

Trendr App

Implementazione Cloud e Mobile

Nome Cognome

Università degli Studi di Bergamo
n.cognome@studenti.unibg.it

2022-01-26

Indice

1 Panoramica generale

- Repositories del progetto
- Descrizione del progetto
- Architettura del progetto

2 BackEnd

- Dati ottenuti da Twitter API
- Dati salvati su MongoDB Database
- Python script notebook su Google Colab
- AWS Cloud per API e funzioni

3 FrontEnd

- Panoramica della WebApp
- Features

4 Deployment & Demo

- Deployment della WebApp
- Live Demo

5 Crediti

- Ringraziamenti

Panoramica generale

- Repositories del progetto
- Descrizione del progetto
 - Lato BackEnd
 - Lato FrontEnd
- Architettura del progetto
 - Layout
 - Tecnologie usate

Repositories del progetto

GitHub TrendrApp Organization

<https://github.com/trendrapp>

Repo Google Colab Python Script Notebook

https://github.com/trendrapp/Google_Colab_Python_Script_Notebook_Twitter_API_Trends_Data_to_MongoDB

Repo AWS Lambda Functions in NodeJS

https://github.com/trendrapp/AWS_Lambda_Functions_NodeJS

Repo AWS API Gateway

https://github.com/trendrapp/AWS_API_Gateway

Repo Frontend Client - Progressive WebApp in VueJS, Nuxt

https://github.com/trendrapp/Frontend_Client_Progressive_WebApp_VueJS_Nuxt

Repo Trendr App Webpage Deploy

<https://github.com/trendrapp/trendrapp.github.io>

Trendr Website, WebApp

<https://trendrapp.github.io/>

Repos su GitHub

The screenshot shows the GitHub interface for the repository "Trendr". The top navigation bar includes links for Repositories, Packages, People, Teams, Projects, and Settings. A search bar and filters for Type and Language are present. The main content area displays several repository cards:

- Frontend_Client_Progressive_WebApp_VueJS_Nuxt**: Frontend Client Progressive WebApp with VueJS and Nuxt. Languages: JavaScript, CSS, HTML, Vue.js, Babel, Webpack. Last updated 27 minutes ago.
- trendrapp.github.io**: Progressive WebApp website deployment files. Visit <https://trendrapp.github.io/>. Languages: JavaScript, CSS, HTML. Last updated 43 minutes ago.
- AWS_API_Gateway**: AWS API Gateway. Languages: Node.js, AWS Lambda, AWS Step Functions, AWS API Gateway. Last updated 12 hours ago.
- AWS_Lambda_Functions_NodeJS**: AWS Lambda Functions in Node.js to handle API requests and responses. Languages: Node.js, AWS Lambda, AWS Lambda@Edge, AWS Lambda Functions. Last updated 12 hours ago.
- Google_Colab_Python_Script_Notebook_Twitter_API_Trends_Data_to_MongoDB**: Google Colab Python Script Notebook - Requests Twitter API Trends Data and saves them to MongoDB Database. Languages: Python, MongoDB, Twitter API, Google Colab. Last updated 20 hours ago.

On the right side, there are sections for Top languages (JavaScript, Python), Most used topics (AWS, Lambda, JavaScript), and People (with a "Invite someone" button).

Descrizione del progetto (BackEnd)

Il progetto consiste di un lato BackEnd in cui si effettuano:

- Acquisizione dei Trending Topics di Twitter dal suo API mediante uno script (Python).
- Salvataggio dei dati in un Database NoSQL (MongoDB).
- Esposizione di un API Gateway per l'accesso ai dati salvati tramite client.
- Gestione mediante Lambda Functions (NodeJS) delle richieste provenienti dall'API Gateway.
- Invio delle response a tali richieste con dati strutturati.

Descrizione del progetto (FrontEnd)

Inoltre c'è un lato FrontEnd in cui si effettuano:

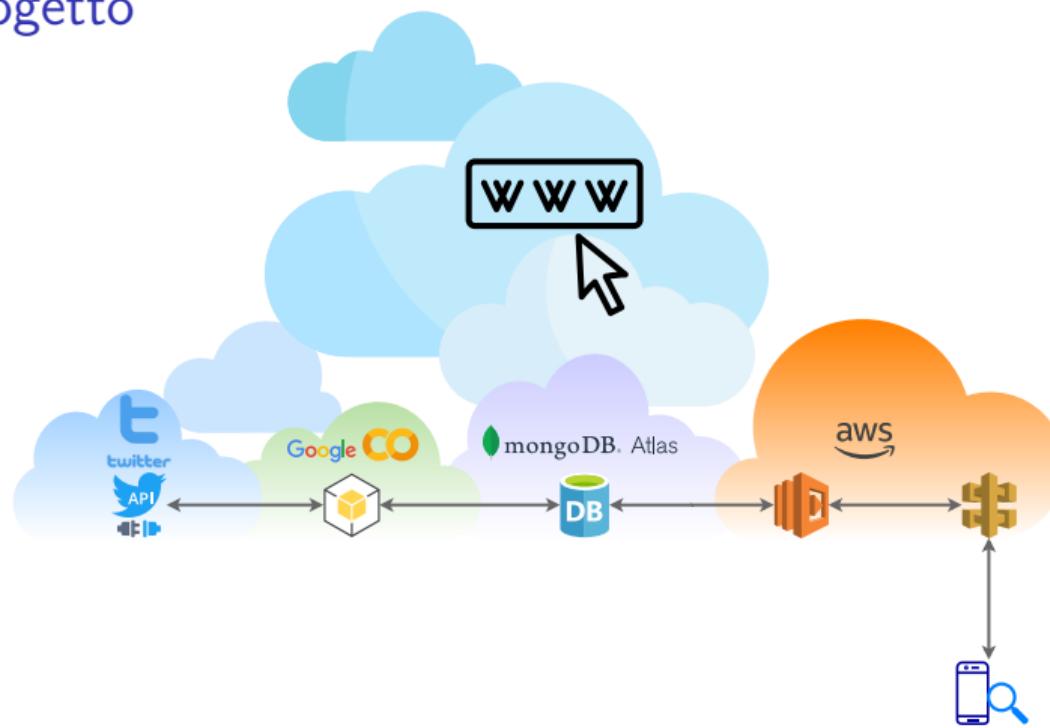
- Interrogazione dell'API Gateway tramite una Progressive WebApp.
- Visualizzazione dei Trending Topics ricevuti come risposta delle interrogazioni.

Il lato client è sviluppato come una Progressive WebApp

Il sito è:

- visualizzabile da qualsiasi client browser desktop
- visualizzabile da browser mobile in modo responsive
- installabile come Progressive WebApp su dispositivi Android e iOS.

Layout del progetto



Tecnologie usate

Lato BackEnd

- Python, Google Colab Notebooks
- MongoDB
- AWS API Gateway
- AWS Lambda Functions
- Javascript, NodeJS, JSON
- Postman
- PyCharm IDE

Lato FrontEnd

- HTML, CSS, Javascript, JSON
- NodeJS, VueJS, Nuxt
- Android OS, iOS
- WebStorm IDE & Chrome DevTools

Deployment

- GitHub Pages

Presentazione

- Latex, Beamer, Overleaf, Draw.io, Inkscape

BackEnd

- Dati ottenuti da Twitter API
 - Twitter Trending Topics
 - GET Locations aventi Trending Topics e i relativi Trends
 - Limiti del request rate di Twitter API
- Dati salvati su MongoDB Database
 - Il database e le collezioni di dati
 - La struttura dei dati
- Python script notebook su Google Colab
 - Google Colab Python Notebook
 - Configurazioni per la connessione a MongoDB e a API di Twitter
 - Procedure dello script
 - Gestione dei limiti del request rate
 - Funzione di inserimento delle available locations e esecuzione delle operazioni
- AWS Cloud per API e funzioni
 - AWS API Gateway - Trends_API
 - AWS Lambda Functions

Twitter API



Twitter Trending Topics

I trending topics sono gli argomenti di cui più si parla in un determinato momento su Twitter, in una determinata località.

