https://bmtgoncalves.github.io/TorinoCourse/

Lecture II - Online Social Networks

Bruno Gonçalves
www.bgoncalves.com

NYU DATA SCIENCE

Authentication Methodologies

- Much of the content available online is only accessible to specific individuals for privacy, copyright protection, etc…
- Three main ways of authenticating users:
  - BasicAuth - The first and most basic one. Plain text user name and password sent to the server
  - OAuth 1 - Developed by a consortium of Industry leaders to provide transparent and secure authentication.
  - OAuth 2 - An improvement on OAuth 1 designed to allow users to more easily share they're content on social media, etc…
  - OpenID - A predecessor to OAuth that has gone out of favor.

BasicAuth

http://requests.readthedocs.org/en/latest/

- "The mother of all authentication protocols"
- Insecure but easy to use with standard implementations in all networking tools
- In particular, in requests:
    - requests.get(url, auth=("user", "pass")) open the given url and authenticate with username="user" and password="pass"

```
import requests
import sys

url = "http://httpbin.org/basic-auth/user/passwd"

request = requests.get(url, auth=("user", "passwd"))

if request.status_code != 200:
    print("Error found", request.get_code(),
file=sys.stderr)

content_type = request.headers["content-type"]

response = request.json()

if response["authenticated"]:
    print("Authentication Successful")
```

basic_auth.py

@bgoncalves

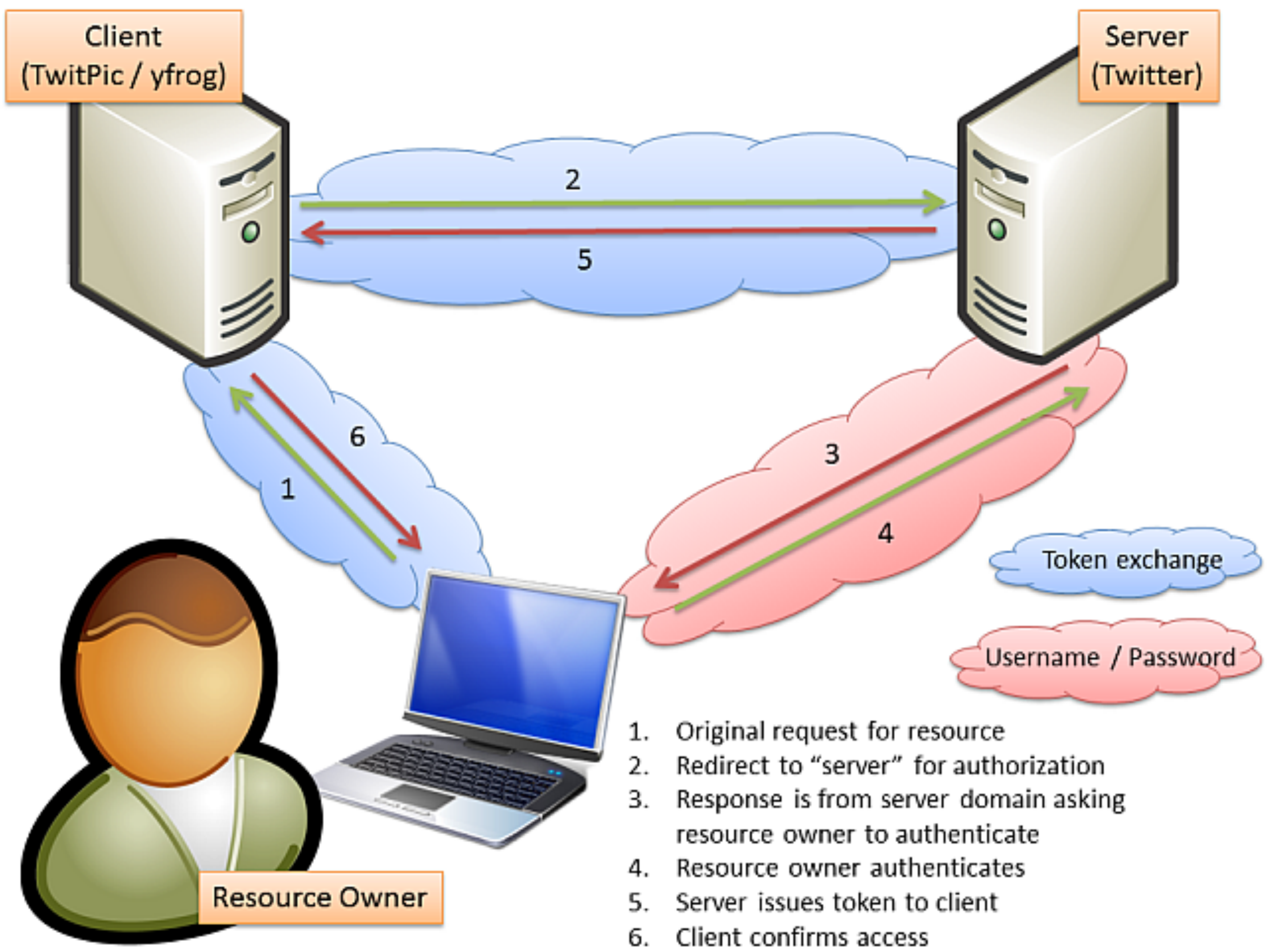
OAuth 1

http://hueniverse.com/oauth/
https://tools.ietf.org/html/rfc5849

- "An open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications."
- The idea is to allow for a safe way to share privileges without divulging private credentials
    - Give XPTO Application permission to post to your Twitter account without having to trust the developers of XPTO with your username/password and while being able to unilaterally revoke privileges.

OAuth 1

http://hueniverse.com/oauth/
https://tools.ietf.org/html/rfc5849



1. Original request for resource
2. Redirect to "server" for authorization
3. Response is from server domain asking resource owner to authenticate
4. Resource owner authenticates
5. Server issues token to client
6. Client confirms access

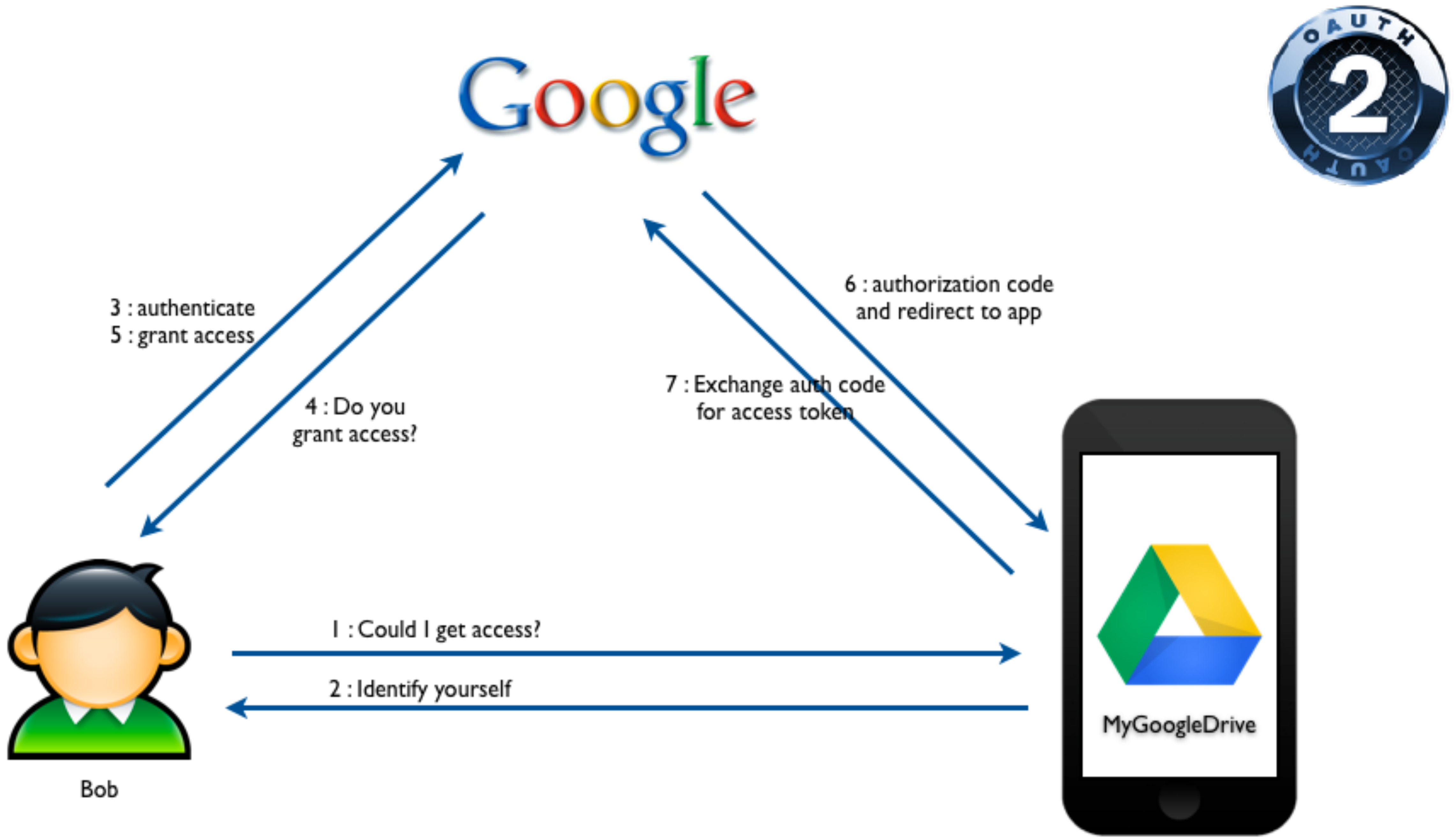
OAuth 1

http://hueniverse.com/oauth/
https://tools.ietf.org/html/rfc5849

- After the "OAuth dance" is concluded, client application has two sets of keys:
  - one that uses to identify itself as a valid application (api_key, api_secret)
  - one that uses to identify the user it wants to access (token, token_secret)
- You can revoke access at any time by letting the token provider that a given app is no long authorized (invalidating token and token_secret).

