEARCH TWEETS
def get_recent_tweet_count_par(str_query):
    return {
        'query': str_query,
        'end_time': None,
        'granularity': 'day',
        #minute, hour or day
        'since_id': None,
        'start_time': None,
        'until_id': None
    }

def
get_recent_search_par(str_query,end_time,max_results,next_token,since_id,star
t_time,until_id):
    return {
        'query': str_query,
        'end_time': end_time,
        #YYYY-MM-DDDDTHH:mm:ssZ
        'expansions':
'author_id,referenced_tweets.id,referenced_tweets.id.author_id,entities.menti
ons.username,attachments.poll_ids,attachments.media_keys,in_reply_to_user_id,
geo.place_id',
        'max_results': max_results,
        'media.fields':
'alt_text,duration_ms,height,media_key,preview_image_url,public_metrics,type,
url,width',
        'next_token': next_token,
        'place.fields':
'contained_within,country,country_code,full_name,geo,id,name,place_type',
        'poll.fields':
'duration_minutes,end_datetime,id,options,voting_status',
        'since_id': since_id,
        'start_time': start_time,
        'tweet.fields':

```

```

'attachments,author_id,context_annotations,conversation_id,created_at,entities,geo,id,in_reply_to_user_id,lang,possibly_sensitive,public_metrics,reference_d_tweets,reply_settings,source,text,withheld',
    'until_id': until_id,
    'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_image_url,protected,public_metrics,url,username,verified,withheld',
}

#QUOTE TWEETS
def get_quote_tweet_par(pagination_token):
    return {
        'expansions':
'author_id,referenced_tweets.id,referenced_tweets.id.author_id,entities.mentions.username,attachments.poll_ids,attachments.media_keys,in_reply_to_user_id,geo.place_id',
        'max_results': None,
        # 'media_fields': 'duration_ms, height, media_key, preview_image_url, type, url, width, public_metrics, non_public_metrics, organic_metrics, promoted_metrics, alt_text',
        'pagination_token': pagination_token,
        'place.fields':
'contained_within,country,country_code,full_name,geo,id,name,place_type',
        'poll.fields':
'duration_minutes,end_datetime,id,options,voting_status',
        'tweet.fields':
'attachments,author_id,context_annotations,conversation_id,created_at,entities,geo,id,in_reply_to_user_id,lang,non_public_metrics,organic_metrics,possibly_sensitive,promoted_metrics,public_metrics,referenced_tweets,reply_settings,source,text,withheld',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_image_url,protected,public_metrics,url,username,verified,withheld'
    }

#MUTES
def get_mutes_lookup_par(pagination_token):
    return {
        'expansions': 'pinned_tweet_id',
        'max_results': None,
        'pagination_token': pagination_token,
        'tweet.fields':
'attachments,author_id,context_annotations,conversation_id,created_at,entities,geo,id,in_reply_to_user_id,lang,non_public_metrics,organic_metrics,possibly_sensitive,promoted_metrics,public_metrics,referenced_tweets,reply_settings,source,text,withheld',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_image_url,protected,public_metrics,url,username,verified,withheld',
    }

#LIST
def user_owned_list(max_results, pagination_token):
    return {
        'expansions': 'owner_id',
        'list.fields':
'created_at,description,follower_count,id,member_count,name,owner_id,private'
    }

```

```

        'max_results': max_results,
        'pagination_token': pagination_token,
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld',
    }

def get_list_lookup_par():
    return {
        'expansions': 'owner_id',
        'list.fields':
'created_at,description,follower_count,id,member_count,name,owner_id,private'
    ,
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld',
    }

def get_pinned_list_par():
    return {
        'expansions': 'owner_id',
        'list.fields':
'created_at,description,follower_count,id,member_count,name,owner_id,private'
    ,
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld',
    }

def get_list_membership_par(max_results, pagination_token):
    return {
        'expansions': 'owner_id',
        'list.fields':
'created_at,description,follower_count,id,member_count,name,owner_id,private'
    ,
        'max_results': max_results,
        'pagination_token': pagination_token,
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld',
    }

def get_list_member_lookup_par(pagination_token, max_results):
    return {
        'expansions': 'pinned_tweet_id',
        'max_results': max_results,
        'pagination_token': pagination_token,
        'tweet.fields':
'attachments,author_id,context_annotations,conversation_id,created_at,entitie
s,geo,id,in_reply_to_user_id,lang,non_public_metrics,organic_metrics,possibly
_sensitive,promoted_metrics,public_metrics,referenced_tweets,reply_settings,s
ource,text,withheld',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld',
    }

```