Twitter usa degli algoritmi per raggruppare le tendenze e gli hashtag correlati allo stesso argomento.

Trends for you



Trending in Italy

Valentina Ferragni

1,108 Tweets

Auto racing - Trending

#BelgianGP

Trending with: #WTF1, #F1naGlobo

132K Tweets

Formula 1 - Trending

Vettel

12.8K Tweets

Formula 1 - Trending

Ferrari

Trending with: Alpha Tauri, #skysportf1

61.2K Tweets

Politics - Trending

#Portland

58.9K Tweets

Show more

Twitter TT, cosa manca

Twitter offre tramite la sua API la possibilità di accedere ai suoi trending topics del momento.

Le API di Twitter però non offrono la possibilità di vedere i trending topics del passato.

Inoltre è possibile vedere i TT del momento di località diverse dalla propria solo se si è già registrati e loggati su Twitter.

GET Available Locations

Via le Twitter APIs, si può ricavare un JSON contenente un elenco delle locations nelle quali al momento ci sono trending topics.

Ogni location (che sia paese o città) ha un suo WOEID (Where on Earth ID).

GET trends/available

Returns the locations that Twitter has trending topic information for.

The response is an array of "locations" that encode the location's **WOEID** and some other human-readable information such as a canonical name and country the location belongs in.

A **WOEID** is a Yahoo Where On Earth ID.

Resource URL

<https://api.twitter.com/1.1/trends/available.json>

Resource Information

Response formats	JSON
Requires authentication?	Yes
Rate limited?	Yes
Requests / 15-min window (user auth)	75
Requests / 15-min window (app auth)	75

Parameters

None

Example Request

GET <https://api.twitter.com/1.1/trends/available.json>

Example Response

```
[  
  {  
    "country": "Sweden",  
    "countryCode": "SE",  
    "name": "Sweden",  
    "parentid": 1,  
    "placeType": {  
      "code": 12,  
      "name": "Country"  
    },  
    "url": "http://where.yahooapis.com/v1/place/23424954",  
    "woeid": 23424954  
},  
]
```

GET Trends

Una volta ottenuto l'elenco delle locations nelle quali al momento ci sono trending topics, interroghiamo l'API di Twitter per ricavare i trends in ciascuna delle locations.

L'unico parametro richiesto è l'WOEID della location.

GET trends/place

Returns the top 50 trending topics for a specific `woeid`, if trending information is available for it.

The response is an array of `trend` objects that encode the name of the trending topic, the query parameter that can be used to search for the topic on Twitter Search, and the Twitter Search URL.

This information is cached for 5 minutes. Requesting more frequently than that will not return any more data, and will count against rate limit usage.

The `tweet_volume` for the last 24 hours is also returned for many trends if this is available.

Resource URL

<https://api.twitter.com/1.1/trends/place.json>

Resource Information

Response formats	JSON
Requires authentication?	Yes
Rate limited?	Yes
Requests / 15-min window (user auth)	75
Requests / 15-min window (app auth)	75

Parameters

Name	Required	Description
id	required	The Yahoo! Where On Earth ID of the location to return trending information for. Global information is available by using 1 as the WOEID.
exclude	optional	Setting this equal to hashtags will remove all hashtags from the trends list.

Example Request

GET <https://api.twitter.com/1.1/trends/place.json?id=1>

Example Response

```
[  
]  
  "trends": [  
    {  
      "name": "#ChainedToTheRhythms",  
      "url": "http://twitter.com/search?q=%23ChainedToTheRhythms",  
      "promoted_content": null,  
      "query": "#ChainedToTheRhythms",  
      "tweet_volume": 48857  
    },  
  ]
```

Twitter API Rate limits

L'accesso a questi dati è limitato ad un numero massimo di requests per utente o app ogni 15 minuti.

Nello sviluppo dello script Python in Google Colab notebook per l'interrogazione di Twitter APIs e il salvataggio dei dati in MongoDB, si rispetta tale limite mantenendo una moderata rate di invio di requests per permettere il reset delle richieste allotted.

Rate limits

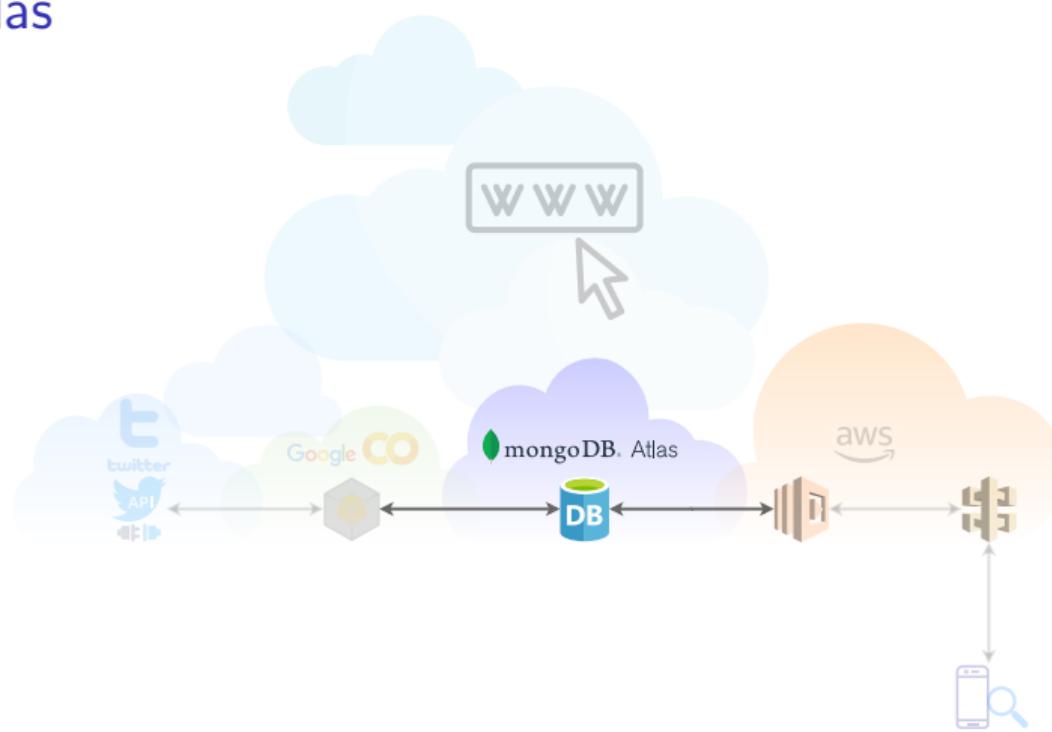
Standard API v1.1 rate limits per window