OAuth 2

https://tools.ietf.org/html/rfc6749
https://tools.ietf.org/html/rfc6750

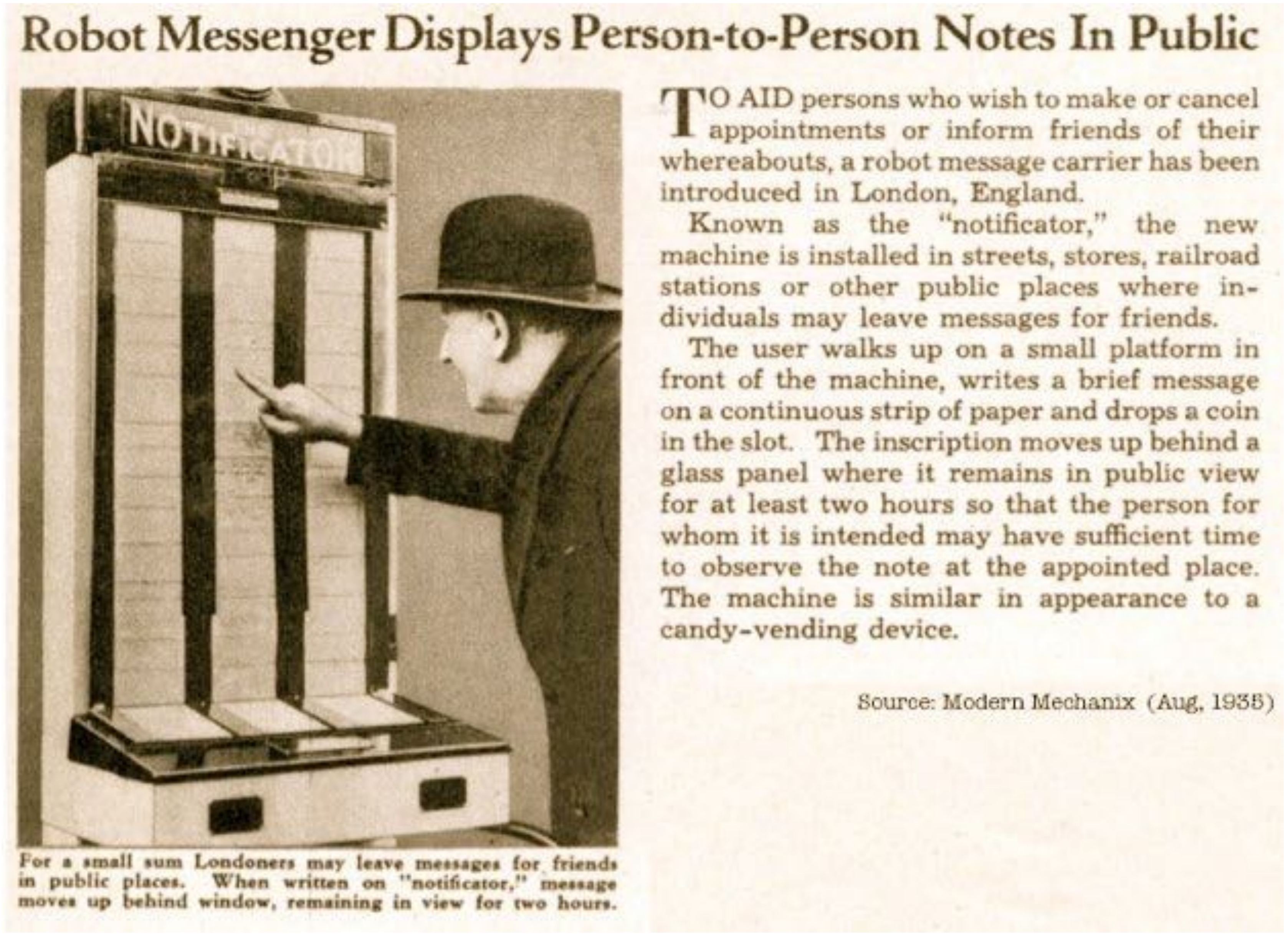


OAuth 2

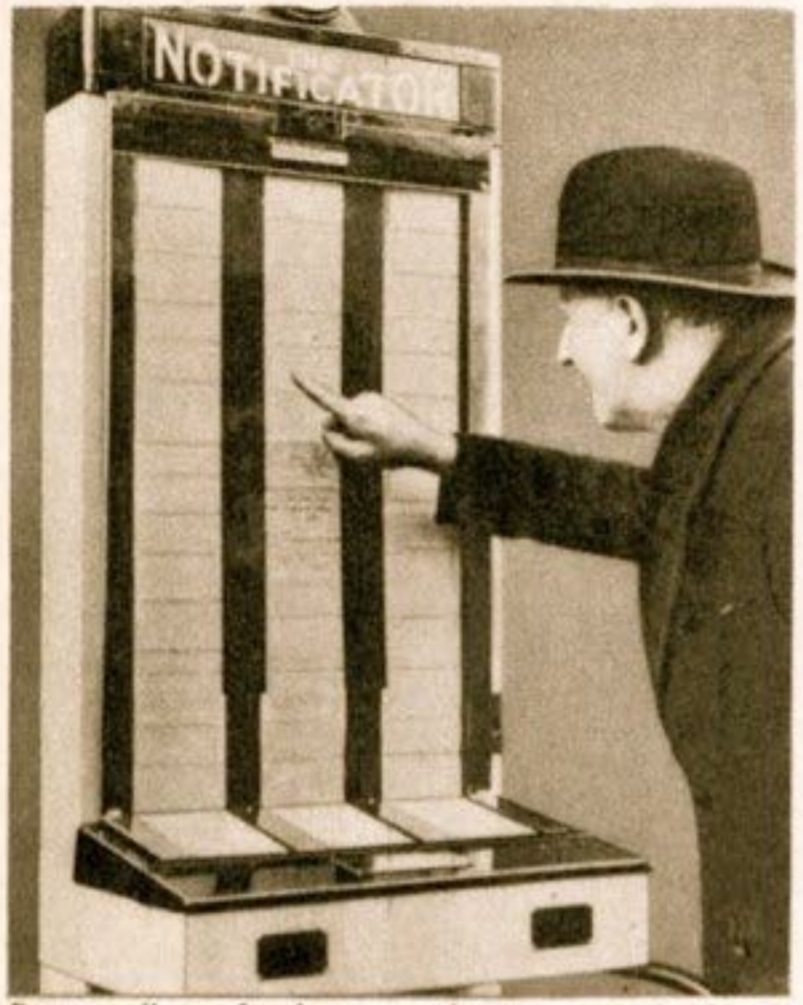
https://tools.ietf.org/html/rfc6749
https://tools.ietf.org/html/rfc6750

- Latest version of OAuth protocol
    - Similar "dance" required
    - Allows for "bearer tokens" - access is given to anyone able to provide a valid token without any further restrictions or authentication
    - access tokens are provided along with the request for the resource through a secure connection
    - tokens can expire automatically
- We will use both OAuth and OAuth2 over the next few days

Twitter



Robot Messenger Displays Person-to-Person Notes In Public



For a small sum Londoners may leave messages for friends in public places. When written on "notificator," message moves up behind window, remaining in view for two hours.

TO AID persons who wish to make or cancel appointments or inform friends of their whereabouts, a robot message carrier has been introduced in London, England.

Known as the "notificator," the new machine is installed in streets, stores, railroad stations or other public places where individuals may leave messages for friends.

The user walks up on a small platform in front of the machine, writes a brief message on a continuous strip of paper and drops a coin in the slot. The inscription moves up behind a glass panel where it remains in public view for at least two hours so that the person for whom it is intended may have sufficient time to observe the note at the appointed place. The machine is similar in appearance to a candy-vending device.

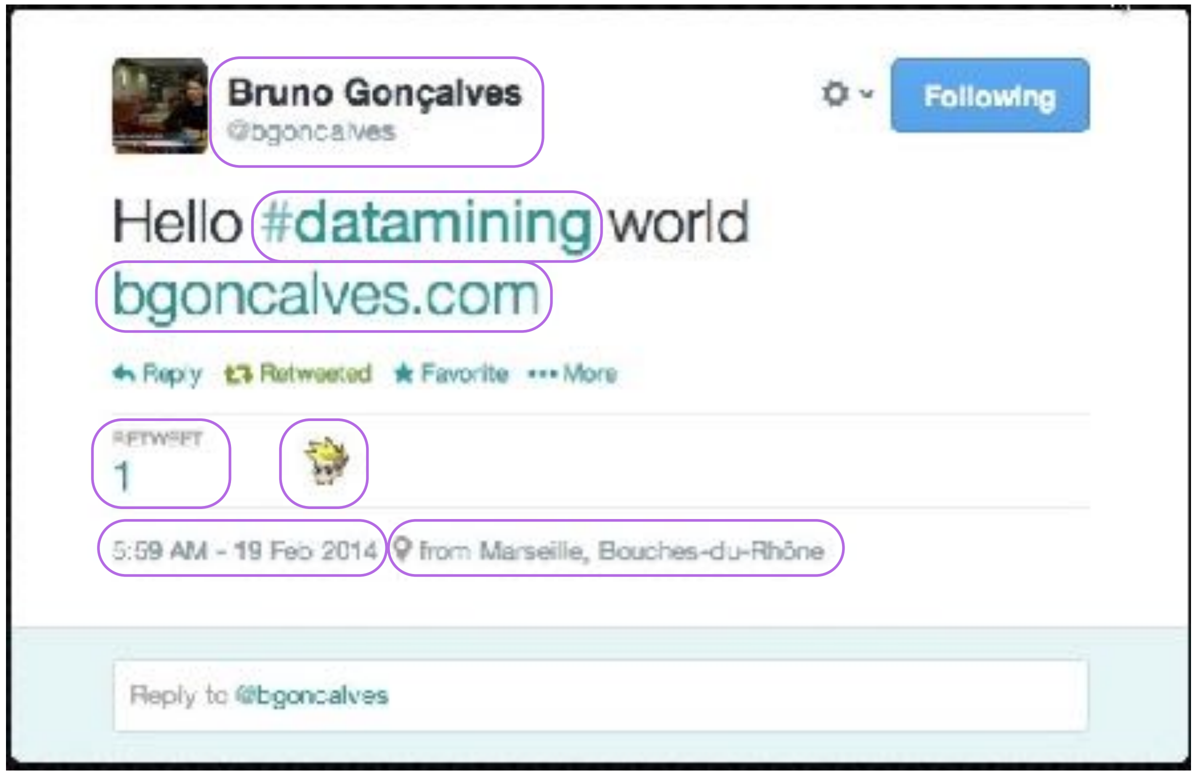
Source: Modern Mechanix (Aug, 1935)

twitter

@bgoncalves

www.bgoncalves.com

Anatomy of a Tweet


Bruno Gonçalves
@bgoncalves

Following

Hello #datamining world
bgoncalves.com

Reply Retweeted Favorite More

RETWEET
1

5:59 AM - 19 Feb 2014 from Marseille, Bouches-du-Rhône

Reply to @bgoncalves

Anatomy of a Tweet

```
[u'contributors',
 u'truncated',
 u'text',
 u'in_reply_to_status_id',
 u'id',
 u'favorite_count',
 u'source',
 u'retweeted',
 u'coordinates',
 u'entities',
 u'in_reply_to_screen_name',
 u'in_reply_to_user_id',
 u'retweet_count',
 u'id_str',
 u'favorited',
 u'user',
 u'geo',
 u'in_reply_to_user_id_str',
 u'possibly_sensitive',
 u'lang',
 u'created_at',
 u'in_reply_to_status_id_str',
 u'place',
 u'metadata']
```

Anatomy of a Tweet

```
[u'contributors',
 u'truncated',
 u'text',
 u'in_reply_to_status_id',
 u'id',
 u'favorite_count',
 u'source',
 u'retweeted',
 u'coordinates',
 u'entities',
 u'in_reply_to_screen_name',
 u'in_reply_to_user_id',
 u'retweet_count',
 u'id_str',
 u'favorited',
 u'user',
 u'geo',
 u'in_reply_to_user_id_str',
 u'possibly_sensitive',
 u'lang',
 u'created_at',
 u'in_reply_to_status_id_str',
 u'place',
 u'metadata']
```

```
[u'follow_request_sent',
 u'profile_use_background_image',
 u'default_profile_image',
 u'id',
 u'profile_background_image_url_https',
 u'verified',
 u'profile_text_color',
 u'profile_image_url_https',
 u'profile_sidebar_fill_color',
 u'entities',
 u'followers_count',
 u'profile_sidebar_border_color',
 u'id_str',
 u'profile_background_color',
 u'listed_count',
 u'is_translation_enabled',
 u'utc_offset',
 u'statuses_count',
 u'description',
 u'friends_count',
 u'location',
 u'profile_link_color',
 u'profile_image_url',
 u'following',
 u'geo_enabled',
 u'profile_banner_url',
 u'profile_background_image_url',
 u'screen_name',
 u'lang',
 u'profile_background_tile',
 u'favourites_count',
 u'name',
 u'notifications',
 u'url',
 u'created_at',
 u'contributors_enabled',
 u'time_zone',
 u'protected',
 u'default_profile',
 u'is_translator']
```

Anatomy of a Tweet

```
[u'contributors',
 u'truncated',
 u'text',
 u'in_reply_to_status_id',
 u'id',
 u'favorite_count',
 u'source',
 u'retweeted',
 u'coordinates',
 u'entities',
 u'in_reply_to_screen_name',
 u'in_reply_to_user_id',
 u'retweet_count',
 u'id_str',
 u'favorited',
 u'user',
 u'geo',
 u'in_reply_to_user_id_str',
 u'possibly_sensitive',
 u'lang',
 u'created_at',
 u'in_reply_to_status_id_str',
 u'place',
 u'metadata']
```

```
u"I'm at Terminal Rodovi\xe1rio de Feira de Santana
 (Feira de Santana, BA) http://t.co/WirvdHwYMq"
```

```
u'<a href="http://foursquare.com" rel="nofollow">
  foursquare</a>'
```

```
[u'symbols',
 u'user_mentions',
 u'hashtags',
 u'urls']
```

```
[u'type',
 u'coordinates']
```

Anatomy of a Tweet

```
[u'contributors',
 u'truncated',
 u'text',
 u'in_reply_to_status_id',
 u'id',
 u'favorite_count',
 u'source',
 u'retweeted',
 u'coordinates',
 u'entities',
 u'in_reply_to_screen_name',
 u'in_reply_to_user_id',
 u'retweet_count',
 u'id_str',
 u'favorited',
 u'user',
 u'geo',
 u'in_reply_to_user_id_str',
 u'possibly_sensitive',
 u'lang',
 u'created_at',
 u'in_reply_to_status_id_str',
 u'place',
 u'metadata']
```

```
u"I'm at Terminal Rodovi\xe1rio de Feira de Santana
 (Feira de Santana, BA) http://t.co/WirvdHwYMq"
```

```
u'<a href="http://foursquare.com" rel="nofollow">
  foursquare</a>'
```

```
[u'symbols',
 u'user_mentions',
 u'hashtags',
 u'urls']
```