```

def get_user_list_followed_par(max_results, pagination_token):
    return {
        'expansions': 'owner_id',
        'list.fields':
'created_at,description,follower_count,id,member_count,name,owner_id,private'
    ,
        'max_results': max_results,
        'pagination_token': pagination_token,
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld',
    }

def get_list_followers_lookup(max_results, pagination_token):
    return {
        'expansions': 'pinned_tweet_id',
        'max_results': max_results,
        'pagination_token': pagination_token,
        'tweet.fields':
'attachments,author_id,context_annotations,conversation_id,created_at,entitie
s,geo,id,in_reply_to_user_id,lang,possibly_sensitive,public_metrics,reference
d_tweets,reply_settings,source,text,withheld',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld',
    }

def get_list_tweets_par(max_results, pagination_token):
    return {
        'expansions': 'author_id',
        'max_results': max_results,
        'pagination_token': pagination_token,
        'tweet.fields':
'attachments,author_id,context_annotations,conversation_id,created_at,entitie
s,geo,id,in_reply_to_user_id,lang,possibly_sensitive,public_metrics,reference
d_tweets,reply_settings,source,text,withheld',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld',
    }

#FOLLOWS
def get_following_lookup_par(max_results, pagination_token):
    return {
        'expansions': 'pinned_tweet_id',
        'max_results': max_results,
        'pagination_token': pagination_token,
        'tweet.fields':
'attachments,author_id,context_annotations,conversation_id,created_at,entitie
s,geo,id,in_reply_to_user_id,lang,possibly_sensitive,public_metrics,reference
d_tweets,reply_settings,source,text,withheld',
        'user.fields':
'created_at,description,entities,id,location,name,pinned_tweet_id,profile_ima
ge_url,protected,public_metrics,url,username,verified,withheld',
    }

```

```

def get_followers_lookup_par(max_results, pagination_token):
    return {
        'expansions': 'pinned_tweet_id',
        'max_results': max_results,
        'pagination_token': pagination_token,
        'tweet.fields':
            'attachments,author_id,context_annotations,conversation_id,created_at,entities,geo,id,in_reply_to_user_id,lang,possibly_sensitive,public_metrics,reference_d_tweets,reply_settings,source,text,withheld',
        'user.fields':
            'created_at,description,entities,id,location,name,pinned_tweet_id,profile_image_url,protected,public_metrics,url,username,verified,withheld',
    }

#BLOCKS
def block_lookups_par(max_results, pagination_token):
    return {
        'expansion': 'pinned_tweet_id',
        'max_results': max_results,
        'pagination_token': pagination_token,
        'tweet.fields': 'attachments, author_id, context_annotations,
conversation_id, created_at, entities, geo, id, in_reply_to_user_id, lang,
non_public_metrics, public_metrics, organic_metrics, promoted_metrics,
possibly_sensitive, referenced_tweets, reply_settings, source, text,
withheld',
        'user.fields': 'created_at, description, entities, id, location,
name, pinned_tweet_id, profile_image_url, protected, public_metrics, url,
username, verified, withheld',
    }

```

3.3 function.py

```

from requests_oauthlib import OAuth1Session
from requests_oauthlib import OAuth2Session
from getwit import model
from getwit import apisecret
import requests
import json
import base64
import hashlib
import re
import os

consumer_key = apisecret.CONSUMER_KEY
consumer_secret = apisecret.CONSUMER_SECRET
bearer_token = apisecret.BEARER_TOKEN
client_id = apisecret.CLIENT_ID
redirect_uri = apisecret.CALLBACK_URL
headers = {"Authorization": "Bearer {}".format(bearer_token)}

#TWEETS
def get_tweet_lookup(tweet_ids):
    request_token_url = "https://api.twitter.com/oauth/request_token"
    oauth = OAuth1Session(consumer_key, client_secret=consumer_secret)

```

```

try:
    fetch_response = oauth.fetch_request_token(request_token_url)
except ValueError:
    print(
        "There may have been an issue with the consumer_key or
consumer_secret you entered."
    )

resource_owner_key = fetch_response.get("oauth_token")
resource_owner_secret = fetch_response.get("oauth_token_secret")
print("Got OAuth token: %s" % resource_owner_key)

# Get authorization
base_authorization_url = "https://api.twitter.com/oauth/authorize"
authorization_url = oauth.authorization_url(base_authorization_url)
print("Click this link and authorize: %s" % authorization_url)
verifier = input("Paste the OAuth_Verifier here: ")

# Get the access token
access_token_url = "https://api.twitter.com/oauth/access_token"
oauth = OAuth1Session(
    consumer_key,
    client_secret=consumer_secret,
    resource_owner_key=resource_owner_key,
    resource_owner_secret=resource_owner_secret,
    verifier=verifier,
)
oauth_tokens = oauth.fetch_access_token(access_token_url)

access_token = oauth_tokens["oauth_token"]
access_token_secret = oauth_tokens["oauth_token_secret"]