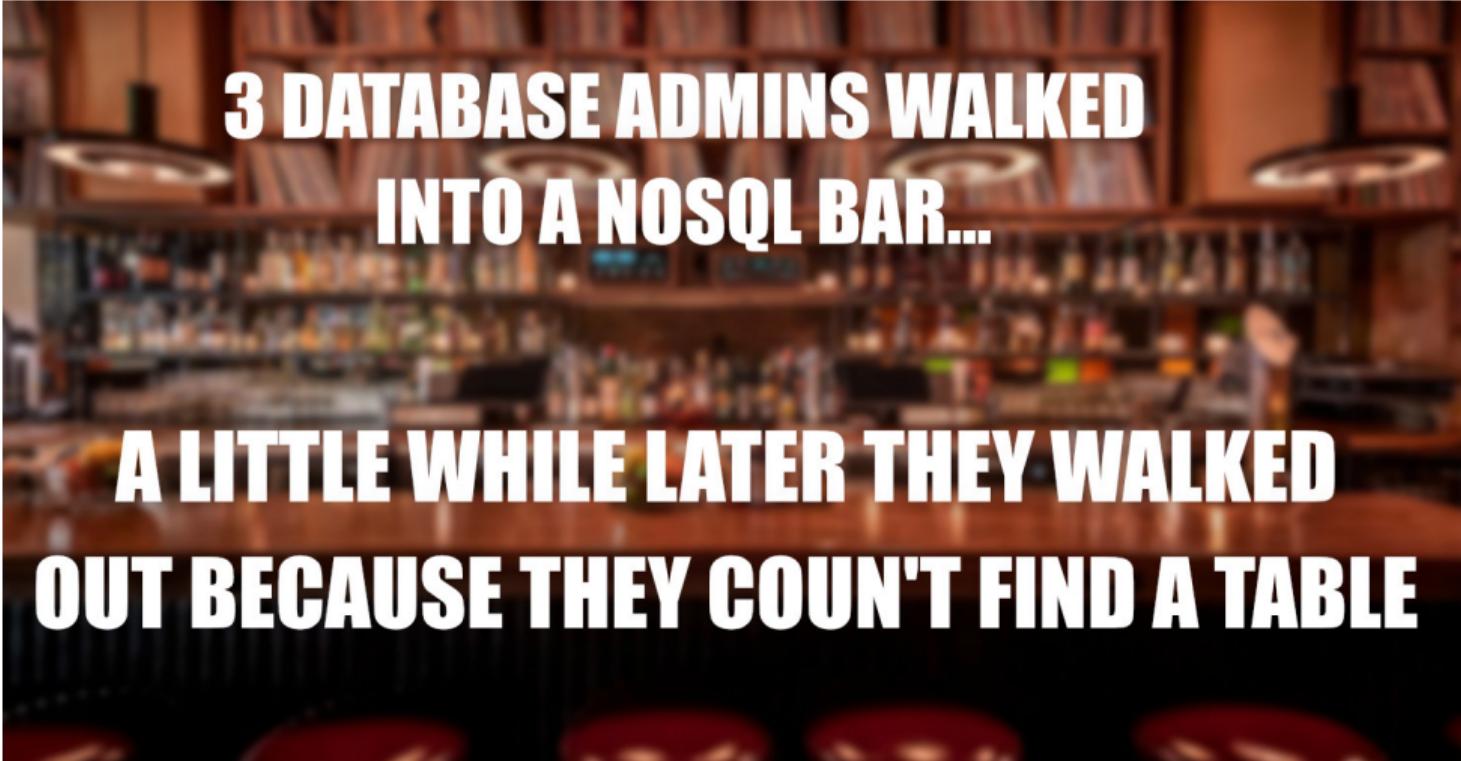
GET endpoints

The standard API rate limits described in this table refer to GET (read) endpoints. Note that endpoints not listed in the chart default to 15 requests per allotted user. All request windows are 15 minutes in length. These rate limits apply to the standard API endpoints only, does not apply to premium APIs.

Endpoint	Requests / window per user	Requests / window per app
GET trends/available	75	75
GET trends/place	75	75

MongoDB Atlas





**3 DATABASE ADMINS WALKED
INTO A NOSQL BAR...**

**A LITTLE WHILE LATER THEY WALKED
OUT BECAUSE THEY COULDN'T FIND A TABLE**

Il cluster

Per cluster in MongoDB si intende sharded cluster. Il motivo è lo scaling delle letture e scritture lungo diversi nodi. Ogni nodo non gestisce tutti i dati.

The screenshot shows the MongoDB Atlas Cluster Overview page. At the top, there's a navigation bar with 'Project 0', 'Atlas', 'Realm', and 'Charts' tabs. Below the navigation, the cluster details are displayed:

- ORG - 2020-08-05**
- Clusters**: Shows a single cluster named 'IlMioCluster' with 'Version 4.2.8'. A search bar above the cluster list says 'Find a cluster...'. Buttons for 'CONNECT', 'METRICS', 'COLLECTIONS', and a gear icon are available.
- DATA STORAGE**: Includes links for 'Clusters', 'Triggers', 'Data Lake', and 'Advanced'.
- SECURITY**: Includes links for 'Database Access' and 'Network Access'.
- CLUSTER TIER**: Shows 'M0 Sandbox (General)'.
- REGION**: Shows 'AWS / Ireland (eu-west-1)'.
- TYPE**: Shows 'Replica Set - 3 nodes'.
- LINKED REALM APP**: Shows 'None Linked'.

Il database e le collezioni

Per il salvataggio (e successiva lettura più agevole) dei dati delle trending topics e delle locations si fa uso di un database con due collections, una per i trends e l'altra per le località.

The screenshot shows the MongoDB Atlas interface for the IIMioCluster database. The top navigation bar includes links for Atlas, Realm, Charts, and a user icon. On the right, it displays the VERSION (4.2.8) and REGION (Region One). The main dashboard shows 1 database and 2 collections. The Trending_Topics collection is selected, displaying its size (101.09MB), total documents (297515), and index size (8.59MB). It also shows tabs for Find, Indexes, Schema Anti-Patterns (0), Aggregation, and Search Indexes. A search bar at the bottom contains the filter query: {"filter": "example"}. Below the collection details, a message says "QUERY RESULTS 1-20 OF MANY".

Formato dei dati in Available_Locations

La struttura dei dati inseriti nella collection delle località è composta da una data-ora, un oggetto "locations" con dentro "Worldwide", il suo WOEID e un array di paesi ciascuno con nome, WOEID e una lista delle proprie città con relativo WOEID.

Formato dei dati in Available_Locations

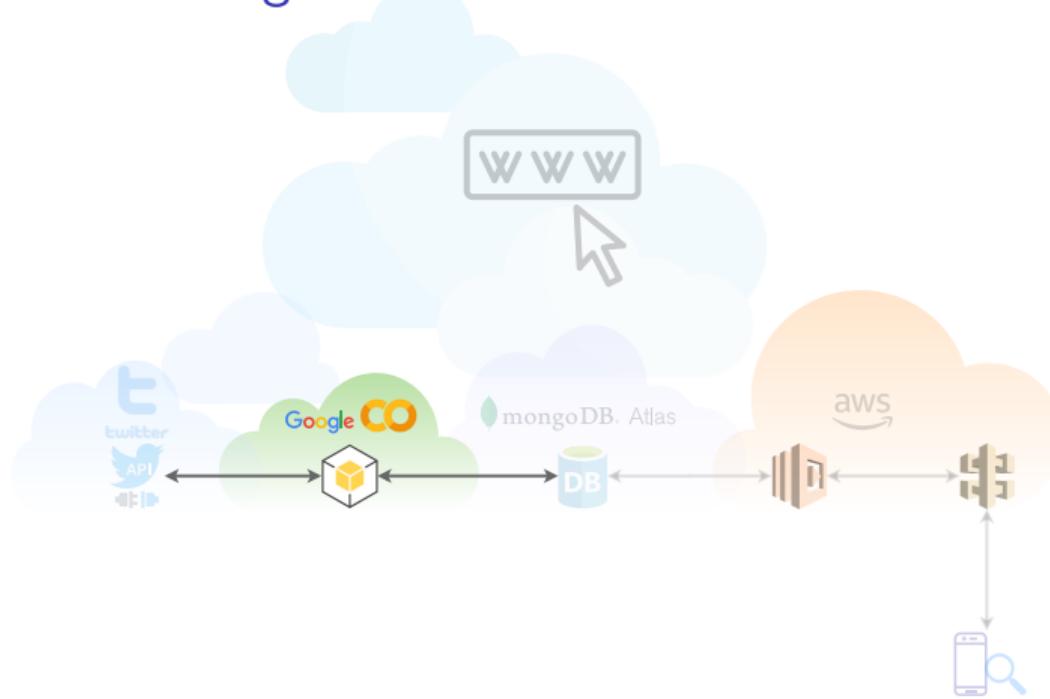
```
_id: ObjectId("5f36bffffd1369ce3e31683e")
dateTime: "2020-08-14-18:45:33"
locations: Object
  locations_name: "Worldwide"
  locations_woeid: 1
countries: Array
  0: Object
    locations_name: "Canada"
    locations_woeid: 23424775
    towns: Array
      0: Object
        locations_name: "Winnipeg"
        locations_woeid: 2972
      1: Object
        locations_name: "Ottawa"
        locations_woeid: 3369
      2: Object
        locations_name: "Quebec"
        locations_woeid: 3444
      3: Object
      4: Object
      5: Object
      6: Object
      7: Object
    1: Object
      locations_name: "United Kingdom"
      locations_woeid: 23424975
    towns: Array
      0: Object
        locations_name: "Birmingham"
        locations_woeid: 12723
      1: Object
      2: Object
      3: Object
      4: Object
      5: Object
```

Formato dei dati in Trending_Topics

La struttura dei dati inseriti nella collection dei trends è composta da una data-ora di inserimento e creazione, nome della location, il tipo, WOEID, WOEID del genitore (genitore di città è paese, genitore di paese è worldwide). Inoltre ci sono i dati del topic come il nome o l'hashtag, l'url, se è un ad, la string query e il volume dei tweet che parlano di quell'argomento."

```
_id: ObjectId("5f39a421b01ed305dec7e89e")
as_of: "2020-08-16T21:24:49Z"
created_at: "2020-08-16T21:17:42Z"
locations_name: "Milan"
location_type: "Town"
locations_woeid: 718345
parent_id: 23424853
parent_name: "Italy"
name: "#discoteche"
url: "http://twitter.com/search?q=%23discoteche"
promoted_content: null
query: "%23discoteche"
tweet_volume: 18171
```

Python notebook su Google Colab



Installazione e importazione delle librerie

dnspython è un DNS toolkit per Python.