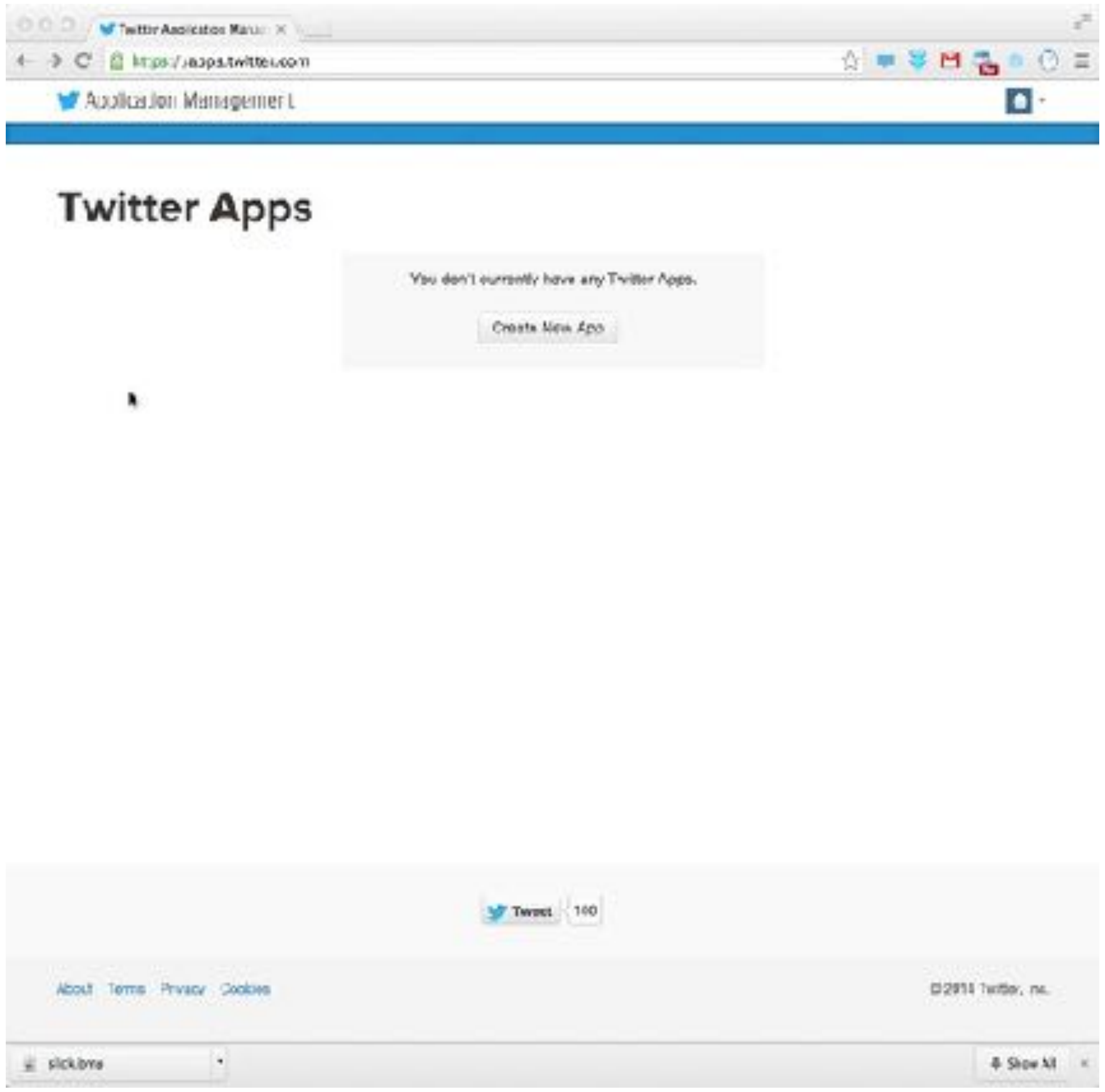
```
{u'display_url': u'4sq.com/1k5MeYF',
 u'expanded_url': u'http://4sq.com/1k5MeYF',
 u'indices': [70, 92],
 u'url': u'http://t.co/WirvdHwYMq'}
```

```
[u'type',
 u'coordinates']
```

@bgoncalves

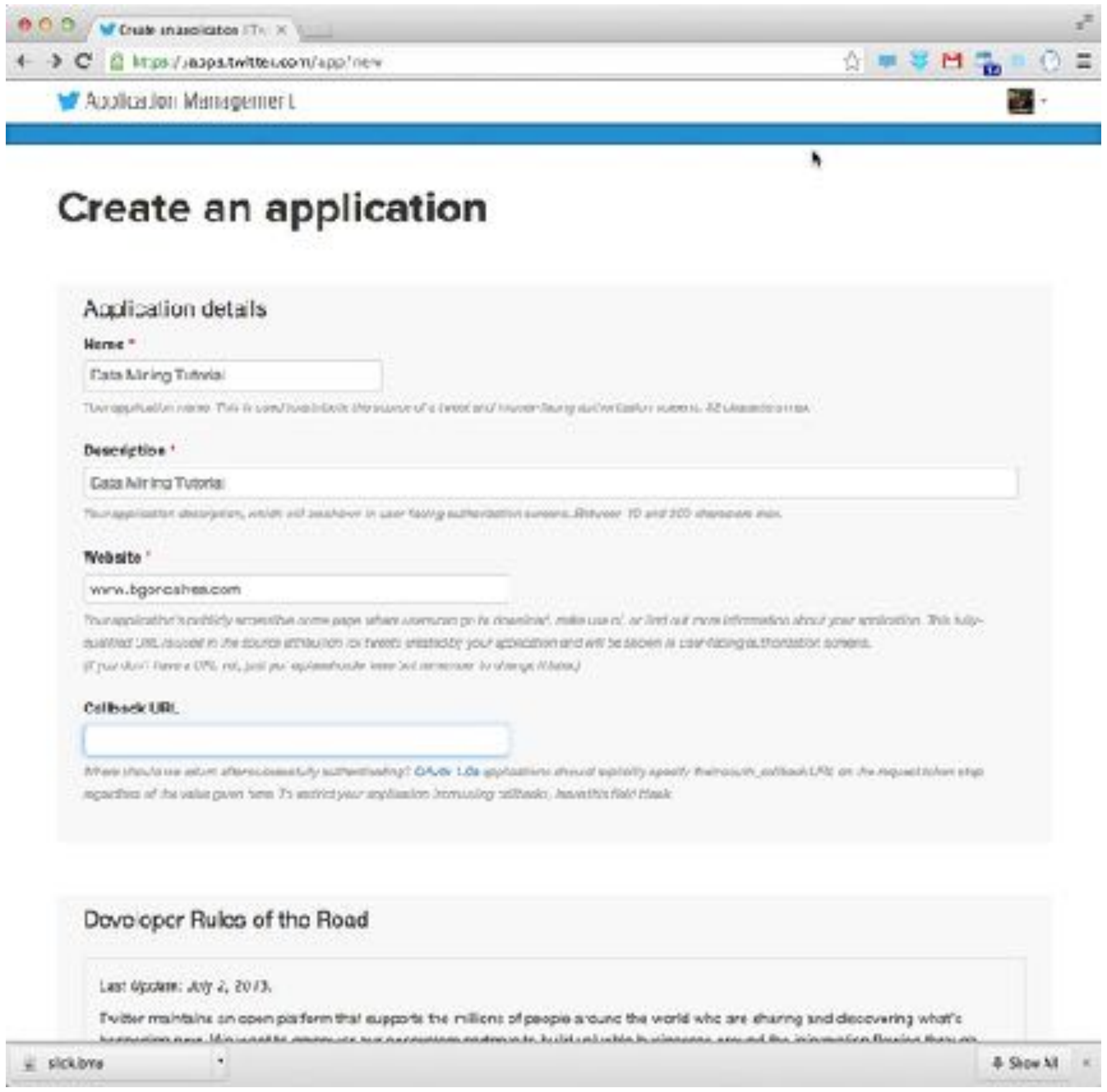
www.bgoncalves.com

Registering an Application



You don't currently have any Twitter Apps.

Registering an Application


https://apps.twitter.com/app/new

Application Management

Create an application

Application details

Name *

Data Mining Tutorial

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

Data Mining Tutorial

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *

www.bgoncalves.com

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

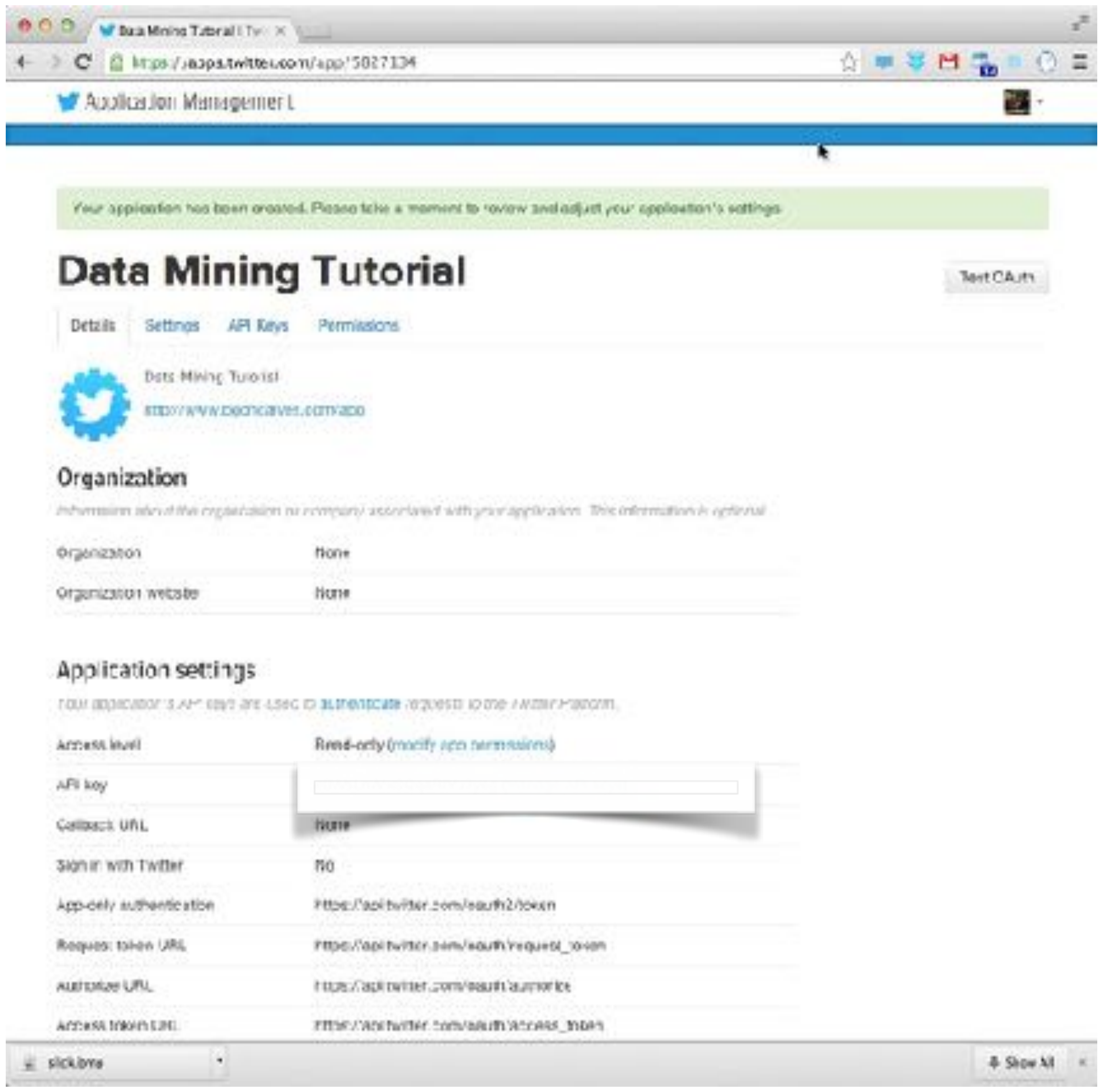
Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

Developer Rules of the Road

Last Update: July 2, 2013.

Twitter maintains an open platform that supports the millions of people around the world who are sharing and discovering what's

Registering an Application


https://apps.twitter.com/app/5027134

Application Management

Your application has been created. Please take a moment to review and adjust your application's settings.