# Make the request
oauth = OAuth1Session(
    consumer_key,
    client_secret=consumer_secret,
    resource_owner_key=access_token,
    resource_owner_secret=access_token_secret,
)

params = model.get_tweet_par(tweet_ids)
response = oauth.get(
    "https://api.twitter.com/2/tweets", params=params
)

if response.status_code != 200:
    raise Exception(
        "Request returned an error: {} {}".format(response.status_code,
response.text)
    )

print("Response code: {}".format(response.status_code))
json_response = response.json()
print(json.dumps(json_response, indent=4, sort_keys=True))

def get_tweet_by(tweet_id):

```

```

request_token_url = "https://api.twitter.com/oauth/request_token"
oauth = OAuth1Session(consumer_key, client_secret=consumer_secret)

try:
    fetch_response = oauth.fetch_request_token(request_token_url)
except ValueError:
    print(
        "There may have been an issue with the consumer_key or
consumer_secret you entered."
    )

resource_owner_key = fetch_response.get("oauth_token")
resource_owner_secret = fetch_response.get("oauth_token_secret")
print("Got OAuth token: %s" % resource_owner_key)

# Get authorization
base_authorization_url = "https://api.twitter.com/oauth/authorize"
authorization_url = oauth.authorization_url(base_authorization_url)
print("Please go here and authorize: %s" % authorization_url)
verifier = input("Paste the PIN here: ")

# Get the access token
access_token_url = "https://api.twitter.com/oauth/access_token"
oauth = OAuth1Session(
    consumer_key,
    client_secret=consumer_secret,
    resource_owner_key=resource_owner_key,
    resource_owner_secret=resource_owner_secret,
    verifier=verifier,
)
oauth_tokens = oauth.fetch_access_token(access_token_url)

access_token = oauth_tokens["oauth_token"]
access_token_secret = oauth_tokens["oauth_token_secret"]

# Make the request
oauth = OAuth1Session(
    consumer_key,
    client_secret=consumer_secret,
    resource_owner_key=access_token,
    resource_owner_secret=access_token_secret,
)

params = model.get_tweet_by_par()
response = oauth.get(
    "https://api.twitter.com/2/tweets/{}".format(tweet_id), params=params
)

if response.status_code != 200:
    raise Exception(
        "Request returned an error: {} {}".format(response.status_code,
response.text)
    )

print("Response code: {}".format(response.status_code))
json_response = response.json()

```

```

print(json.dumps(json_response, indent=4, sort_keys=True))

#LIKES
def get_likes_lookup(tweet_id, max_results, pagination_token):
    url = "https://api.twitter.com/2/tweets/{}/liking_users".format(tweet_id)
    params = model.get_likes_lookup_par(max_results, pagination_token)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
        if respon.status_code != 200:
            raise Exception(
                "Request returned an error: {} {}".format(
                    respon.status_code, respon.text
                )
            )

def get_liked_tweets(user_id, max_results, pagination_token):
    url = "https://api.twitter.com/2/users/{}/liked_tweets".format(user_id)
    params = model.get_liked_tweets_par(max_results, pagination_token)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
        if respon.status_code != 200:
            raise Exception(
                "Request returned an error: {} {}".format(
                    respon.status_code, respon.text
                )
            )

#TIMELINE
def get_user_timeline(user_id):
    url = 'https://api.twitter.com/2/users/{}/tweets'.format(user_id)
    params = model.get_user_timeline_par()
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
    if respon.status_code != 200:
        raise Exception(
            "Request returned an error: {} {}".format(
                respon.status_code, respon.text
            )
        )

def get_user_mention_timeline(user_id):
    url = 'https://api.twitter.com/2/users/{}/mentions'.format(user_id)
    params = model.get_user_mention_timeline_par()
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)

```



```

for response_line in respon.iter_lines():
    if response_line:
        json_response = json.loads(response_line)
        print(json.dumps(json_response, indent=4, sort_keys=True))
if respon.status_code != 200:
    raise Exception(
        "Request returned an error: {} {}".format(
            respon.status_code, respon.text
        )
    )

#USERS
def get_user_lookup_by(usernames):
    request_token_url = "https://api.twitter.com/oauth/request_token"
    oauth = OAuth1Session(consumer_key, client_secret=consumer_secret)

    try:
        fetch_response = oauth.fetch_request_token(request_token_url)
    except ValueError:
        print(
            "There may have been an issue with the consumer_key or
consumer_secret you entered."
        )

    resource_owner_key = fetch_response.get("oauth_token")
    resource_owner_secret = fetch_response.get("oauth_token_secret")
    print("Got OAuth token: %s" % resource_owner_key)