▼ Installazione delle librerie necessarie

```
1 !pip install --user dnspython  
2 !pip install --user twitter
```

pymongo si usa per accedere a MongoDB.

▼ Importazione delle librerie

```
1 import pymongo  
2 import dns # necessario per una connessione MongoDB con SRV  
3 import twitter  
4 import time  
5 from datetime import datetime
```

twitter si usa per accedere alle APIs di Twitter.

Configurazione chiavi per la connessione al MongoDB

▼ Configurazione del database MongoDB, delle collezioni e utenti

```
1 dbURI = "mongodb+srv://ilmioccluster.wzsfjl.mongodb.net"
2 dbName = "Database_del_Progetto"
3 collectionNameTrendingTopics = "Trending_Topics"
4 collectionNameAvailableLocations = "Available_locations"
5 dbUsername = ""
6 dbPassword = ""
7 # localJsonDataFile = "data.json"
```

Configurazione chiavi per la connessione al Twitter API

- ▼ Configurazione delle access keys all'API di Twitter

```
1 API_key = ''  
2 API_secret = ''  
3 accessToken = ''  
4 accessTokenSecret = ''
```

- ▼ Configurazione del tempo in secondi per il reset della rate limit di requests all'API di twitter

```
1 seconds_to_reset = 15 * 60 # 15 minuti
```

Funzioni di connessione, pulizia di una collection, rinomina e stampa

▼ Definizione delle funzioni di connessione, pulizia di una collection, rinomina e stampa

```
1 # Connect to a MongoDB database
2 def connectToDB(uri, uname, pword):
3     return pymongo.MongoClient(uri, username=uname, password=pword)
```

```
1 # Pulizia delle collection
2 def cleanCollection(collection):
3     collection.delete_many({})
```

```
1 # Rinomina collection
2 def renameCollection(oldCollection, newCorrectionName):
3     oldCollection.rename(newCorrectionName)
```

```
1 # Stampa contenuto di una collection
2 def printCollectionData(collection):
3     for elemento in collection.find():
4         print(elemento)
```

Funzione di controllo di duplicati, voci già presenti

Controlla se un certo trend di una data in una location è già presente in database (per evitare duplicati quando lo script Python si esegue più volte al giorno).

Funzione di controllo che dato un trending topic di una certa location in una certa data, controlla che non sia già presente in una collection

```
1 def checkIfPresent(collection, trendingTopicRecord):
2     data = trendingTopicRecord['created_at'][0:10]
3     woeid = trendingTopicRecord['locations_woeid']
4     trendName = trendingTopicRecord['name']
5     presentRecord = collection.find({"$and": [{"created_at": {"$regex": str(data) + '.*'}},
6                                         {'locations_woeid': woeid},
7                                         {'name': trendName}]}))
8     for records in presentRecord:
9         return True
10    return False
```

Funzione di inserimento dei trends e gestione del rate limit

Definizione di una funzione che data una location, l'API di twitter, una collection delle TT e i tempi da una request all'altra all'API di Twitter e di reset delle rate limit, inserisce le trending topics in tale collezione

```

1  def getAndAddTrendingTopics(location,
2                               api_di_twitter,
3                               collezione_delle_trending_topics,
4                               seconds_between_each_app_request_arg,
5                               seconds_to_reset_arg):
6      rate_status_trends = api_di_twitter.application.rate_limit_status()['resources']['trends'][ '/trends/place' ]
7      time.sleep(seconds_between_each_app_request_arg) # wait for trends request time
8      # print("seconds_between_each_app_request ", seconds_between_each_app_request_arg)
9      limit_trends = rate_status_trends['limit']
10     remaining_limit_trends = rate_status_trends['remaining']
11     seconds_between_each_trends_request = seconds_to_reset_arg / limit_trends
12     # print("remaining_limit_trends ", remaining_limit_trends)
13     # print("seconds_between_each_trends_request", seconds_between_each_trends_request)
14
15     while remaining_limit_trends <= 1:
16         remaining_limit_trends = \
17             api_di_twitter.application.rate_limit_status()['resources']['trends'][ '/trends/place' ][ 'remaining' ]
18         time_to_wait = 0 if remaining_limit_trends > 1 else max(seconds_between_each_trends_request,
19                                                               seconds_between_each_app_request_arg)
20         time.sleep(time_to_wait) # wait for trends or app request time
21         # print("seconds_between_each_app_trend_req ", time_to_wait)
22         # print("remaining_limit_trends ", remaining_limit_trends)
23
24     trendsDataJsonText = api_di_twitter.trends.place(_id=location['woeid'])
25     time.sleep(max(seconds_between_each_trends_request - seconds_between_each_app_request_arg,
26                    0)) # wait for trends request time - app request time
27     trends = trendsDataJsonText[0]['trends']
28     dateTimenow = datetime.today().strftime('%Y-%m-%d %H:%M:%S')
29     metadata = {
30         "as_of": trendsDataJsonText[0]['as_of'],
31         "created_at": str(dateTimenow),
32         "locations_name": location['name'],
33         "location_type": location['placeType'][ 'name' ],
34         "locations_woeid": location['woeid'],
35         "parent_id": location['parentid'],
36         "parent_name": "Worldwide" if location['placeType'][ 'name' ] == "Country" else location[ 'country' ]
37     }
38 }
39 for trend in trends:
40     record = {"metadata": **trend}
41     if not checkIfPresent(collezione_delle_trending_topics, record): # controlla che tale trending topic non sia già in db
42         collezione_delle_trending_topics.insert_one(record)

```

Funzione di inserimento delle available locations (1)

Continua nella slide dopo...

Definizione di una funzione che dato l'API di Twitter, la collection delle trending topics e delle available locations, dei tempi di secondi tra ogni request e di reset delle rate limit, prende dall'API di twitter i dati e inserisce senza duplicati le available locations con trends e i rispettivi trending topics nelle due collections.

```
1  def getAndAddAvailableLocationsAndTrendingTopics(api_di_twitter,
2                                                 collezione_trending_topics,
3                                                 collezione_available_locations,
4                                                 secondsBetweenEachAppRequestArg,
5                                                 secondsToReset):
6
7     dateNow = datetime.today().strftime('%Y-%m-%d')
8     dateTimNow = datetime.today().strftime('%Y-%m-%d-%H:%M:%S')
9     woiedsWhereTrendsAvailable = api_di_twitter.trends.available()
10
11    for availableLocation in woiedsWhereTrendsAvailable:
12
13        print("Adding", availableLocation['name'], "to the available locations collection.")
14
15        # Inserimento locationi in available locations collection
16        locationRecord = {"locations_name": "Worldwide", "locations_woeid": 1, "countries": []}
17        locationInDBquery = collezione_available_locations.find({'dateTime': {"$regex": str(dateNow) + '.*'}})
18
19        locationInDB = ""
20        for item in locationInDBquery:
21            locationInDB = item
22
23        if len(locationInDB) == 0: # empty locationInDB
24            if availableLocation['placeType'][0]['name'] != "Supername":
25                if availableLocation['placeType'][0]['name'] == "Country":
26                    locationRecord['countries'] = [
27                        {"locations_name": availableLocation['name'], "locations_woeid": availableLocation['woeid'],
28                         "towns": []}]
29                else: # it's a town
30                    locationRecord['countries'] = \
31                        [
32                            {
33                                "locations_name": availableLocation['country'],
34                                "locations_woeid": availableLocation['parentid'],
35                            }
36                        ]
37                        towns = [{"locations_name": availableLocation['name'],
38                                  "locations_woeid": availableLocation['woeid']}]
39                        locationRecord = (**locationRecord['countries'][0], **towns)
40
41                jsonRecord = {"dateTime": dateTimNow, "locations": locationRecord}
42                collezione_available_locations.insert_one(jsonRecord)
```