Data Mining Tutorial

Test OAuth

Details | Settings | API Keys | Permissions

Data Mining Tutorial
http://www.bgoncalves.com/app

Organization

Information about the organization or company associated with your application. This information is optional.

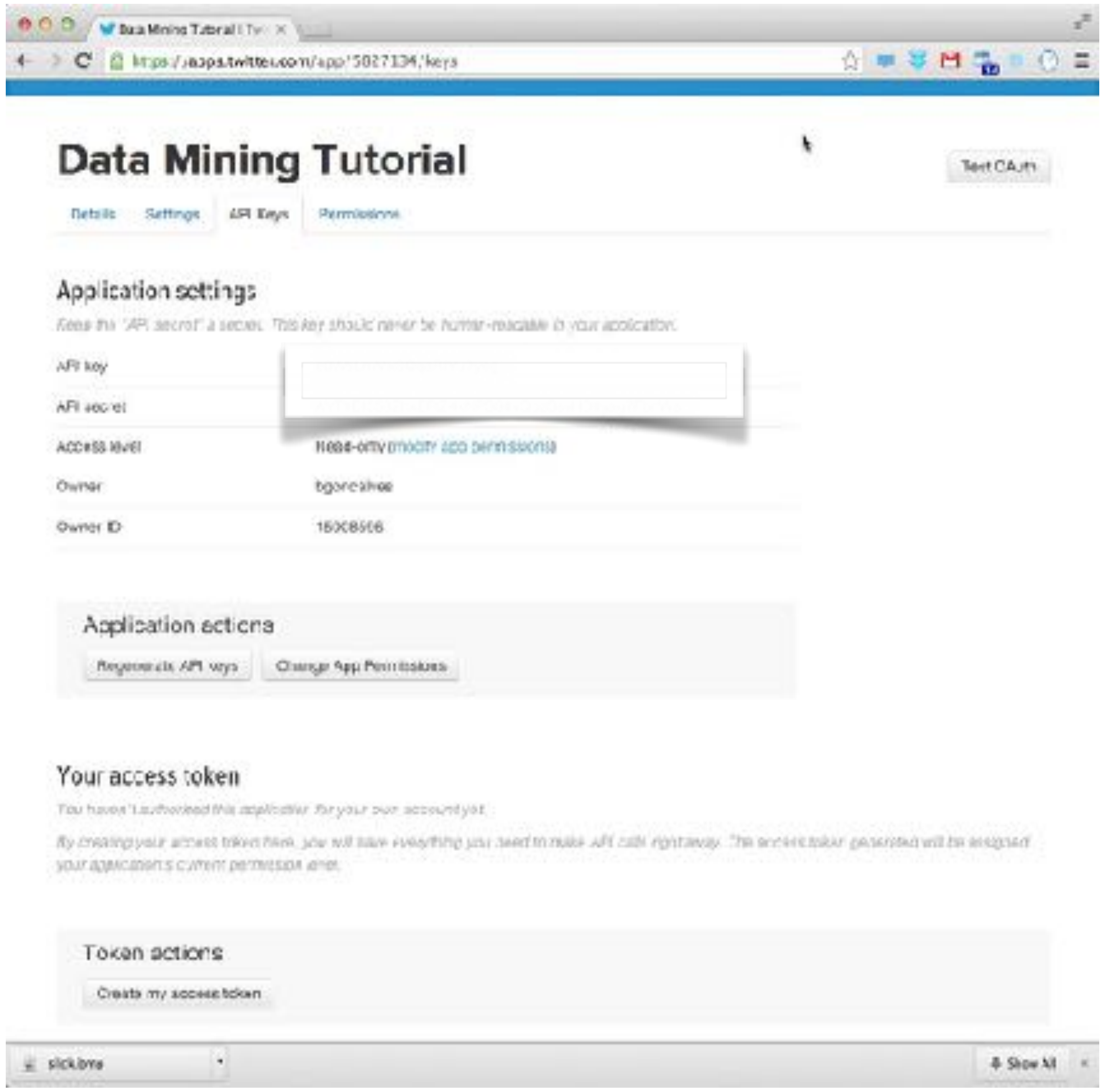
Organization	None
Organization website	None

Application settings

Your application's API keys are used to authenticate requests to the Twitter Platform.

Access level	Read-only (modify app permissions)
API key	
Callback URL	None
Sign in with Twitter	No
App-only authentication	https://api.twitter.com/oauth2/token
Request token URL	https://api.twitter.com/oauth/request_token
Authorize URL	https://api.twitter.com/oauth/authorize
Access token URL	https://api.twitter.com/oauth/access_token

Registering an Application



https://apps.twitter.com/app/5027134/keys

Data Mining Tutorial

Application settings

Keep the "API secret" a secret. This key should never be human-readable in your application.

API key	[redacted]
API secret	[redacted]
Access level	Read-only (modify app permissions)
Owner	bgoncalves
Owner ID	15008506

Application actions

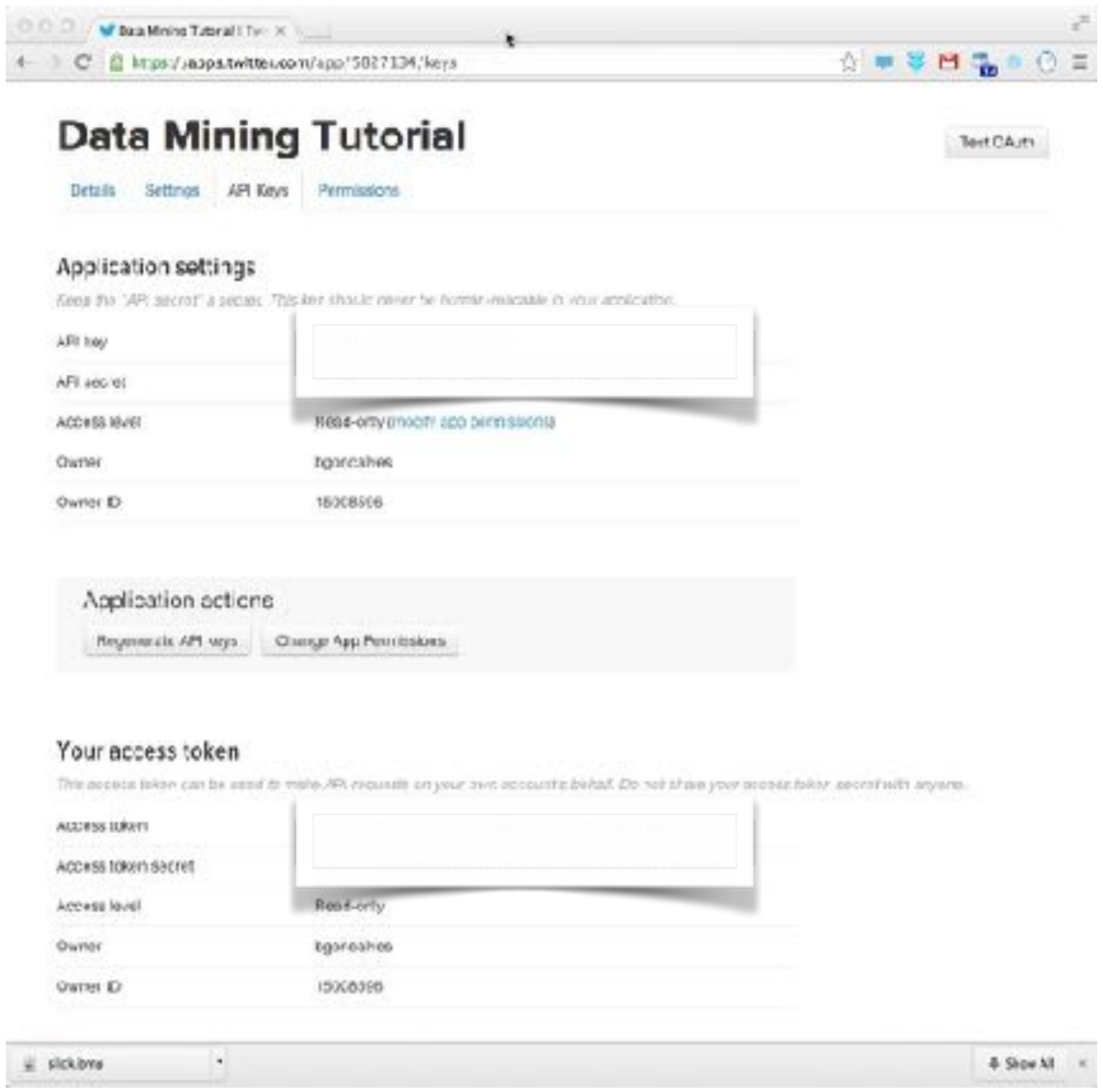
Your access token

You haven't authorized this application for your own account yet.

By creating your access token here, you will have everything you need to make API calls right away. The access token generated will be assigned your application's current permission level.

Token actions

Registering an Application



https://apps.twitter.com/app/5027134/keys

Data Mining Tutorial

Test OAuth

Details | Settings | API Keys | Permissions

Application settings

Keep the "API secret" a secret. This key should never be human-readable in your application.

API key	[redacted]
API secret	
Access level	Read-only (modify app permissions)
Owner	bgoncalves
Owner ID	15008506

Application actions

Regenerate API keys | Change App Permissions

Your access token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

Access token	[redacted]
Access token secret	
Access level	Read-only
Owner	bgoncalves
Owner ID	15008506

@bgoncalves www.bgoncalves.com

API Basics

https://dev.twitter.com/docs

- The twitter module provides the oauth interface. We just need to provide the right credentials.
- Best to keep the credentials in a dict and parametrize our calls with the dict key. This way we can switch between different accounts easily.
- .Twitter(auth) takes an OAuth instance as argument and returns a Twitter object that we can use to interact with the API
- Twitter methods mimic API structure
- 4 basic types of objects:
    - Tweets
    - Users
    - Entities

Authenticating with the API

```
import tweepy
from twitter_accounts import accounts

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

twitter_api = twitter.Twitter(auth=auth)
```

- In the remainder of this course, the accounts dict will live inside the twitter_accounts.py file
- 4 basic types of objects:
  - Tweets
  - Users
  - Entities
  - Places

@bgoncalves

twitter_authentication.py

Searching for Tweets

https://dev.twitter.com/docs/api/1.1/get/search/tweets

- .search.tweets(query, count)
    - query is the content to search for
    - count is the maximum number of results to return
- returns dict with a list of "statuses" and "search_metadata"

```
{u'completed_in': 0.027,
 u'count': 15,
 u'max_id': 438088492577345536,
 u'max_id_str': u'438088492577345536',
 u'next_results': u'?max_id=438088485145034752&q=soccer&include_entities=1',
 u'query': u'soccer',
 u'refresh_url': u'?since_id=438088492577345536&q=soccer&include_entities=1',
 u'since_id': 0,
 u'since_id_str': u'0'}
```

- search_results["search_metadata"]["next_results"] can be used to get the next page of results

Searching for Tweets

https://dev.twitter.com/docs/api/1.1/get/search/tweets

```
query = "instagram"
count = 200

search_results = twitter_api.search.tweets(q=query, count=count)

statuses = search_results["statuses"]
tweet_count = 0

while True:
    try:
        next_results = search_results["search_metadata"]["next_results"]

        args = dict(parse.parse_qsl(next_results[1:]))

        search_results = twitter_api.search.tweets(**args)
        statuses = search_results["statuses"]

        print(search_results["search_metadata"]["max_id"])

        for tweet in statuses:
            tweet_count += 1

            if tweet_count % 10000 == 0:
                print(tweet_count,file=sys.stderr)

            print (tweet["text"])
    except:
        break
```

twitter_search.py

Streaming data

https://dev.twitter.com/docs/api/1.1/post/statuses/filter

- The Streaming api provides realtime data, subject to filters
- Use TwitterStream instead of Twitter object (.TwitterStream(auth=twitter_api.auth))
- .status.filter(track=q) will return tweets that match the query q in real time
- Returns generator that you can iterate over

Streaming data

https://dev.twitter.com/docs/api/1.1/post/statuses/filter

```
import twitter
from twitter_accounts import accounts

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

stream_api = twitter.TwitterStream(auth=auth)

query = "bieber"

stream_results = stream_api.statuses.filter(track=query)

for tweet in stream_results:
    print(tweet["text"])
```

@bgoncalves

twitter_stream.py

User profiles

https://dev.twitter.com/docs/api/1.1/get/users/lookup

- .users.lookup() returns user profile information for a list of user_ids or screen_names