    # Get authorization
    base_authorization_url = "https://api.twitter.com/oauth/authorize"
    authorization_url = oauth.authorization_url(base_authorization_url)
    print("Please go here and authorize: %s" % authorization_url)
    verifier = input("Paste the PIN here: ")

    # Get the access token
    access_token_url = "https://api.twitter.com/oauth/access_token"
    oauth = OAuth1Session(
        consumer_key,
        client_secret=consumer_secret,
        resource_owner_key=resource_owner_key,
        resource_owner_secret=resource_owner_secret,
        verifier=verifier,
    )
    oauth_tokens = oauth.fetch_access_token(access_token_url)

    access_token = oauth_tokens["oauth_token"]
    access_token_secret = oauth_tokens["oauth_token_secret"]

    # Make the request
    oauth = OAuth1Session(
        consumer_key,
        client_secret=consumer_secret,
        resource_owner_key=access_token,
        resource_owner_secret=access_token_secret,
    )

```

```

params = model.get_user_lookup_by_par(usernames)
response = oauth.get(
    "https://api.twitter.com/2/users/by", params=params
)

if response.status_code != 200:
    raise Exception(
        "Request returned an error: {} {}".format(response.status_code,
response.text)
    )

print("Response code: {}".format(response.status_code))
json_response = response.json()
print(json.dumps(json_response, indent=4, sort_keys=True))

def get_my_profile():
    request_token_url = "https://api.twitter.com/oauth/request_token"
    oauth = OAuth1Session(consumer_key, client_secret=consumer_secret)

    try:
        fetch_response = oauth.fetch_request_token(request_token_url)
    except ValueError:
        print(
            "There may have been an issue with the consumer_key or
consumer_secret you entered."
        )

    resource_owner_key = fetch_response.get("oauth_token")
    resource_owner_secret = fetch_response.get("oauth_token_secret")
    print("Got OAuth token: %s" % resource_owner_key)

    # Get authorization
    base_authorization_url = "https://api.twitter.com/oauth/authorize"
    authorization_url = oauth.authorization_url(base_authorization_url)
    print("Please go here and authorize: %s" % authorization_url)
    verifier = input("Paste the PIN here: ")

    # Get the access token
    access_token_url = "https://api.twitter.com/oauth/access_token"
    oauth = OAuth1Session(
        consumer_key,
        client_secret=consumer_secret,
        resource_owner_key=resource_owner_key,
        resource_owner_secret=resource_owner_secret,
        verifier=verifier,
    )
    oauth_tokens = oauth.fetch_access_token(access_token_url)

    access_token = oauth_tokens["oauth_token"]
    access_token_secret = oauth_tokens["oauth_token_secret"]

    # Make the request
    oauth = OAuth1Session(
        consumer_key,
        client_secret=consumer_secret,
        resource_owner_key=access_token,
        resource_owner_secret=access_token_secret,

```

```

)

params = model.get_my_par()
response = oauth.get(
    "https://api.twitter.com/2/users/me", params=params
)

if response.status_code != 200:
    raise Exception(
        "Request returned an error: {} {}".format(response.status_code,
response.text)
    )

print("Response code: {}".format(response.status_code))
json_response = response.json()
print(json.dumps(json_response, indent=4, sort_keys=True))

def get_user_lookup_by_username(username):
    request_token_url = "https://api.twitter.com/oauth/request_token"
    oauth = OAuth1Session(consumer_key, client_secret=consumer_secret)

    try:
        fetch_response = oauth.fetch_request_token(request_token_url)
    except ValueError:
        print(
            "There may have been an issue with the consumer_key or
consumer_secret you entered."
        )

    resource_owner_key = fetch_response.get("oauth_token")
    resource_owner_secret = fetch_response.get("oauth_token_secret")
    print("Got OAuth token: %s" % resource_owner_key)

    # Get authorization
    base_authorization_url = "https://api.twitter.com/oauth/authorize"
    authorization_url = oauth.authorization_url(base_authorization_url)
    print("Click this link and authorize: %s" % authorization_url)
    verifier = input("Paste the OAuth Verifier here: ")

    # Get the access token
    access_token_url = "https://api.twitter.com/oauth/access_token"
    oauth = OAuth1Session(
        consumer_key,
        client_secret=consumer_secret,
        resource_owner_key=resource_owner_key,
        resource_owner_secret=resource_owner_secret,
        verifier=verifier,
    )
    oauth_tokens = oauth.fetch_access_token(access_token_url)

    access_token = oauth_tokens["oauth_token"]
    access_token_secret = oauth_tokens["oauth_token_secret"]