Funzione di inserimento delle available locations (2)

```

41         else: # locationInDB is not empty, contains locations
42             if availableLocation['placeType']['name'] != "Supername":
43                 if availableLocation['placeType']['name'] == "Country":
44                     # check if country is present
45                     alreadyPresent = False
46                     for country in locationInDB['locations']['countries']:
47                         if country['locations_woeid'] == availableLocation['woeid']:
48                             alreadyPresent = True
49                         # if country is not present add it
50                         if not alreadyPresent:
51                             newCountry = {"locations_name": availableLocation['name'],
52                                         "locations_woeid": availableLocation['woeid'], "towns": []}
53                             locationInDB['locations']['countries'].append(newCountry)
54                             jsonRecord = {"dateTime": dateNow, "locations": locationInDB['locations']}
55                             collezione_available_locations.delete_one({'dateTime': {"$regex": str(dateNow) + '.'}})
56                             collezione_available_locations.insert_one(jsonRecord)
57             else: # if it's a town
58                 # check if the town is present
59                 alreadyPresentCountry = False
60                 for country in locationInDB['locations']['countries']:
61                     # check first if the country is present
62                     if country['locations_woeid'] == availableLocation['parentid']:
63                         alreadyPresentCountry = True
64                         alreadyPresentTown = False
65                         # Country is present, check if town is present
66                         for town in country['towns']:
67                             if town['locations_woeid'] == availableLocation['woeid']:
68                                 alreadyPresentTown = True
69                         # if country present but town not present, add town
70                         if not alreadyPresentTown:
71                             newTown = {"locations_name": availableLocation['name'],
72                                         "locations_woeid": availableLocation['woeid']}
73                             country['towns'].append(newTown)
74                             jsonRecord = {"dateTime": dateNow, "locations": locationInDB['locations']}
75                             collezione_available_locations.delete_one({'dateTime': {"$regex": str(dateNow) + '.'}})
76                             collezione_available_locations.insert_one(jsonRecord)
77                     # if country not present, add country and town
78                     if not alreadyPresentCountry:
79                         newCountry = {"locations_name": availableLocation['country'],
80                                       "locations_woeid": availableLocation['parentid'], "towns": []}
81
82                         newTown = {"locations_name": availableLocation['name'],
83                                     "locations_woeid": availableLocation['woeid']}
84                         newCountry['towns'].append(newTown)
85                         locationInDB['locations']['countries'].append(newCountry)
86                         jsonRecord = {"dateTime": dateNow, "locations": locationInDB['locations']}
87                         collezione_available_locations.delete_one({'dateTime': {"$regex": str(dateNow) + '.'}})
88                         collezione_available_locations.insert_one(jsonRecord)
89
90             getAndAddTrendingTopics(availableLocation,
91                                     twitterAPI,
92                                     collezione_trending_topics,
93                                     secondsBetweenEachAppRequestArg,
94                                     secondsToReset) # inserisci i trends

```

Esecuzione delle operazioni

Il 'main' dove si chiamano le funzioni definite precedentemente

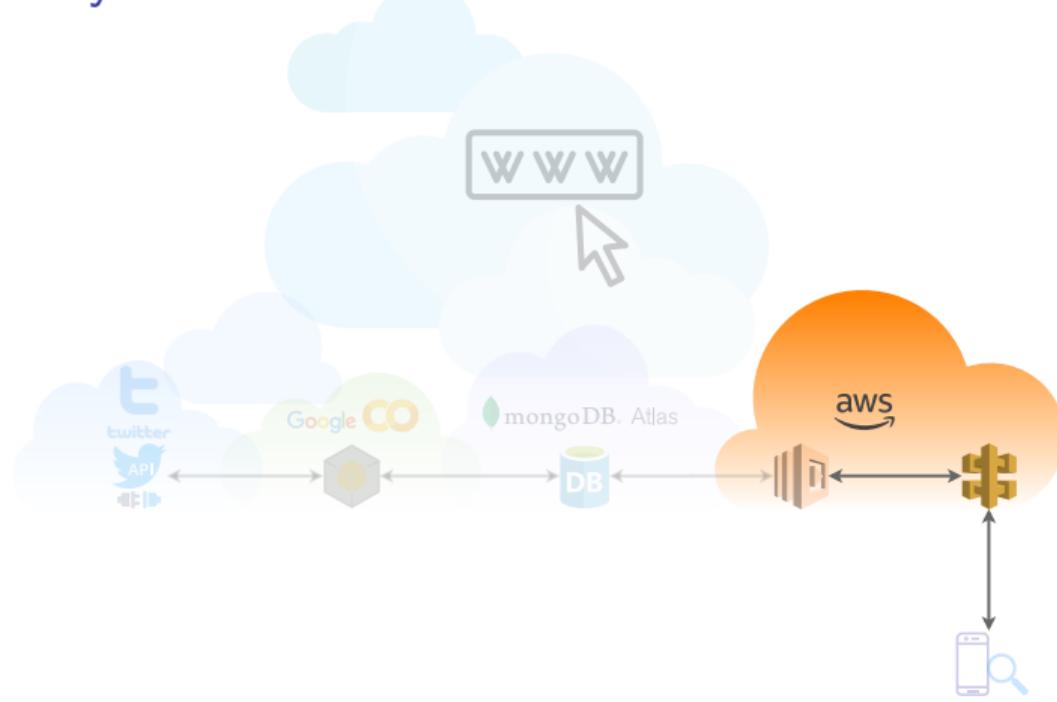
Creazione delle connessioni all'API di twitter e a MongoDB ed esecuzione delle operazioni di inserimento nelle collections delle locations con available trending topics e i rispettivi TT.