- list should be comma separated and provided as a string

```python
import twitter
from twitter_accounts import accounts

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

twitter_api = twitter.Twitter(auth=auth)

screen_names = ",".join(["diunito", "giaruffo"])

search_results = twitter_api.users.lookup(screen_name=screen_names)

for user in search_results:
  print(user["screen_name"], "has", user["followers_count"], "followers")
```

@bgoncalves

twitter_users.py

Social Connections

https://dev.twitter.com/docs/api/1.1/get/friends/ids
https://dev.twitter.com/docs/api/1.1/get/followers/ids

- .friends.ids() and .followers.ids() returns a list of up to 5000 of a users friends or followers for a given screen_name or user_id
- result is a dict containing multiple fields:

```
[u'next_cursor_str',
 u'previous_cursor',
 u'ids',
 u'next_cursor',
 u'previous_cursor_str']
```

- ids are contained in results["ids"].
- results["next_cursor"] allows us to obtain the next page of results.
- .friends.ids(screen_name=screen_name, cursor=results["next_cursor"]) will return the next page of results
- cursor=0 means no more results

Social Connections

https://dev.twitter.com/docs/api/1.1/get/friends/ids
https://dev.twitter.com/docs/api/1.1/get/followers/ids

```
import twitter
from twitter_accounts import accounts

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

twitter_api = twitter.Twitter(auth=auth)

screen_name = "stephen_wolfram"

cursor = -1
followers = []

while cursor != 0:
    result = twitter_api.followers.ids(screen_name=screen_name,
cursor=cursor)

    followers += result["ids"]
    cursor = result["next_cursor"]

print("Found", len(followers), "Followers")
```

@bgoncalves

twitter_followers.py

User Timeline

https://dev.twitter.com/docs/api/1.1/get/statuses/user_timeline

- .statuses.user_timeline() returns a set of tweets posted by a single user
- Important options:
    - include_rts='true' to Include retweets by this user
    - count=200 number of tweets to return in each call
    - trim_user='true' to not include the user information (save bandwidth and processing time)
    - max_id=1234 to include only tweets with an id lower than 1234
- Returns at most 200 tweets in each call. Can get all of a users tweets (up to 3200) with multiple calls using max_id

User Timeline

https://dev.twitter.com/docs/api/1.1/get/statuses/user_timeline

```
import twitter
from twitter_accounts import accounts

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

twitter_api = twitter.Twitter(auth=auth)
screen_name = "BarackObama"

args = { "count" : 200,
         "trim_user": "true",
         "include_rts": "true"
        }

tweets = twitter_api.statuses.user_timeline(screen_name = screen_name, **args)
tweets_new = tweets

while len(tweets_new) > 0:
    max_id = tweets[-1]["id"] - 1
    tweets_new = twitter_api.statuses.user_timeline(screen_name = screen_name, max_id=max_id, **args)
    tweets += tweets_new

print(“Found", len(tweets), “tweets")
```

@bgoncalves

twitter_timeline.py

Social Interactions

```
import twitter
from twitter_accounts import accounts

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

twitter_api = twitter.Twitter(auth=auth)
screen_name = "BarackObama"
args = { "count" : 200, "trim_user": "true", "include_rts": "true"}

tweets = twitter_api.statuses.user_timeline(screen_name=screen_name, **args)
tweets_new = tweets

while len(tweets_new) > 0:
    max_id = tweets[-1]["id"] - 1
    tweets_new = twitter_api.statuses.user_timeline(screen_name=screen_name, max_id=max_id, **args)
    tweets += tweets_new

user = tweets[0]["user"]["id"]

for tweet in tweets:
    if "retweeted_status" in tweet:
        print(user, "->", tweet["retweeted_status"]["user"]["id"])
    elif tweet["in_reply_to_user_id"]:
        print(tweet["in_reply_to_user_id"], "->", user)
```

twitter_interactions.py

Streaming Geocoded data

https://dev.twitter.com/streaming/overview/request-parameters#locations

- The Streaming api provides realtime data, subject to filters
- Use TwitterStream instead of Twitter object (.TwitterStream(auth=twitter_api.auth))
- .status.filter(track=q) will return tweets that match the query q in real time
- Returns generator that you can iterate over
- .status.filter(locations=bb) will return tweets that occur within the bounding box bb in real time
- bb is a comma separated pair of lon/lat coordinates.
  - -180,-90,180,90 - World
  - -74,40,-73,41 - NYC

Streaming Geocoded data

https://dev.twitter.com/streaming/overview/request-parameters#locations

```python
import twitter
from twitter_accounts import accounts
import sys
import gzip

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

stream_api = twitter.TwitterStream(auth=auth)

query = "-74,40,-73,41"  # NYC
stream_results = stream_api.statuses.filter(locations=query)
tweet_count = 0

fp = gzip.open("NYC.json.gz", "a")

for tweet in stream_results:
    try:
        tweet_count += 1
        print (tweet_count, tweet["id"])
        print(tweet, file=fp)
    except:
        pass

    if tweet_count % 10000 == 0:
        print(tweet_count, file=sys.stderr)
        break
```

@bgoncalves

location_twitter.py

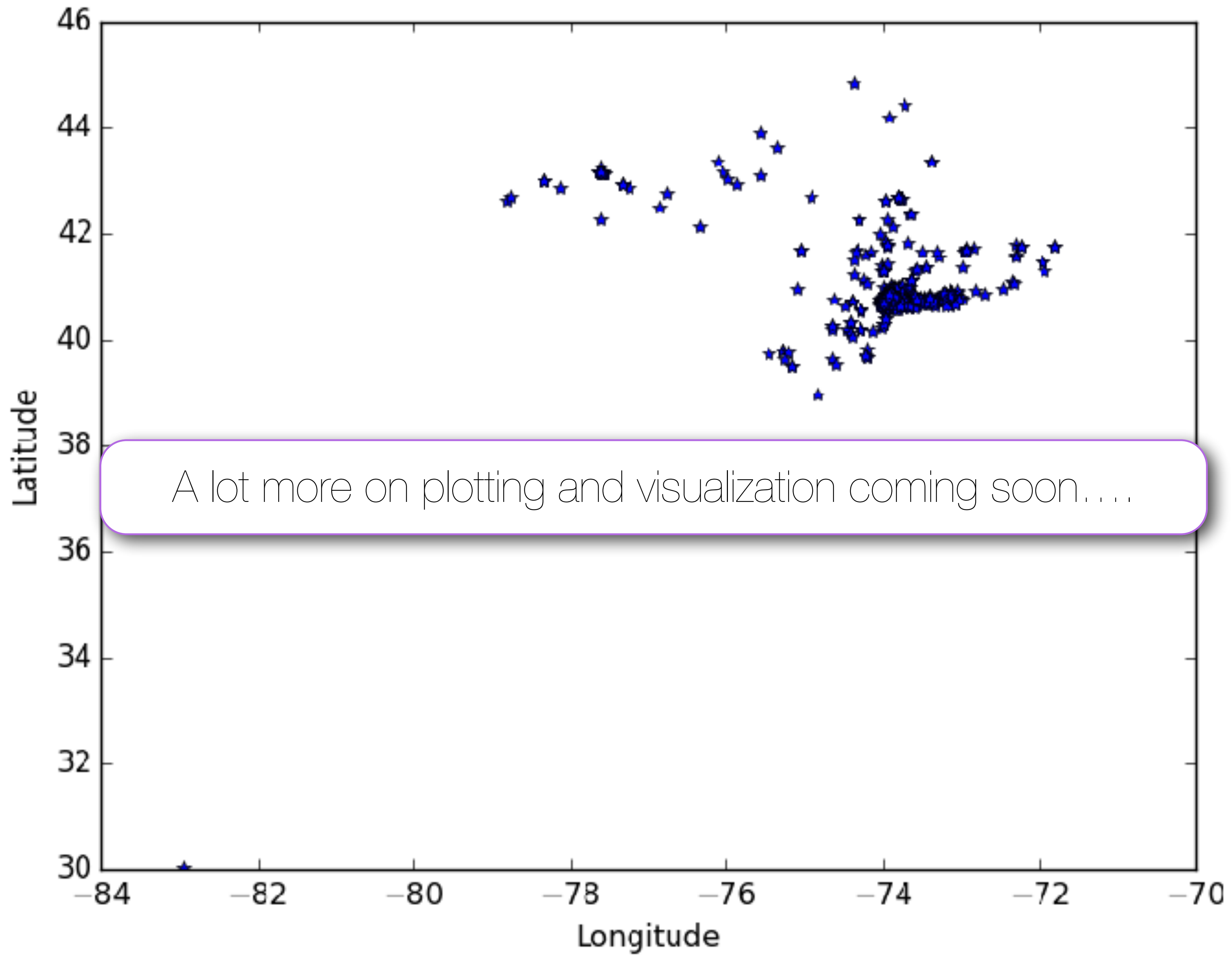
Plotting geolocated tweets

```
import sys
import gzip
import matplotlib.pyplot as plt

x = []
y = []

line_count = 0

try:
    for line in gzip.open(sys.argv[1]):
        try:
            tweet = eval(line.strip())
            line_count += 1

            if "coordinates" in tweet and tweet["coordinates"] is not None:
                x.append(tweet["coordinates"]["coordinates"][0])
                y.append(tweet["coordinates"]["coordinates"][1])
        except:
            pass
except:
    pass

print("Read", line_count, "and found", len(x), "geolocated tweets", file=sys.stderr)

plt.plot(x, y, '*')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.savefig(sys.argv[1] + '.png')
plt.close()
```

plot_tweets.py

@bgoncalves

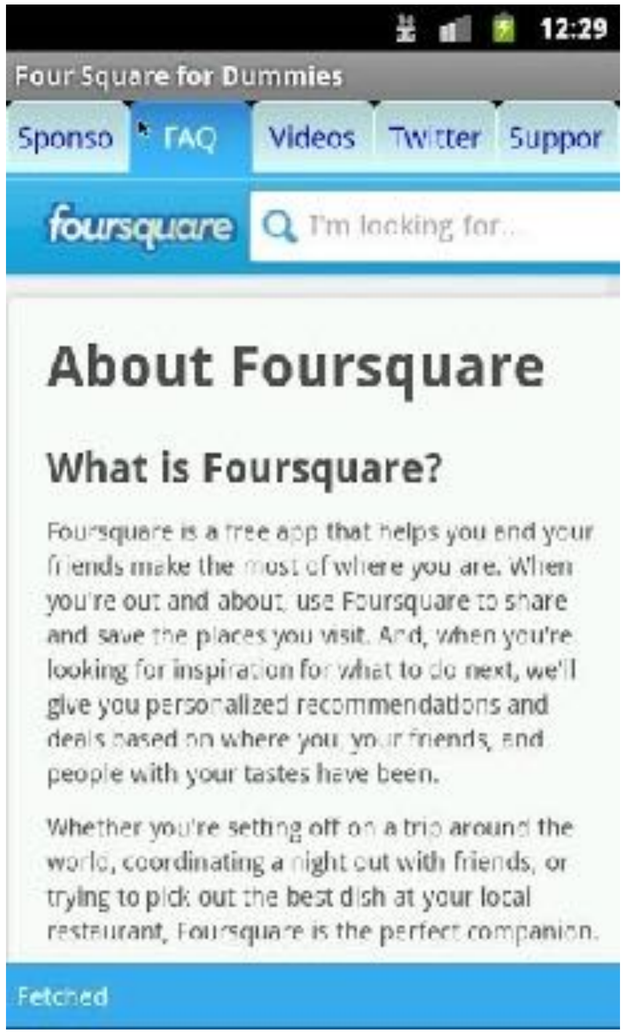
Plotting geolocated tweets



A lot more on plotting and visualization coming soon….

