

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

# **Bachelor Thesis**

# **GeoTwit – Progress Report**

## Table of contents

1. Introduction.....	4
1.1. Peer.....	4
1.2. Generalities.....	4
1.3. Work Environment.....	4
1.4. Project's Objectives.....	4
2. Specifications.....	5
2.1. Description of the project.....	5
2.2. Features.....	5
2.2.1. Tasks.....	5
2.2.2. Used Technologies.....	6
2.2.3. Coding Conventions.....	6
2.2.4. Future Extensions.....	6
3. Analysis.....	7
3.1. Existing Applications.....	7
3.2. Technologies.....	8
3.2.1. Scala.....	8
3.2.2. Play Framework.....	8
3.2.3. Twitter's APIs.....	9
3.2.3.a. Twitter's Technologies.....	9
3.2.3.b. OAuth.....	10
3.2.3.c. REST / Streaming APIs.....	11
3.2.3.d. Interesting Twitter's Subjects.....	22
3.2.4. Twitter4J.....	26
3.2.5. Geolocation Features.....	29
3.2.5.a. Google Maps.....	29
3.2.5.b. OpenStreetMap.....	29
3.2.6. Data Grouping.....	30
3.3. Prototype Applications.....	30
3.3.1. Leaflet.....	30
3.3.2. Leaflet – Countries' Borders.....	32
3.3.3. Twitter4JDesktop.....	34
3.3.4. Leaflet and Twitter4J.....	34
3.3.5. Discussion.....	36
4. The GeoTwit Application.....	36
4.1. Mock-Up.....	36
4.2. UML Diagram of the Application.....	36
4.3. Implementation details.....	36
4.3.1. Behavior of the Views.....	36
4.3.2. Twitter4J.....	36
4.3.3. Connection Process.....	36
5. Conclusions.....	36
5.1. Abstract.....	36
5.2. Review of the Current State.....	37
5.3. Encountered Problems.....	37
5.4. Final Discussions.....	37
6. General References.....	37

## Table of illustrations

Illustration 1: example of Twitter application's authorization.....	10
Illustration 2: use case of the REST API within a web application, taken from the documentation of the Twitter's APIs.....	11
Illustration 3: use case of the Streaming API within a web application, also taken from the documentation of the Twitter's APIs.....	12
Illustration 4: graphical representation of the coordinates pair bounding Switzerland, taken from Google Maps.....	14
Illustration 5: search of Tweets with keyword only.....	14
Illustration 6: search of Tweets with location only.....	15
Illustration 7: search of Tweets with keyword and location.....	15
Illustration 8: search of Tweets with keyword and a manual filtering of the location.....	15
Illustration 9: authorization of the API console through our Twitter account.....	20
Illustration 10: results of the latest French Tweets containing the “yverdon” keyword and (OR) being located near Yverdon-Les-Bains in the API console.....	21
Illustration 11: results on Twitter of the latest French Tweets containing the “yverdon” keyword and (OR) being located near Yverdon-Les-Bains.....	21
Illustration 12: results on Twitter of the popular and positive Tweets, having the “#pizza” hashtag and (OR) being located near to Rome.....	22
Illustration 13: tweets results for the UK-RU's match.....	26
Illustration 14: progression of the UK-RU's match, taken from the RTS web site.....	26
Illustration 15: Twitter application's parameters.....	27
Illustration 16: Twitter application's authorization's page.....	28
Illustration 17: PIN code generated from Twitter.....	28
Illustration 18: output example of a Twitter's desktop application.....	28

# 1. Introduction

First note: native French speaker, I decided to write my whole Bachelor thesis in English instead of French, as a personal goal and also for my near future (it can indeed prove to an employer I'm comfortable with the Shakespeare language). I tried to be as applied as possible, but it can still remain some grammatical typos so please excuse me in advance.