    # Make the request
    oauth = OAuth1Session(
        consumer_key,
        client_secret=consumer_secret,

```

```

        resource_owner_key=access_token,
        resource_owner_secret=access_token_secret,
    )

    params = model.get_user_lookup_by_username_par()
    response = oauth.get(
        "https://api.twitter.com/2/users/by/username/{}".format(username),
        params=params
    )

    if response.status_code != 200:
        raise Exception(
            "Request returned an error: {} {}".format(response.status_code,
            response.text)
        )

    print("Response code: {}".format(response.status_code))
    json_response = response.json()
    print(json.dumps(json_response, indent=4, sort_keys=True))

def get_users_lookup(user_ids):
    request_token_url = "https://api.twitter.com/oauth/request_token"
    oauth = OAuth1Session(consumer_key, client_secret=consumer_secret)

    try:
        fetch_response = oauth.fetch_request_token(request_token_url)
    except ValueError:
        print(
            "There may have been an issue with the consumer_key or
            consumer_secret you entered."
        )

    resource_owner_key = fetch_response.get("oauth_token")
    resource_owner_secret = fetch_response.get("oauth_token_secret")
    print("Got OAuth token: %s" % resource_owner_key)

    # Get authorization
    base_authorization_url = "https://api.twitter.com/oauth/authorize"
    authorization_url = oauth.authorization_url(base_authorization_url)
    print("Please go here and authorize: %s" % authorization_url)
    verifier = input("Paste the PIN here: ")

    # Get the access token
    access_token_url = "https://api.twitter.com/oauth/access_token"
    oauth = OAuth1Session(
        consumer_key,
        client_secret=consumer_secret,
        resource_owner_key=resource_owner_key,
        resource_owner_secret=resource_owner_secret,
        verifier=verifier,
    )
    oauth_tokens = oauth.fetch_access_token(access_token_url)

    access_token = oauth_tokens["oauth_token"]
    access_token_secret = oauth_tokens["oauth_token_secret"]

    # Make the request

```

```

oauth = OAuth1Session(
    consumer_key,
    client_secret=consumer_secret,
    resource_owner_key=access_token,
    resource_owner_secret=access_token_secret,
)

params = model.get_users_lookup_par(user_ids)
response = oauth.get(
    "https://api.twitter.com/2/users", params=params
)

if response.status_code != 200:
    raise Exception(
        "Request returned an error: {} {}".format(response.status_code,
response.text)
    )

print("Response code: {}".format(response.status_code))
json_response = response.json()
print(json.dumps(json_response, indent=4, sort_keys=True))

def get_user_lookup_by_id(user_id):
    request_token_url = "https://api.twitter.com/oauth/request_token"
    oauth = OAuth1Session(consumer_key, client_secret=consumer_secret)

    try:
        fetch_response = oauth.fetch_request_token(request_token_url)
    except ValueError:
        print(
            "There may have been an issue with the consumer_key or
consumer_secret you entered."
        )

    resource_owner_key = fetch_response.get("oauth_token")
    resource_owner_secret = fetch_response.get("oauth_token_secret")
    print("Got OAuth token: %s" % resource_owner_key)

    # Get authorization
    base_authorization_url = "https://api.twitter.com/oauth/authorize"
    authorization_url = oauth.authorization_url(base_authorization_url)
    print("Please go here and authorize: %s" % authorization_url)
    verifier = input("Paste the PIN here: ")

    # Get the access token
    access_token_url = "https://api.twitter.com/oauth/access_token"
    oauth = OAuth1Session(
        consumer_key,
        client_secret=consumer_secret,
        resource_owner_key=resource_owner_key,
        resource_owner_secret=resource_owner_secret,
        verifier=verifier,
    )
    oauth_tokens = oauth.fetch_access_token(access_token_url)

    access_token = oauth_tokens["oauth_token"]
    access_token_secret = oauth_tokens["oauth_token_secret"]

```

```

# Make the request
oauth = OAuth1Session(
    consumer_key,
    client_secret=consumer_secret,
    resource_owner_key=access_token,
    resource_owner_secret=access_token_secret,
)

params = model.get_user_lookup_by_id_par()
response = oauth.get(
    "https://api.twitter.com/2/users/{}".format(user_id), params=params
)

if response.status_code != 200:
    raise Exception(
        "Request returned an error: {} {}".format(response.status_code,
response.text)
    )

print("Response code: {}".format(response.status_code))
json_response = response.json()
print(json.dumps(json_response, indent=4, sort_keys=True))