Foursquare

Foursquare



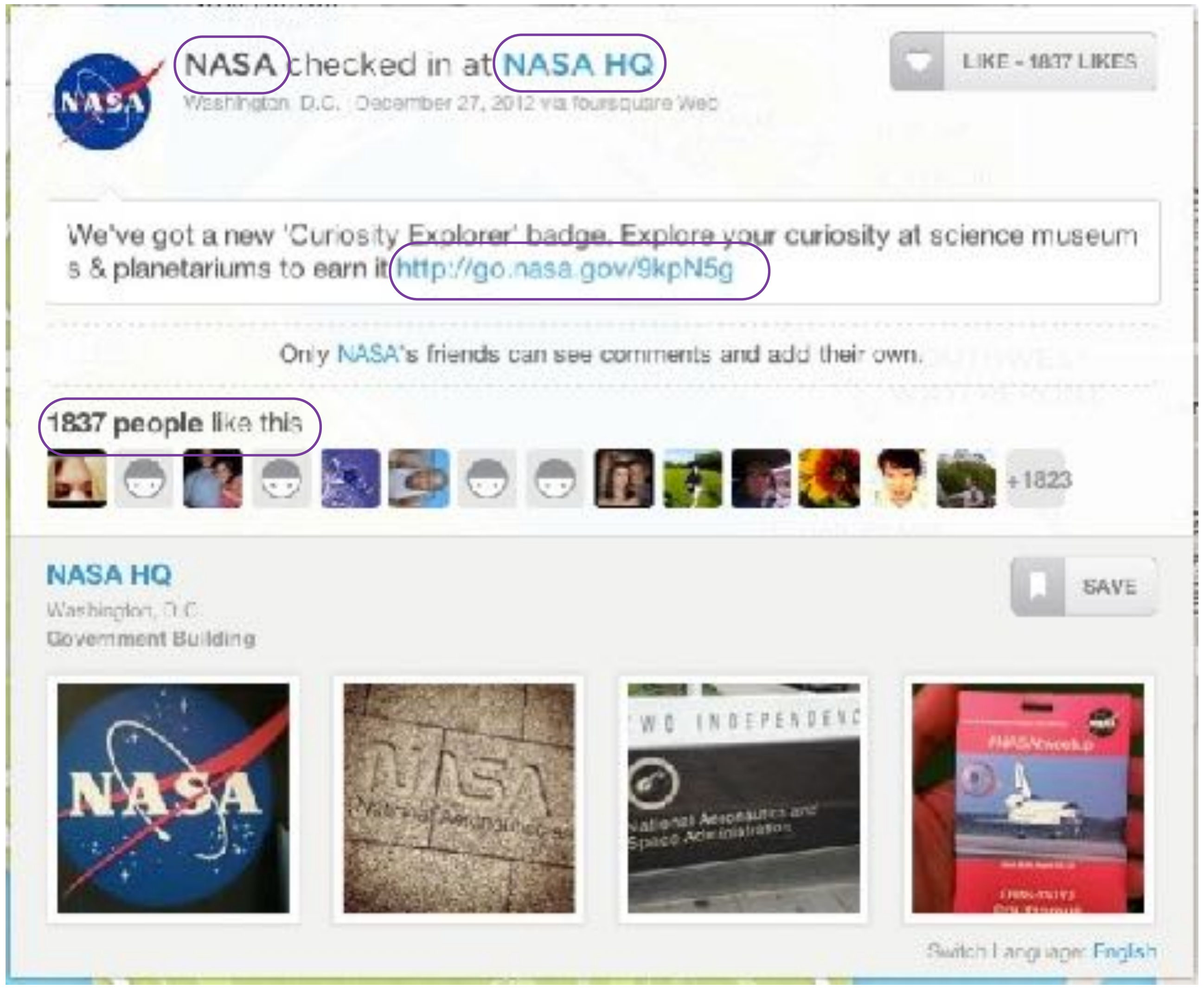
About Foursquare

What is Foursquare?

Foursquare is a free app that helps you and your friends make the most of where you are. When you're out and about, use Foursquare to share and save the places you visit. And, when you're looking for inspiration for what to do next, we'll give you personalized recommendations and deals based on where you, your friends, and people with your tastes have been.

Whether you're setting off on a trip around the world, coordinating a night out with friends, or trying to pick out the best dish at your local restaurant, Foursquare is the perfect companion.

Anatomy of a Checkin



NASA checked in at NASA HQ

Washington, D.C. December 27, 2012 via foursquare Web

LIKE - 1837 LIKES

We've got a new 'Curiosity Explorer' badge. Explore your curiosity at science museums & planetariums to earn it http://go.nasa.gov/9kpN5g

Only NASA's friends can see comments and add their own.

1837 people like this

+1823

NASA HQ

Washington, D.C.

Government Building

SAVE

Switch Language: English

@bgoncalves www.bgoncalves.com

Anatomy of a Checkin

```
[u'venue',
 u'like',
 u'photos',
 u'source',
 u'visibility',
 u'entities',
 u'shout',
 u'timeZoneOffset',
 u'type',
 u'id',
 u'createdAt',
 u'likes']
```

Anatomy of a Checkin

```
[u'venue',
 u'like',
 u'photos',
 u'source',
 u'visibility',
 u'entities',
 u'shout',
 u'timeZoneOffset',
 u'type',
 u'id',
 u'createdAt',
 u'likes']
```

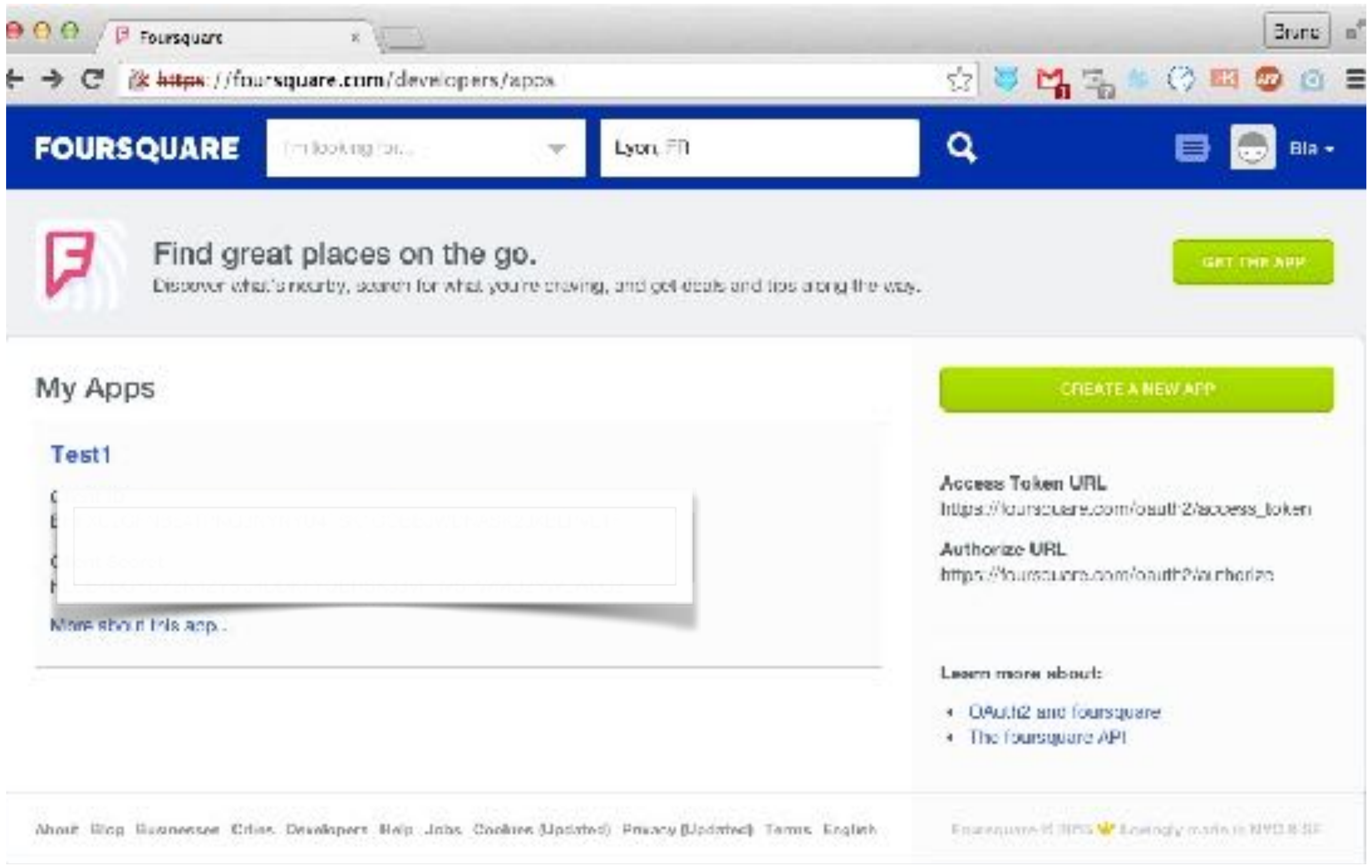
```
[u'verified',
 u'name',
 u'url',
 u'like',
 u'contact',
 u'location',
 u'stats',
 u'id',
 u'categories',
 u'likes']
```

```
[{u'indices': [112, 137],
  u'object': {u'url': u'http://go.nasa.gov/9kpN5g'},
  u'type': u'url'}]
```

```
u"We've got a new 'Curiosity Explorer' badge. Explore your
curiosity at science museums & planetariums to earn it http://
go.nasa.gov/9kpN5g"
```

```
{u'count': 1837,
 u'groups': [{u'count': 1837, u'items': [], u'type': u'others'}],
 u'summary': u'1837 likes'}
```

Registering An Application

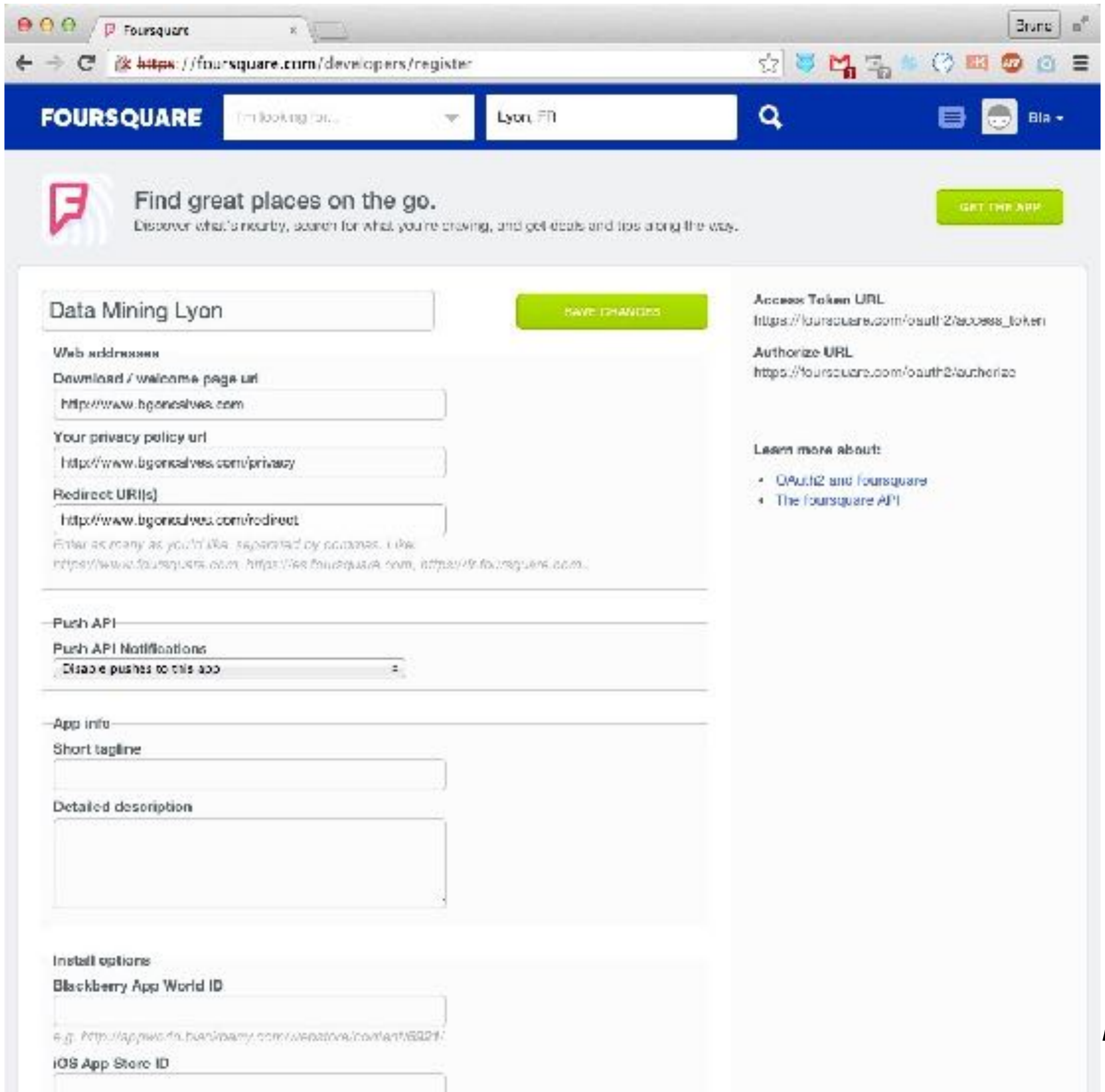


Access Token URL
https://foursquare.com/oauth2/access_token

Authorize URL
https://foursquare.com/oauth2/authorize

@bgoncalves www.bgoncalves.com

Registering An Application


Foursquare
https://foursquare.com/developers/register
FOURSQUARE
I'm looking for...
Lyon, FR
Bla
Find great places on the go.
Discover what's nearby, search for what you're craving, and get deals and tips along the way.
GET THE APP
SAVE CHANGES

Data Mining Lyon

Web addresses

Download / welcome page url

http://www.bgoncalves.com

Your privacy policy url

http://www.bgoncalves.com/privacy

Redirect URI(s)

http://www.bgoncalves.com/redirect

Enter as many as you'd like, separated by commas. Like: https://www.foursquare.com, https://es.foursquare.com, https://fr.foursquare.com.