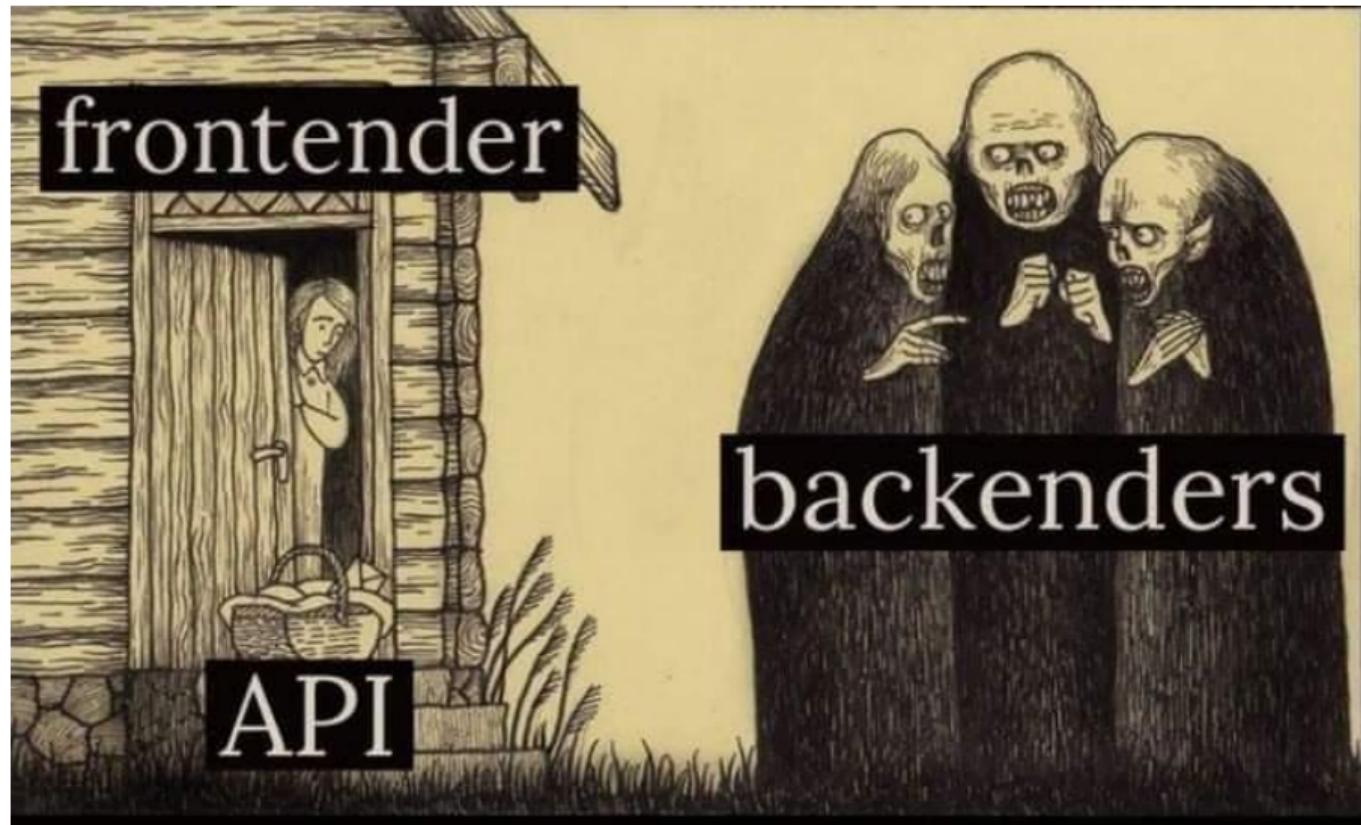
```
1 authentication = twitter.oauth.OAuth(accessToken, accessTokenSecret, API_key, API_secret)
2 twitterAPI = twitter.Twitter(auth=authentication)
3
4 client = connectToDB(dbURL, dbUsername, dbPassword)
5 db = client[dbName]
6 collezioneTrendingTopics = db[collectionNameTrendingTopics]
7 collezioneAvailableLocations = db[collectionNameAvailableLocations]
8
9 # cleanCollection(collezioneTrendingTopics)
10 # printCollectionData(collezioneTrendingTopics)
11
12 rate_status_app = \
13     twitterAPI.application.rate_limit_status()['resources']['application'][ '/application/rate_limit_status']
14 limit_app = rate_status_app['limit']
15 remaining_limit_app = rate_status_app['remaining']
16 # print("remaining_limit_app", remaining_limit_app)
17 seconds_between_each_app_request = seconds_to_reset / limit_app
18
19 getAndAddAvailableLocationsAndTrendingTopics(twitterAPI,
20                                               collezioneTrendingTopics,
21                                               collezioneAvailableLocations,
22                                               seconds_between_each_app_request,
23                                               seconds_to_reset)
```

AWS Cloud Platform



AWS API Gateway - Trends_API





AWS API Gateway - Trends_API/locations

Per richiedere le locations con trends in una certa data, l'unico parametro è la data.

Le richieste e le risposte vengono gestite dalla AWS Lambda Function 'getAvailableLocationsByDate'.

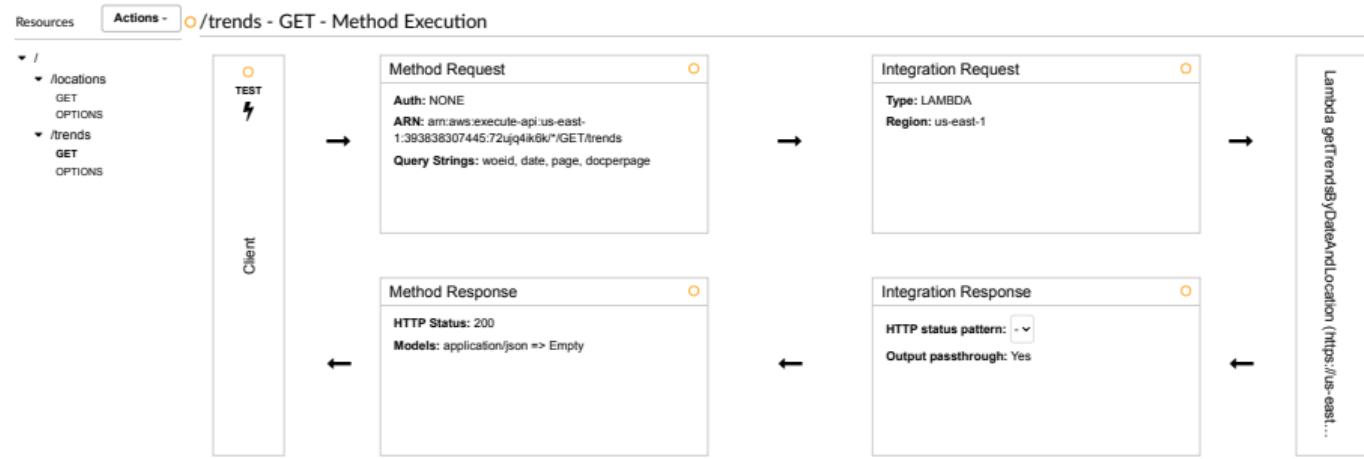


AWS API Gateway - Trends_API/trends

Per richiedere le trending topics in una certa data in una certa location, i parametri richiesti sono la data e l'woeid della location.

Inoltre per l'impaginazione lato FrontEnd serve sapere il numero della pagina e il numero di trends da inviare in risposta.

Le richieste e le risposte vengono gestite dalla AWS Lambda Function 'getTrendsByDateAndLocation'.



AWS Lambda Functions





imgflip.com

AWS Lambda Functions - getAvailableLocationsByDate

Interroga il MongoDB database per avere le locations figlie di un elemento con data quella fornita come parametro della query all'API.

Risponde alla richiesta dell'API.

```
1 const connect_to_db = require('./db');
2
3 const location = require('./locations');
4
5 module.exports.handler = (event, context, callback) => {
6   context.callbackWaitsForEmptyEventLoop = false;
7   console.log(`Received event:\n${JSON.stringify(event, null, 2)}`);
8
9   // set default
10  if(!event.date) {
11    callback(null, {
12      statusCode: 500, headers: { 'Content-Type': 'text/plain' },
13      body: 'Could not fetch the locations. Date is null.'
14    })
15  }
16
17  connect_to_db().then(() => {
18    console.log(`=> get all available locations that have trends in that date`);
19    location.find({dateTime: new RegExp(event.date)}, {_id: 0})
20      .then(
21        locations => {
22          callback(null, {
23            statusCode: 200,
24            body: {locations}
25          })
26        }
27      )
28      .catch(err =>
29        callback(null, {
30          statusCode: err.statusCode || 500,
31          headers: { 'Content-Type': 'text/plain' },
32          body: 'Could not fetch the locations.'
33        })
34      );
35  });
36};
```

AWS Lambda Functions - getTrendsByDateAndLocation

Interroga il MongoDB database per avere i nomi dei trends presenti con data e location fornita come parametro della query all'API.

```
1 const connect_to_db = require('./db');
2
3 const trend = require('./trends');
4
5 module.exports.handler = (event, context, callback) => {
6   context.callbackWaitsForEmptyEventLoop = false;
7   console.log('Received event:', JSON.stringify(event, null, 2));
8
9   // set default
10  if(!event.date || !event.woeid) {
11    callback(null, {
12      statusCode: 500,
13      headers: { 'Content-Type': 'text/plain' },
14      body: 'Could not fetch the trends. Date or location woeid is null.'
15    })
16  }
17
18  if (!event.docperpage) { event.docperpage = 10 }
19  if (!event.page) { event.page = 1 }
20
21  connect_to_db().then(() => {
22    console.log('>> get all trends');
23    trend.find({created_at: new RegExp(event.date), locations_woeid: event
24      .woeid})
25      .skip((event.docperpage * event.page) - event.docperpage)
26      .limit(event.docperpage)
27      .then(trends => {
28        trends = trends.map(trend => ({
29          name: trend.name,
30          url: trend.url,
31          tweetvolume: trend.tweet_volume
32        }));
33
34        callback(null, {
35          statusCode: 200,
36          body: {
37            date: event.date,
38            woeid: event.woeid,
39            docperpage: event.docperpage,
40            page: event.page,
41            trends: trends
42          }
43        })
44      )
45      .catch(err =>
46        callback(null, {
47          statusCode: err.statusCode || 500,
48          headers: { 'Content-Type': 'text/plain' },
49          body: 'Could not fetch the trends.'
50        })
51  });
52};
```

Risponde alla richiesta dell'API.