#SPACES
def get_space_lookup(space_id):
    url = 'https://api.twitter.com/2/spaces/'.format(space_id)
    params = model.get_spaces_by_id_par(space_id)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
    if respon.status_code != 200:
        raise Exception(
            "Request returned an error: {} {}".format(
                respon.status_code, respon.text
            )
        )

def get_search_space(str_query):
    url = 'https://api.twitter.com/2/spaces/search'.format(str_query)
    params = model.get_search_space_par(str_query)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
    if respon.status_code != 200:
        raise Exception(
            "Request returned an error: {} {}".format(
                respon.status_code, respon.text
            )
        )

```

```

def get_spaces(space_ids):
    url = 'https://api.twitter.com/2/spaces'
    params = model.get_spaces_par(space_ids)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
    if respon.status_code != 200:
        raise Exception(
            "Request returned an error: {} {}".format(
                respon.status_code, respon.text
            )
        )

def get_spaces_by_creator_ids(user_ids):
    url = 'https://api.twitter.com/2/spaces/by/creator_ids'.format(user_ids)
    params = model.get_spaces_by_creator_ids_par(user_ids)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
    if respon.status_code != 200:
        raise Exception(
            "Request returned an error: {} {}".format(
                respon.status_code, respon.text
            )
        )

#OAuth2.0 Authorization Code with PKCE
def get_spaces_buyers(space_id):
    url = 'https://api.twitter.com/2/spaces/{}/buyers'.format(space_id)
    params = model.get_spaces_buyers()
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
    if respon.status_code != 200:
        raise Exception(
            "Request returned an error: {} {}".format(
                respon.status_code, respon.text
            )
        )

def get_spaces_tweet(space_id):
    url = 'https://api.twitter.com/2/spaces/:id/tweets'.format(space_id)
    params = model.get_search_space_par(space_id)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)

```

```

        print(json.dumps(json_response, indent=4, sort_keys=True))
    if respon.status_code != 200:
        raise Exception(
            "Request returned an error: {} {}".format(
                respon.status_code, respon.text
            )
        )

#VOLUME STREAMS
def get_sample_stream():
    headers = {"Authorization": "Bearer {}".format(bearer_token),
               "User-Agent": "v2SampleStreamJS"}
    url = 'https://api.twitter.com/2/tweets/sample/stream'
    respon = requests.request("GET", url, headers=headers, stream=True)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
    if respon.status_code != 200:
        raise Exception(
            "Request returned an error: {} {}".format(
                respon.status_code, respon.text
            )
        )

#RETWEETS
def get_retweets_lookup(tweet_id, pagination_token):
    url = "https://api.twitter.com/2/tweets/{}/retweeted_by".format(tweet_id)
    params = model.get_retweets_lookup_par(pagination_token)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
    if respon.status_code != 200:
        raise Exception(
            "Request returned an error: {} {}".format(
                respon.status_code, respon.text
            )
        )

#SEARCH TWEETS
def get_recent_tweet_count(str_query):
    url = "https://api.twitter.com/2/tweets/counts/recent".format(str_query)
    params = model.get_recent_tweet_count_par(str_query)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
    if respon.status_code != 200:
        raise Exception(
            "Request returned an error: {} {}".format(
                respon.status_code, respon.text
            )
        )

```



```

    )
    )

#TWEET COUNT
def
get_recent_search(str_query,end_time,max_results,next_token,since_id,start_time,until_id):
    url = "https://api.twitter.com/2/tweets/search/recent".format(str_query)
    params =
model.get_recent_search_par(str_query,end_time,max_results,next_token,since_id,start_time,until_id)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
    if respon.status_code != 200:
        raise Exception(
            "Request returned an error: {} {}".format(
                respon.status_code, respon.text
            )
        )
    )

#QUOTE TWEETS
def get_quote_tweet(tweet_id, pagination_token):
    url = "https://api.twitter.com/2/tweets/{}/quote_tweets".format(tweet_id)
    params = model.get_quote_tweet_par(pagination_token)
    respon = requests.request("GET",url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
    if respon.status_code != 200:
        raise Exception(
            "Request returned an error: {} {}".format(
                respon.status_code, respon.text
            )
        )
    )

#MUTES
def get_mutes_lookup(user_id, pagination_token):
    request_token_url = "https://api.twitter.com/oauth/request_token"
    oauth = OAuth1Session(consumer_key, client_secret=consumer_secret)

    try:
        fetch_response = oauth.fetch_request_token(request_token_url)
    except ValueError:
        print(
            "There may have been an issue with the consumer_key or
consumer_secret you entered."
        )

    resource_owner_key = fetch_response.get("oauth_token")
    resource_owner_secret = fetch_response.get("oauth_token_secret")
    print("Got OAuth token: %s" % resource_owner_key)

```

```

# Get authorization
base_authorization_url = "https://api.twitter.com/oauth/authorize"
authorization_url = oauth.authorization_url(base_authorization_url)
print("Please go here and authorize: %s" % authorization_url)
verifier = input("Paste the PIN here: ")