Push API

Push API Notifications

Disable pushes to this app

App info

Short tagline

Detailed description

Install options

Blackberry App World ID

e.g. http://appworld.blackberry.com/webstore/content/6921/

iOS App Store ID

Access Token URL

https://foursquare.com/oauth2/access_token

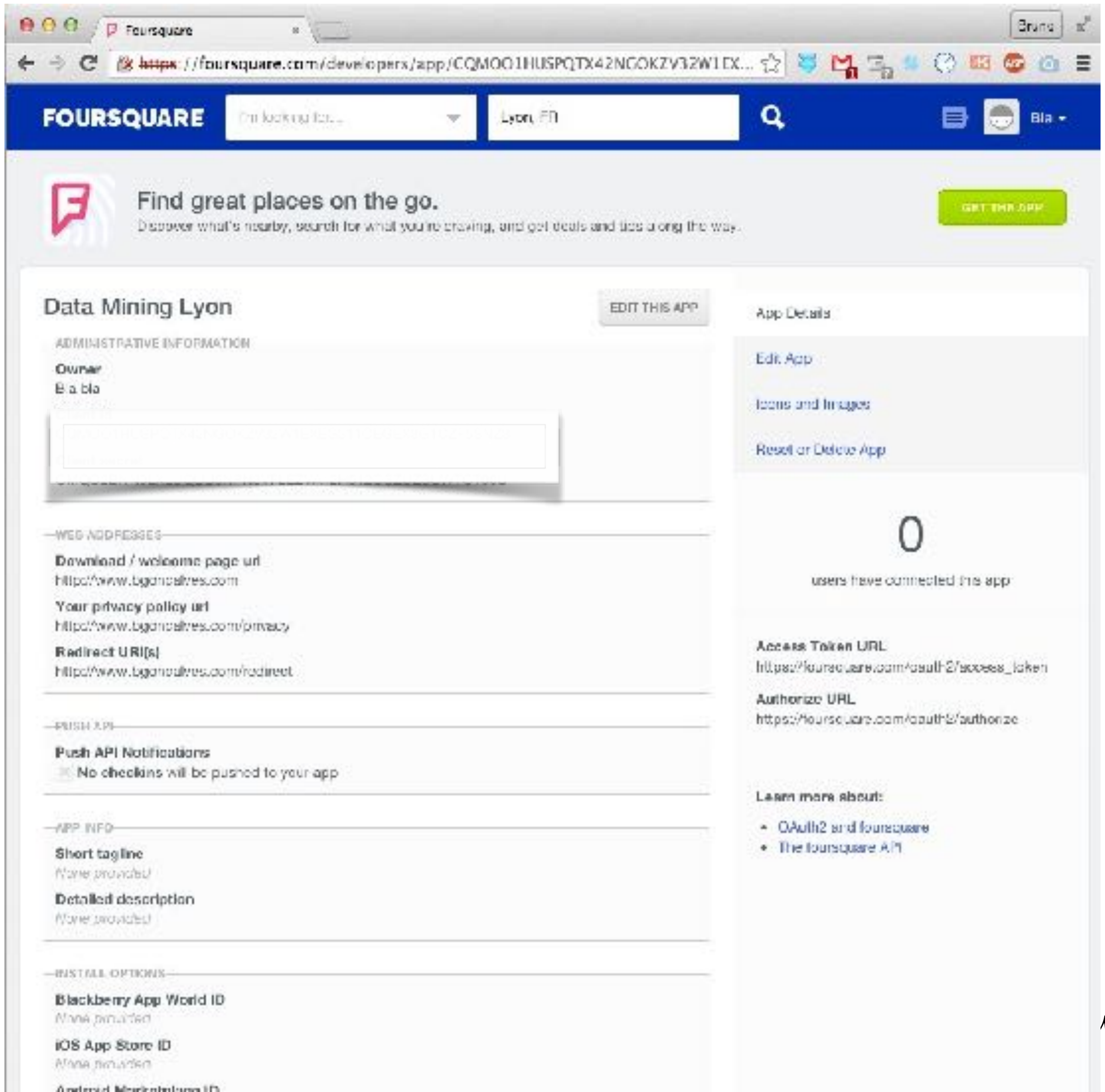
Authorize URL

https://foursquare.com/oauth2/authorize

Learn more about:

- OAuth2 and foursquare
- The foursquare API

Registering An Application



https://foursquare.com/developers/app/CQMOO1HUSPQTX42NGOKZV32W1EX...

FOURSQUARE

Lyon, FR

Find great places on the go.

Discover what's nearby, search for what you're craving, and get deals and tips along the way.

GET THE APP

Data Mining Lyon

EDIT THIS APP

ADMINISTRATIVE INFORMATION

Owner
Bla bla

[redacted]

WEB ADDRESSES

Download / welcome page url
http://www.bgoncalves.com

Your privacy policy url
http://www.bgoncalves.com/privacy

Redirect URI[s]
http://www.bgoncalves.com/redirect

PUSH API

Push API Notifications
No checkins will be pushed to your app

APP INFO

Short tagline
None provided

Detailed description
None provided

INSTALL OPTIONS

Blackberry App World ID
None provided

iOS App Store ID
None provided

Android Marketplace ID

App Details

Edit App

Icons and Images

Reset or Delete App

0

users have connected this app

Access Token URL
https://foursquare.com/oauth2/access_token

Authorize URL
https://foursquare.com/oauth2/authorize

Learn more about:

- OAuth2 and foursquare
- The foursquare API

Registering An Application

- We now have our client_id and client_secret that we can use to request an access token.
- First we request an auth_url, a URL where the "user" will be asked to login to foursquare and authorize our app

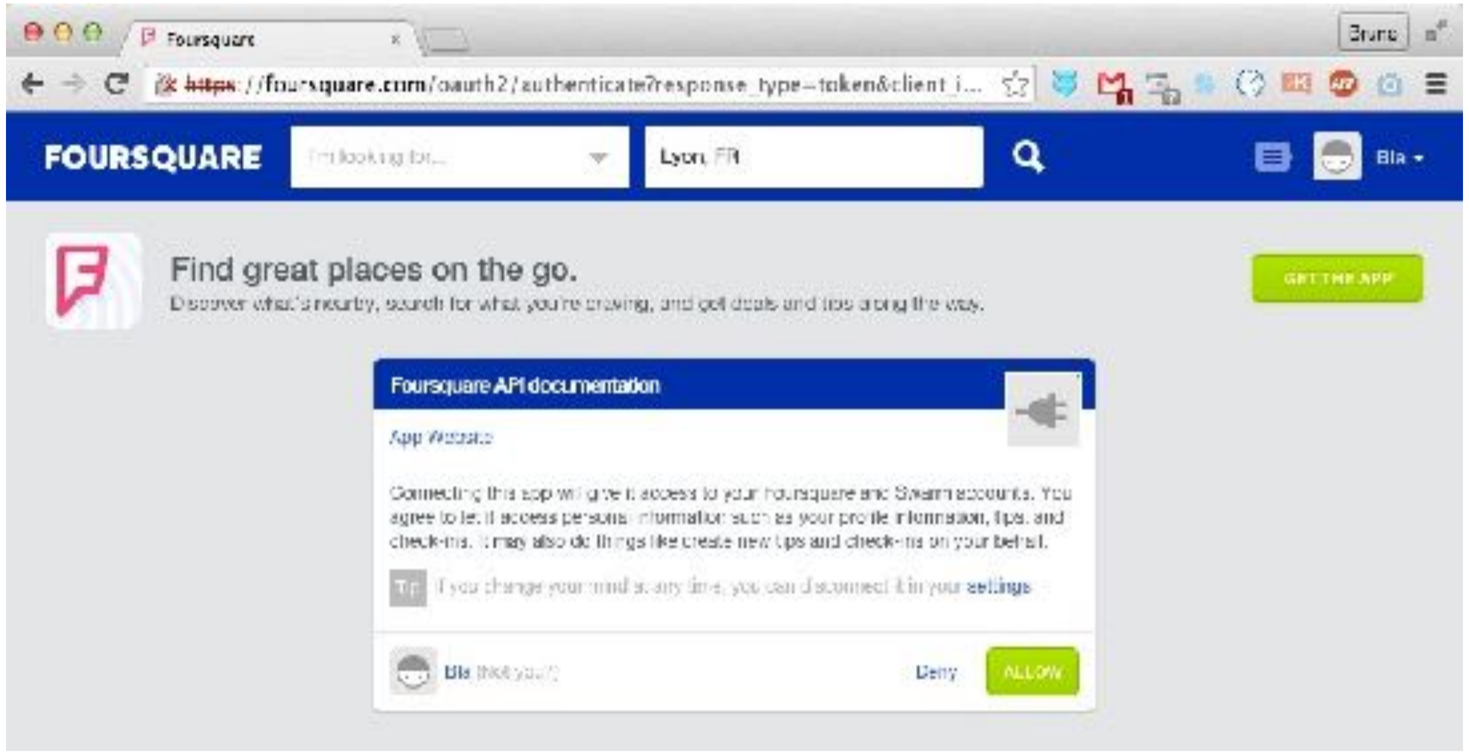
```
import foursquare

accounts = {"tutorial": {"client_id": "CLIENT_ID",
                         "client_secret": "CLIENT_SECRET",
                         "access_token": ""
                         }
            }

app = accounts["tutorial"]

client = foursquare.Foursquare(client_id=app["client_id"],
                               client_secret=app["client_secret"],
                               redirect_uri='http://www.bgoncalves.com/redirect')

auth_uri = client.oauth.auth_url()
print(auth_uri)
```

- In the remainder of this lecture, the accounts dict will live inside the foursquare_accounts.py file

foursquare_auth_uri.py

Registering An Application

- Now we must access our auth_uri in a browser, login and authorize the application


Foursquare
https://foursquare.com/oauth2/authenticate?response_type=token&client_i...
Bruno
FOURSQUARE
I'm looking for...
Lyon, FR
Bla
GET THE APP
Deny
ALLOW

Find great places on the go.

Discover what's nearby, search for what you're craving, and get deals and tips along the way.

Foursquare API documentation

App Website

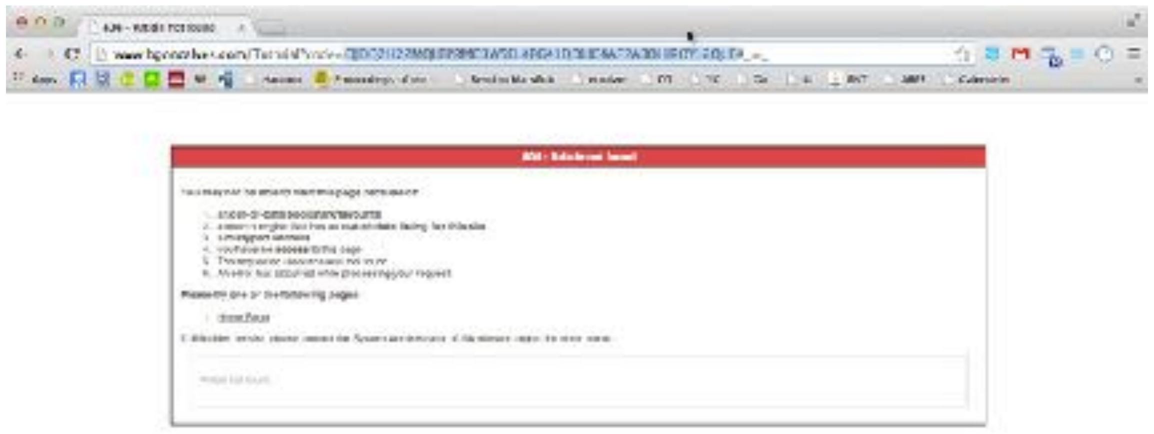
Connecting this app will give it access to your foursquare and Swarm accounts. You agree to let it access personal information such as your profile information, tips, and check-ins. It may also do things like create new tips and check-ins on your behalf.