FrontEnd

- Panoramica della WebApp
 - Tecnologie VueJS e Nuxt per sito web e progressive mobile app
 - Views e schermate durante l'esecuzione
- Features
 - Installazione su mobile da browser
 - Date disponibili con trends del passato
 - Locations con trends, paesi e città
 - Ricerca trends del passato
 - Lazy loading, infinite scroll

Progressive WebApp

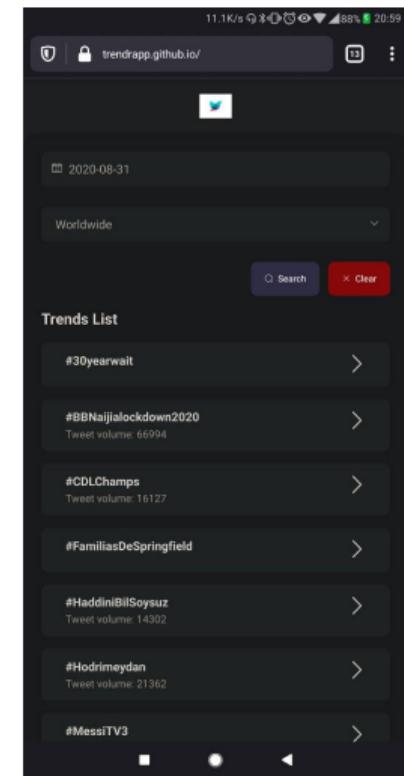
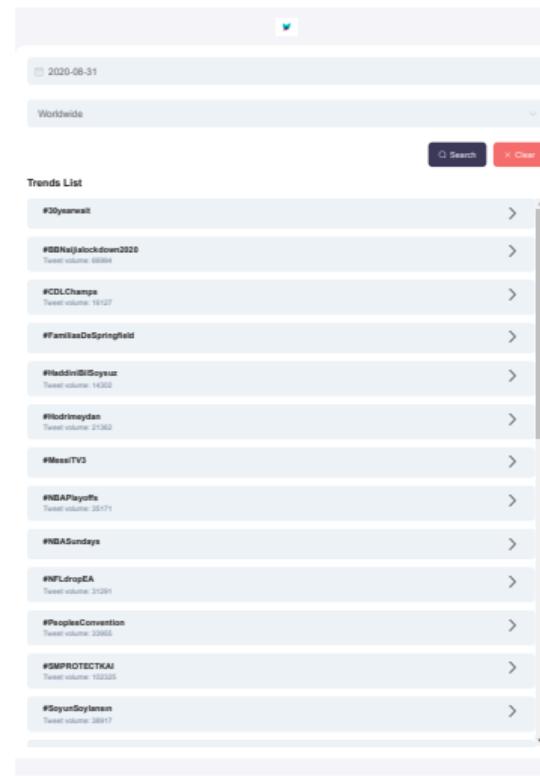


Schermata iniziale della WebApp

Data di default settata a quella di oggi.

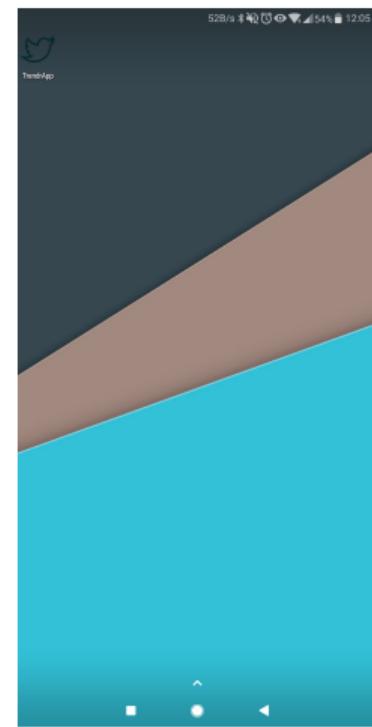
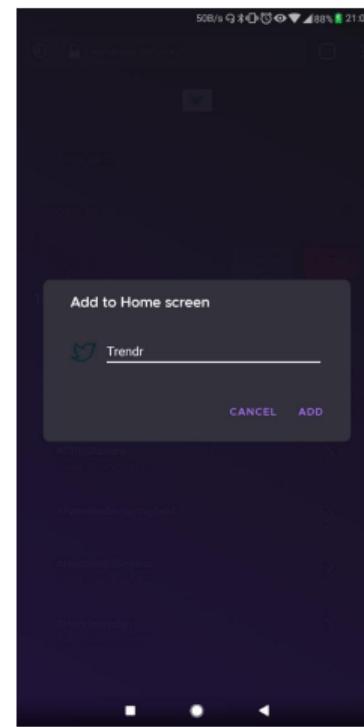
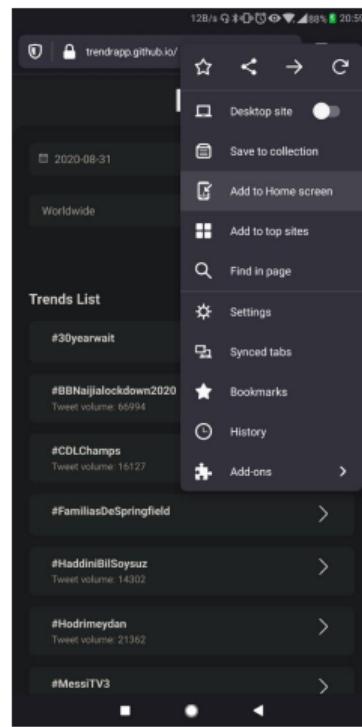
Location di default settata a 'Worldwide', WOEID = 1.

In automatico la WebApp mostra le trending topics del giorno nel mondo.



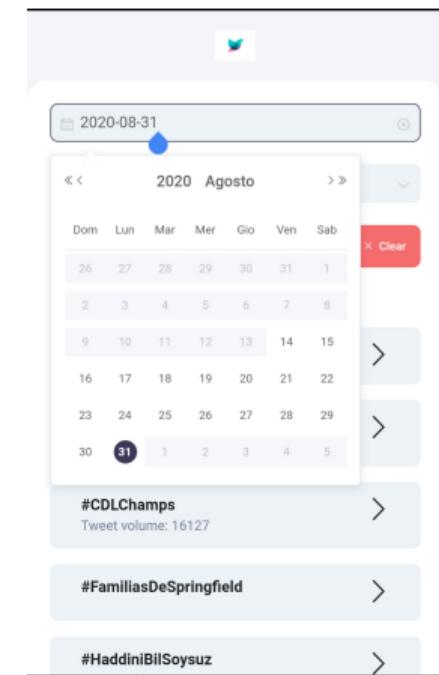
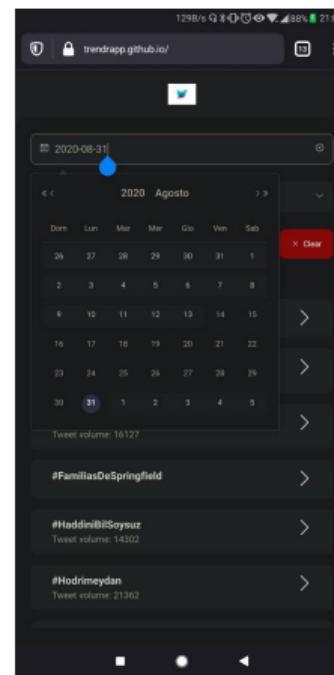
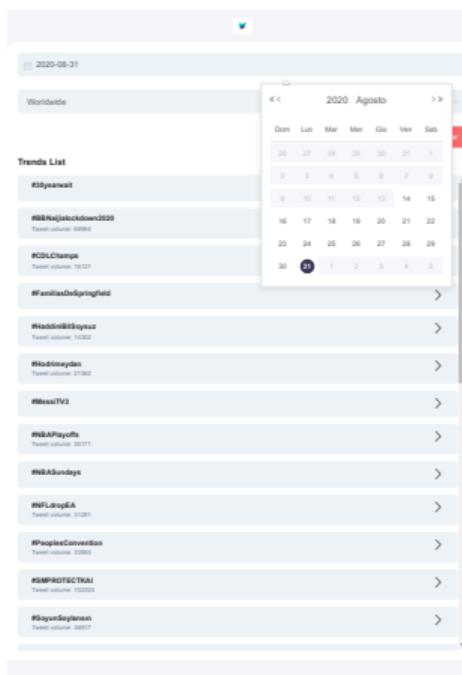
Installazione su mobile da browser