## 1.1. Peer

Student / Developer: Miguel Santamaria

Professor / Supervisor: Fatemi Nastaran

## 1.2. Generalities

<b>Name of the project</b>	GeoTwit
<b>Short description</b>	GeoTwit is an application, which allows you to visualize real-time activities on Twitter's subjects with a map.
<b>Start date</b>	22 February 2016
<b>End date</b>	29 July 2016 – 12:00
<b>Actual duration</b>	450 hours
<b>Oral defense</b>	Between 5 and 16 September 2016

## 1.3. Work Environment

This project is carried out as the final Bachelor thesis within the TIC department of the Computer Engineering sector of the Yverdon-les-Bains's HEIG-VD school, for the IICT institute. The student, supported by his supervisor, works alone on his project, at the level of both development and documentation. Notice that the subject is not confidential.

The project – selected in December 2015 – is spread across various areas, among which the web development as well as the intelligent analysis (data mining) and the data visualization.

## 1.4. Project's Objectives

The Bachelor thesis consists of a personal study of a technical and scientific problem.

It forces a student to work on his own project during a whole semester. The main objectives of this work are to allow the student to develop autonomous and management

skills, by analyzing, resolving and developing a problem.

In case of problems and to ensure a successful working, the student is supported by a supervisor – generally a professor from the HEIG-VD – who give him advices and check the progression of the project. At the end of the work an external expert will also judge the whole project and will be present during the student's oral defense.

## 2. Specifications

### 2.1. Description of the project

The goal of this project – GeoTwit – is to set up a web application allowing the users to enter two subject of their choice and to visualize real-time activities for these subjects on Twitter and on a geographic map. The application will allow many interesting uses, for example the real-time comparison of the people's inputs in the German and French parts of Switzerland during a time of Swiss votes.

### 2.2. Features

The application will provide the following functionalities:

- The reading of keywords and the selection of geographic areas on the map by the user.
- The retrieval of Tweets, using the Twitter's APIs.  
Notice that only a certain percentage of these Tweets have geographic information, necessary for the future operations; a first filtering will thus be operated here.
- The analysis and the filtering of Tweets in order to calculate the number of Tweets by areas and by subjects.
- The visualization of the results on a map.
- The interaction (zoom-in, zoom-out, etc.) with the map. The development of this feature will involve the introduction of appropriated algorithms (like Tweet grouping) to allow an appropriated visualization by the selected zoom level.

#### 2.2.1. Tasks

Here is the chronological list of the various achievements which will be operated during this project:

- Use of the Twitter's APIs, including:
  - Retrieval and filtering of Tweets by one or more keywords.
  - Retrieval of geographic information attached to a Tweet.

- Analysis of the percentage of Tweets having the geographic information (necessary for further processing) and the search of keywords having good results, in order to determinate the best keywords to use for properly testing the different project's features.
- Use of a cartographic library in order to use geographic maps.
- Development of the Tweet's acquisition and analysis parts.
- Development of the web interface of the application, allowing the user to enter subjects to compare, to choose the geographic areas on the map, to visualize results and to interact with the map.
- Development of a grouping algorithm in order to group Tweets by the map's zoom level.
- Tests and validation.

### **2.2.2. Used Technologies**

It was suggested to use the Scala language coupled with the Play framework, while noting other technological choices was quite possible.

Due to both my personal interests and a certain curiosity for unknown technologies in general, we decided that these choices will be retained.

By not knowing either of these technologies, I will firstly need to learn the Scala language as well as the use of the Play framework.

Notice that different web languages (like JavaScript in particular) could be used to complement the application.

### **2.2.3. Coding Conventions**

I will use the main coding conventions we saw during the HEIG-VD's courses as well as the Scala and Play conventions, among which:

1. Writing of the comments in English.
2. Use of the camelCase format in the code.
3. Writing of constants in uppercase.
4. Writing of indentations with the tabulation key.
5. Opening braces ('{') are on the same line as the instructions.

### **2.2.4. Future Extensions**

Here is a non-exhaustive list of various extensions, which could be developed if time allows it or in a hypothetical future development:

1. Implementation of an user management (inscription, connection) in order to backup the different obtained results.
2. Storage of the most successful keywords as well as the areas having the best geographic results within the application.
3. **Implementation of two alternatives to display the results:**
  1. **A real-time data refreshment, allowing the application to automatically receive and process new Tweets; this version will be implemented anyway.**
  2. **A static way, which will only display the results once without updating them.**
4. Addition of social features (sharing of results, notes, etc.).
5. Add a gamification system (allowing users to gain points and trophies by the results rate of their search, implementation of a general ranking, etc.).