Tip If you change your mind at any time, you can disconnect it in your settings

Bla (not you?)

Registering An Application

- Afterwards we will be redirected automatically to out "redirect_uri" we used when registering the application


[illegible]

- Don't worry about the error. What we need is just the code portion of the url (the part between the "=" and the "#".

- This is the final piece of the puzzle.

Registering An Application

- with this code we can now request an auth_token which will allow us to authenticate with the Foursquare API

```
access_token = client.oauth.get_token('CODE')
```

- This will return the OAuth2 access_token that we can then use directly.

```
import foursquare
from foursquare_accounts import accounts

app = accounts["tutorial"]

client = foursquare.Foursquare(client_id=app["client_id"],

client_secret=app["client_secret"])

client.set_access_token(app["access_token"])
```

- Much simpler and intuitive
- Less prone to mistakes
- Automatically takes care of all the dirty details

foursquare_authentication.py

Objects

- Two main types of objects:
    - Users
    - Venues
- Multiple possible actions (by Users on Venues)
    - Checkin
    - Like
    - Tip

API Limitations

- Users have privacy concerns with respect to publicly sharing their location.
    - "Stalker apps"
    - "Please Rob Me"
- Privacy is a big concern for Foursquare
- API structure reflects this
- "Easy" to get information on users and on venues. Connecting users to venues much harder to obtain.

Venues

https://developer.foursquare.com/overview/venues.html

- Venues correspond to physical locations
- Are perhaps the most important object in the Foursquare universe
- API is particularly generous, allowing for 5000 requests per hour.
- .venues(venue_id)
  Returns a venue object
- .venues.similar(venue_id)
  Returns a list of similar venues (abbreviated)
- .venues.search({"query":query, "near":location})
  Searches for places matching the query ("pizza", "Eiffel Tower", etc) near location ("Paris", etc).

```
[u'city',
 u'cc',
 u'country',
 u'state',
 u'address',
 u'lat',
 u'lng']
```

```
[u'count',
 u'groups',
 u'summary']
```

```
[u'rating',
 u'reasons',
 u'likes',
 u'mayor',
 u'createdAt',
 u'verified',
 u'id',
 u'shortUrl',
 u'pageUpdates',
 u'location',
 u'tips',
 u'listed',
 u'canonicalUrl',
 u'tags',
 u'photos',
 u'attributes',
 u'stats',
 u'dislike',
 u'hereNow',
 u'categories',
 u'name',
 u'like',
 u'phrases',
 u'specials',
 u'contact',
 u'popular',
 u'timeZone']
```

Similar Venues

https://developer.foursquare.com/overview/venues.html

```
import foursquare
from foursquare_accounts import accounts

app = accounts["tutorial"]

client = foursquare.Foursquare(client_id=app["client_id"],
                               client_secret=app["client_secret"])

client.set_access_token(app["access_token"])

venue_id = "43695300f964a5208c291fe3"
venue = client.venues(venue_id)
similar = client.venues.similar(venue_id)

print("Similar venues to", venue["venue"]["name"], "(", venue["venue"]["hereNow"]["summary"], ")")

for venue in similar["similarVenues"]["items"]:
    print(venue["name"])
```

foursquare_venues.py

Users

https://developer.foursquare.com/docs/users/users

- Users interact with venues in multiple ways:
    - checkin
    - leaving a "tip"
    - "liking"
- Users connect to each other through friendship/colocation
- .users(user_id) Returns the user object
- .users.friends(user_id) Returns the list of friends for a user
- .users.checkins(user_id) Returns list of (public) checkins
- .users.search({"twitter":screen_name}) Search for the user with the given twitter screen_name. Returns abbreviated user object

Tips

https://developer.foursquare.com/docs/venues/tips

- Users can leave tips in venues at any time (without checking in)
- (Reduced) Tips for a venue can be accessed using .venues.tips(venue_id)
- Limited to a maximum of 500 per call, defined with the "count" parameter. Get further tips with "offset" parameter (same as for friends).
- Full Tip objets can be obtained with .tips(tip_id)
- Contain (Reduced) User object and are public, providing an easy way to connect users with venues.

Tips

```
import foursquare
from foursquare_accounts import accounts

app = accounts["tutorial"]

client = foursquare.Foursquare(client_id = app["client_id"],
                               client_secret = app["client_secret"])

client.set_access_token(app["access_token"])

venue_id = "43695300f964a5208c291fe3"

tips = client.venues.tips(venue_id)
tips_list = tips["tips"]["items"]
tip_count = tips["tips"]["count"]

while len(tips_list) < tip_count:
    tips = client.venues.tips(venue_id, {"offset":
len(tips_list)})
    tips_list += tips["tips"]["items"]

print len(tips_list), tip_count

for tip in tips_list:
    print tip["user"]["id"], tip["text"]
```

@bgoncalves

foursquare_tips.py

Checkins

https://developer.foursquare.com/docs/checkins/checkins.html

- Checkins are the *Raison d'être* of Foursquare.
- They connect Users with Venues providing valuable temporal and demographic information.
- .checkins(checkin_id) Returns the Checkin object
- .users.checkins(user_id) Returns the list of ***Public*** checkins for User user_id or all checkins if user_id is friends of the user using the application.

```
import foursquare
from foursquare_accounts import accounts

app = accounts["tutorial"]

client = foursquare.Foursquare(client_id=app["client_id"],

client_secret=app["client_secret"])

client.set_access_token(app["access_token"])

checkin_id = "5089b44319a9974111a6c882"

checkin = client.checkins(checkin_id)
user_name = checkin["checkin"]["user"]["firstName"]

print(checkin_id, "was made by", user_name)
```

foursquare_checkins.py

@bgoncalves

Checkins

https://developer.foursquare.com/docs/checkins/checkins.html

- Users have the option of sharing their checkins through Twitter and Facebook, making them publicly accessible
- The status text is shared along with the URL of the web version of the checkin.
- To allow Twitter and Facebook friends to access the checkin, a special "access_token", called a signature, is added to the checkin URL.
- Each signature is valid for just a single checkin and it allows anyone to access the respective checkin

Checkins

https://developer.foursquare.com/docs/checkins/checkins.html

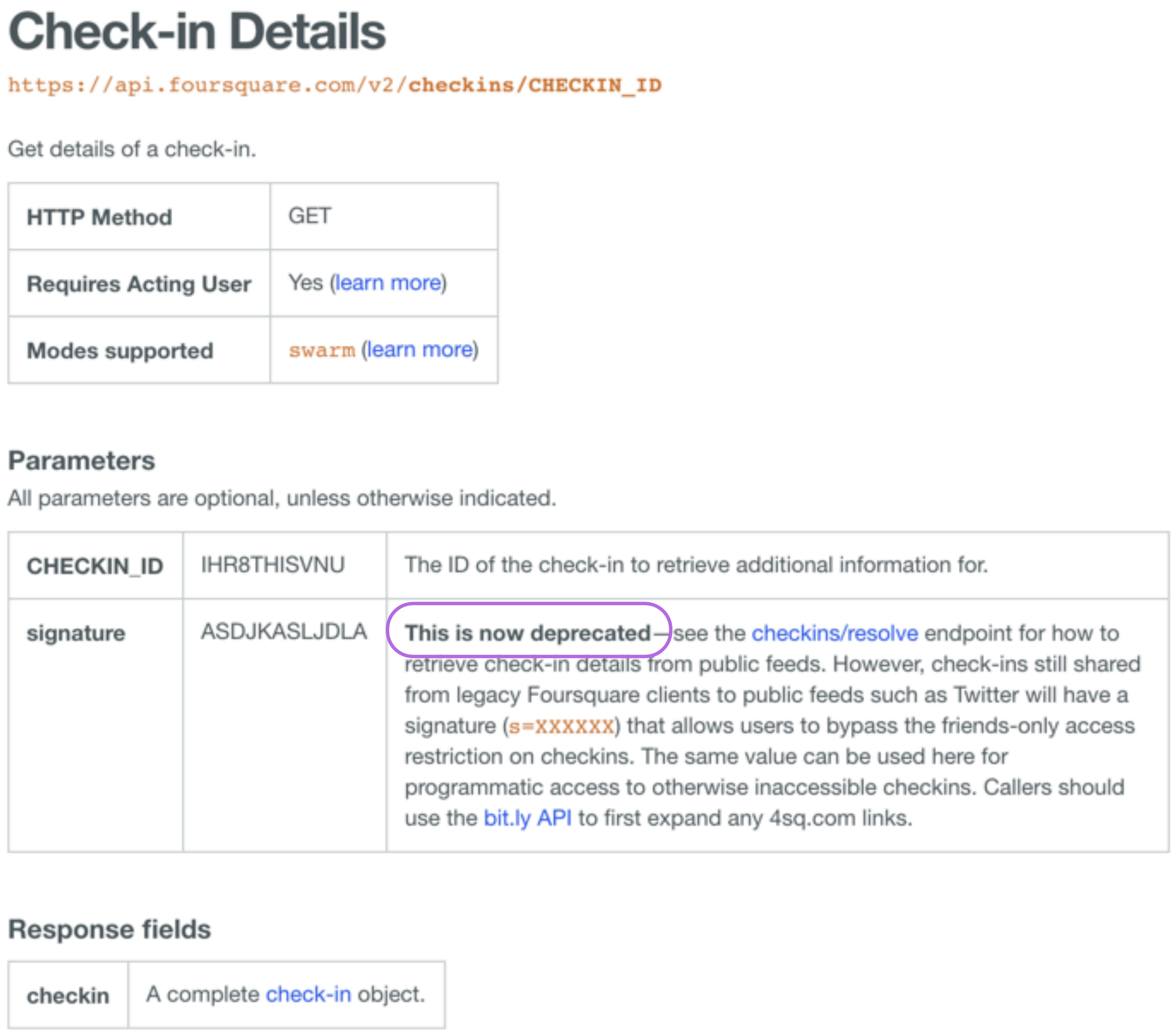
- Signed checkin urls are of the form:

  https://foursquare.com/<user>/checkin/<checkin_id>?s=<signature>&ref=tw

- For example:

  https://foursquare.com/tyayayayaa/checkin/5304b652498e734439d8711f?s=ScMqmpSLg1buhGXQicDJS4A_FVY&ref=tw

- corresponds to user tyayayayaa performing checkin 5304b652498e734439d8711f and has signature ScMqmpSLg1buhGXQicDJS4A_FVY
- .checkins(checkin_id, {"signature": signature}) can be used to query the API using the signature key to access a private checkin



Check-in Details

`https://api.foursquare.com/v2/checkins/CHECKIN_ID`

Get details of a check-in.

HTTP Method	GET
Requires Acting User	Yes (learn more)
Modes supported	`swarm` (learn more)

Parameters

All parameters are optional, unless otherwise indicated.

CHECKIN_ID	IHR8THISVNU	The ID of the check-in to retrieve additional information for.
signature	ASDJKASLJDLA	**This is now deprecated**—see the checkins/resolve endpoint for how to retrieve check-in details from public feeds. However, check-ins still shared from legacy Foursquare clients to public feeds such as Twitter will have a signature (`s=XXXXXX`) that allows users to bypass the friends-only access restriction on checkins. The same value can be used here for programmatic access to otherwise inaccessible checkins. Callers should use the bit.ly API to first expand any 4sq.com links.

Response fields

checkin	A complete check-in object.

Resolving checkins… the hard way!

```
from  foursquare_accounts import accounts
from urllib import parse
import posixpath
import requests

app = accounts["tutorial"]

url_base = "https://api.foursquare.com/v2/checkins/resolve?shortId=%s&oauth_token=%s&v=20160912"

swarm_url = "https://www.swarmapp.com/c/j0cBYuhNHki"
parsed_url = parse.urlparse(swarm_url)
short_id = posixpath.basename(parsed_url.path)

url = url_base % (short_id, app["access_token"])

req = requests.get(url)

checkin = req.json()["response"]["checkin"]
checkin_id = checkin["id"]
user_name = checkin["user"]["firstName"]

print(short_id, ":", checkin_id, "was made by", user_name)
```

@bgoncalves

foursquare_checkins_hard.py