Installazione su Mozilla Firefox e Chrome in mobile



Date disponibili con trends del passato

Scelta data in Web broser su desktop, in mobile su Firefox e Chrome.



Data in formato ISO, detto anche militare (YYYY-MM-DD)



Locations con trends, paesi



Locations con trends, paese - città

Non selezionato, poi Italia, Milano su Firefox per mobile e UK, Londra su Chrome per mobile

2020-08-20

Worldwide

Trends List

- #Italyarexit
- #CDLChamps Tweet volume: 88894
- #FamiliasDeSpringfield
- #HaddiniBillSoysuz Tweet volume: 14302
- #Hodrimeydan Tweet volume: 21362
- #MissTV2
- #MAPPlayOffs Tweet volume: 87771
- #NBASundays
- #NFLBloopers Tweet volume: 31200
- #PeopleConversion Tweet volume: 33988
- #PROTECTCKKAI Tweet volume: 102220
- #RiyadSaqqaan Tweet volume: 38817

OK/s 88% 21:00

2020-08-31

trendrapp.github.io/

Italy / Milan

Location	Hashtag	Tweet Volume
Guatemala (1)		
India (21)		
Indonesia (11)		
Ireland (3)		
Israel (3)		
Italy (7)		
Bologna		
Genoa		
Milan	Milan	
Naples		
Palermo		
Rome		

#CDLChamps Tweet volume: 16127

#FamiliasDeSpringfield

#HaddiniBillSoysuz Tweet volume: 14302

#Hodrimeydan Tweet volume: 21362

2020-08-26

United Kingdom / London

Search Clear

Trends List

- #blacklivesmatter Tweet volume: 500552
- #engvpak Tweet volume: 12096
- #glazersout Tweet volume: 16935
- #gsbout
- #mondaythoughts

Risultati di trends del passato

2020-08-20

Italy / Milan

Search Clear

Trends List

- #30Agosto Tweet volume: 26020
- #20Agosto Tweet volume: 16332
- #years offittinca Tweet volume: 80423
- #BTS_Dynamite_ONWord Tweet volume: 408868
- #BayernOL
- #Brasile
- #Cittando
- #Covid_19 Tweet volume: 128039
- #DemConvention Tweet volume: 87378
- #FantasticoGigante
- #Gicola
- #HappyBirthdayDensi Tweet volume: 10487
- #LYOBAY Tweet volume: 26544

2020-08-31

Italy / Milan

Search Clear

Trends List

- #30Agosto Tweet volume: 24076
- #90giomiperinnamorarsi
- #Alaphilippe
- #Altofonte
- #BelgianGP Tweet volume: 142771
- #Diaco
- #Maldini

2020-08-26

United Kingdom / London

Search Clear

Trends List

- #blacklivesmatter Tweet volume: 500552
- #engvpak Tweet volume: 12096
- #glazersout Tweet volume: 16935
- #gsbout
- #mondaythoughts

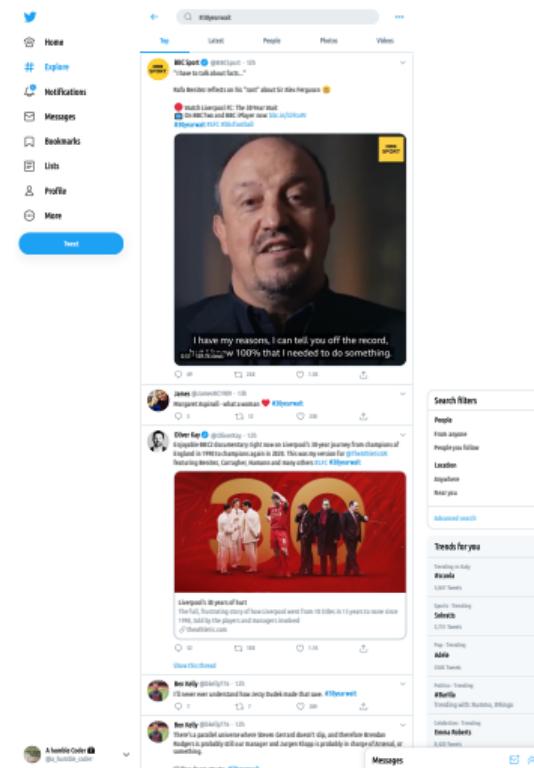
Lazy loading, infinite scroll

A screenshot of a mobile application interface. At the top, there is a date selector showing "2020-08-20" and a location selector showing "Italy / Milan". Below these are two buttons: a dark blue "Search" button and a red "Clear" button. The main area is titled "Trends List" and displays a list of trending topics with arrows to the right of each entry. The topics listed are: #unTemaAlGiorno, #upas, Crisi, Draghi (with a note: Tweet volume: 12212), Dzeko, Gabry (with a note: Tweet volume: 238679), Il Leone, Maldini, Montecarlo, Mussolini, Rangnick, Thiago Silva (with a note: Tweet volume: 13600), and bocca.

A screenshot of a mobile application interface. At the top, there is a date selector showing "2020-08-26" and a location selector showing "United Kingdom / London". Below these are two buttons: a dark blue "Search" button and a red "Clear" button. The main area is titled "Trends List" and displays a list of trending topics with arrows to the right of each entry. The topics listed are: kenyanne conway (with a note: Tweet volume: 152344), kimmich (with a note: Tweet volume: 14831), land of hope and glory (with a note: Tweet volume: 15168), and mbappe (with a note: Tweet volume: 111969). At the bottom of the screen, there is a horizontal scroll bar with a circular arrow icon indicating the ability to scroll through more content.

Apertura di un risultato dalla lista dei trends

Il trend si apre in una nuova tab o nel browser di default se in mobile.



Deployment & Demo

- Deployment della WebApp
- Live Demo

Deployment della Progressive WebApp su GitHub Pages

The screenshot shows the GitHub repository page for 'trendrapp/trendrapp.github.io'. The repository has 1 branch and 0 tags. The master branch has 4 commits. The deployment history shows several new deploys for files like _nuxt, trendingLoader, .nojekyll, 200.html, favicon.ico, index.html, and sw.js. The repository has 1 pull request, 0 issues, and 0 actions. It includes sections for About, Releases, Packages, Environments, and Languages.

About
Progressive WebApp, website deployment files. Visit <https://trendrapp.github.io/>

Releases
No releases published [Create a new release](#)

Packages
No packages published [Publish your first package](#)

Environments 1
github-pages Active

Languages
HTML 91.4% JavaScript 8.6%

Live demonstration del funzionamento della WebApp

page intentionally left blank

Crediti

- Ringraziamenti

Ringrazio...

Mauro Pelucchi, Prof.

per la sua prontezza e disponibilità nell'aiutare a capire meglio i concetti di Cloud Computing su Amazon Web Services e i suoi suggerimenti riguardo i trade-offs da accettare nella progettazione del programma.

That's all Folks!