# Get the access token
access_token_url = "https://api.twitter.com/oauth/access_token"
oauth = OAuth1Session(
    consumer_key,
    client_secret=consumer_secret,
    resource_owner_key=resource_owner_key,
    resource_owner_secret=resource_owner_secret,
    verifier=verifier,
)
oauth_tokens = oauth.fetch_access_token(access_token_url)

access_token = oauth_tokens["oauth_token"]
access_token_secret = oauth_tokens["oauth_token_secret"]

# Make the request
oauth = OAuth1Session(
    consumer_key,
    client_secret=consumer_secret,
    resource_owner_key=access_token,
    resource_owner_secret=access_token_secret,
)

params = model.get_mutes_lookup_par(pagination_token)
response = oauth.get(
    "https://api.twitter.com/2/users/{}/muting".format(user_id),
    params=params
)

if response.status_code != 200:
    raise Exception(
        "Request returned an error: {} {}".format(response.status_code,
        response.text)
    )

print("Response code: {}".format(response.status_code))
json_response = response.json()
print(json.dumps(json_response, indent=4, sort_keys=True))

#LIST
def get_user_owned_list(user_id, max_results, pagination_token):
    url = "https://api.twitter.com/2/users/{}/owned_lists".format(user_id)
    params = model.user_owned_list(max_results, pagination_token)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
        if respon.status_code != 200:
            raise Exception(

```

```

        "Request returned an error: {} {}".format(
            respon.status_code, respon.text
        )
    )

def get_list_lookup(list_id):
    url = "https://api.twitter.com/2/lists/{}".format(list_id)
    params = model.get_list_lookup_par()
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
        if respon.status_code != 200:
            raise Exception(
                "Request returned an error: {} {}".format(
                    respon.status_code, respon.text
                )
            )

def get_pinned_list(user_id):
    request_token_url = "https://api.twitter.com/oauth/request_token"
    oauth = OAuth1Session(consumer_key, client_secret=consumer_secret)

    try:
        fetch_response = oauth.fetch_request_token(request_token_url)
    except ValueError:
        print(
            "There may have been an issue with the consumer_key or\n"
            "consumer_secret you entered."
        )

    resource_owner_key = fetch_response.get("oauth_token")
    resource_owner_secret = fetch_response.get("oauth_token_secret")
    print("Got OAuth token: %s" % resource_owner_key)

    # Get authorization
    base_authorization_url = "https://api.twitter.com/oauth/authorize"
    authorization_url = oauth.authorization_url(base_authorization_url)
    print("Please go here and authorize: %s" % authorization_url)
    verifier = input("Paste the PIN here: ")

    # Get the access token
    access_token_url = "https://api.twitter.com/oauth/access_token"
    oauth = OAuth1Session(
        consumer_key,
        client_secret=consumer_secret,
        resource_owner_key=resource_owner_key,
        resource_owner_secret=resource_owner_secret,
        verifier=verifier,
    )
    oauth_tokens = oauth.fetch_access_token(access_token_url)

    access_token = oauth_tokens["oauth_token"]
    access_token_secret = oauth_tokens["oauth_token_secret"]

```

```

# Make the request
oauth = OAuth1Session(
    consumer_key,
    client_secret=consumer_secret,
    resource_owner_key=access_token,
    resource_owner_secret=access_token_secret,
)

params = model.get_pinned_list_par()
response = oauth.get(
    "https://api.twitter.com/2/users/{}/pinned_lists".format(user_id),
    params=params
)

if response.status_code != 200:
    raise Exception(
        "Request returned an error: {} {}".format(response.status_code,
        response.text)
    )

print("Response code: {}".format(response.status_code))
json_response = response.json()
print(json.dumps(json_response, indent=4, sort_keys=True))

def get_user_list_memberships(user_id, max_results, pagination_token):
    url =
    "https://api.twitter.com/2/users/{}/list_memberships".format(user_id)
    params = model.get_list_membership_par(max_results, pagination_token)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
        if respon.status_code != 200:
            raise Exception(
                "Request returned an error: {} {}".format(
                    respon.status_code, respon.text
                )
            )

def get_list_member(list_id, max_results, pagination_token):
    url = "https://api.twitter.com/2/lists/{}/members".format(list_id)
    params = model.get_list_member_lookup_par(max_results, pagination_token)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
        if respon.status_code != 200:
            raise Exception(
                "Request returned an error: {} {}".format(
                    respon.status_code, respon.text
                )
            )

```

```

def get_user_list_followed(user_id, max_results, pagination_token):
    url = "https://api.twitter.com/2/users/{}/followed_lists".format(user_id)
    params = model.get_user_list_followed_par(max_results, pagination_token)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
        if respon.status_code != 200:
            raise Exception(
                "Request returned an error: {} {}".format(
                    respon.status_code, respon.text
                )
            )

def get_list_followers(list_id, max_results, pagination_token):
    url = "https://api.twitter.com/2/lists/{}/followers".format(list_id)
    params = model.get_list_followers_lookup(max_results, pagination_token)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
        if respon.status_code != 200:
            raise Exception(
                "Request returned an error: {} {}".format(
                    respon.status_code, respon.text
                )
            )

def get_list_tweets(list_id, max_results, pagination_token):
    url = "https://api.twitter.com/2/lists/{}/tweets".format(list_id)
    params = model.get_list_tweets_par(max_results, pagination_token)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
        if respon.status_code != 200:
            raise Exception(
                "Request returned an error: {} {}".format(
                    respon.status_code, respon.text
                )
            )

#FOLLOWS
def get_following_lookup(user_id, max_results, pagination_token):
    url = "https://api.twitter.com/2/users/{}/following".format(user_id)
    params = model.get_following_lookup_par(max_results, pagination_token)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)

```

```

        print(json.dumps(json_response, indent=4, sort_keys=True))
    if respon.status_code != 200:
        raise Exception(
            "Request returned an error: {} {}".format(
                respon.status_code, respon.text
            )
        )
    )

def get_followers_lookup(user_id, max_results, pagination_token):
    url = "https://api.twitter.com/2/users/{}/followers".format(user_id)
    params = model.get_following_lookup_par(max_results, pagination_token)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
    if respon.status_code != 200:
        raise Exception(
            "Request returned an error: {} {}".format(
                respon.status_code, respon.text
            )
        )
    )

#BLOCK
def get_block_lookup(user_id, max_results, pagination_token):
    request_token_url = "https://api.twitter.com/oauth/request_token"
    oauth = OAuth1Session(consumer_key, client_secret=consumer_secret)

    try:
        fetch_response = oauth.fetch_request_token(request_token_url)
    except ValueError:
        print(
            "There may have been an issue with the consumer_key or
consumer_secret you entered."
        )

    resource_owner_key = fetch_response.get("oauth_token")
    resource_owner_secret = fetch_response.get("oauth_token_secret")
    print("Got OAuth token: %s" % resource_owner_key)

    # Get authorization
    base_authorization_url = "https://api.twitter.com/oauth/authorize"
    authorization_url = oauth.authorization_url(base_authorization_url)
    print("Please go here and authorize: %s" % authorization_url)
    verifier = input("Paste the PIN here: ")

    # Get the access token
    access_token_url = "https://api.twitter.com/oauth/access_token"
    oauth = OAuth1Session(
        consumer_key,
        client_secret=consumer_secret,
        resource_owner_key=resource_owner_key,
        resource_owner_secret=resource_owner_secret,
        verifier=verifier,
    )
    oauth_tokens = oauth.fetch_access_token(access_token_url)

```

```

access_token = oauth_tokens["oauth_token"]
access_token_secret = oauth_tokens["oauth_token_secret"]

# Make the request
oauth = OAuth1Session(
    consumer_key,
    client_secret=consumer_secret,
    resource_owner_key=access_token,
    resource_owner_secret=access_token_secret,
)

params = model.block_lookups_par(max_results, pagination_token)
response = oauth.get(
    "https://api.twitter.com/2/users/{}/blocking".format(user_id),
    params=params
)

if response.status_code != 200:
    raise Exception(
        "Request returned an error: {} {}".format(response.status_code,
        response.text)
    )

print("Response code: {}".format(response.status_code))
json_response = response.json()
print(json.dumps(json_response, indent=4, sort_keys=True))

#OAUTH 2.0 App-Only & end point full archive search Twitter v2 for academic
research
#SEARCH TWEETS ALL
def full_archive_search(str_query, max_results):
    url = "https://api.twitter.com/2/tweets/search/all".format(str_query)
    params = model.full_archive_search_par(str_query, max_results)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))
        if respon.status_code != 200:
            raise Exception(
                "Request returned an error: {} {}".format(
                    respon.status_code, respon.text
                )
            )

def full_archive_tweet_count(str_query):
    url = "https://api.twitter.com/2/tweets/counts/all".format(str_query)
    params = model.full_archive_tweet_count_par(str_query)
    respon = requests.request("GET", url, headers=headers, params=params)
    print(respon.status_code)
    for response_line in respon.iter_lines():
        if response_line:
            json_response = json.loads(response_line)
            print(json.dumps(json_response, indent=4, sort_keys=True))

```

```
if respon.status_code != 200:
    raise Exception(
        "Request returned an error: {} {}".format(
            respon.status_code, respon.text
        )
    )
```