Implemented extensions are marked as bold.

## 3. Analysis

### 3.1. Existing Applications

There is already many existing applications allowing an user to show Tweets on a map, among which :

1. **One Million Tweet Map** (<http://onemilliontweetmap.com/>): the one million tweet map application approaches what will be done during this project, by displaying real-time Tweets (filtered or not) on a map with grouping methods.
2. **TrendsMap** (<http://trendsmap.com/> - found via *mashable.com*): this application shows a real-time mapping of Twitter trends across the world. If you click on any word you will see a real-time stream of relevant Tweets. You have to register and / or pay to access more content.
3. **Tweetping** (<https://tweetping.net/>): shows real-time reception of Tweets all across the world on a map.
4. **TweetMap** (<https://worldmap.harvard.edu/tweetmap/>): with this tool you can visualize Tweets, which have been written in a given period, and watch some interesting graphs.
5. **Stweet** (<http://www.we-love-the.net/Stweet/> found via *mashable.com*): this service mashes up Twitter and Google Street View, providing a location-based look at the neighborhood Tweets.

6. **GeoChirp** (<http://www.geochirp.com/>): this app allows you to search for Tweets in a given location, but only in a static way (no streaming then).
7. **Tweet To Map** (<http://tweettomap.com/>): this web plug-in allows you to put markers on a map, based on Tweets' data ; as I am writing, it only supports static Twitter content, so I won't use it.

## 3.2. Technologies

This chapter contains descriptions of the main used technologies within the project.

### 3.2.1. Scala

According to Wikipedia, Scala is a multi-paradigm (object-oriented, imperative, concurrent and functional) programming language designed at the EPFL in 2003 by Martin Odersky. It has a strong and static typing discipline and runs on a JVM (Java Virtual Machine). As Scala heavily draws on Java, Java libraries may be used directly in Scala code and vice versa.

#### Motivations

I often heard about Scala in a good way from a lot of people and always wanted to try it out. I also enjoyed learning functional languages (like Haskell or Java8) as well as object-oriented ones during my formation. When I saw this project can be done with this language I decided to taking the risk.

### 3.2.2. Play Framework

Play Framework is an open-source and web framework created in 2007 by Guillaume Bort and allowing the developer to write web application with Java or Scala. As I decided to use Scala in my project this choice turns out to be a good one, as the framework is compatible with the language, is well-documented and appreciated by the community. My supervisor – Mrs. Fatemi – also well knows both the framework and the language.

#### Installation

The installation of the framework is really simple so I won't rewrite it here:

<https://www.playframework.com/documentation/2.5.x/Installing>.

#### Create a new project

To create a new Play's Scala project, just type “*activator new [APPLICATION-NAME] play-scala*”, where [APPLICATION-NAME] is the name of the application you want to create.

Once done, get in the new created folder and type “*activator*” to enter the Play console, in which you can simply compile your code and start the server with the “*run*” command!



Once the server is launched the code will automatically compile when you load the web page.

You can find a description of an application's layout right here:

<https://www.playframework.com/documentation/2.5.x/Anatomy>. Everything is well explained so I won't copy it here, but here are further explanation I discovered by doing some tests:

- The layout of the application's web pages is located in "app/views/main.scala.html".
- The first parameter of the @main instruction in a view contains the page's title.

### 3.2.3. Twitter's APIs

Before starting, notice that all the content of this chapter comes from the documentation of the official Twitter's development web site: <https://dev.twitter.com/overview/documentation>. Thus, many diagrams contained in this document are taken from this documentation.

Please also notice that Twitter offers a "Best Practices" page, which will be very helpful during the conception of the application, both to the security and the optimization:

<https://dev.twitter.com/overview/general>.

To create a new application using the Twitter API, you have to go there:

<https://apps.twitter.com/>.

#### 3.2.3.a. Twitter's Technologies

Twitter offers many tools to developers to facilitate the use of their technologies within applications, which include:

- **Fabric**: mobile development tool used to easily interact in a stable way with the social features of Twitter (sharing, connection, etc.).
- **Twitter for Websites**: widgets collection, which easily integrates Twitter's widgets (Tweet and "Follow" buttons, Tweets displaying, etc.) in a web site.
- **OAuth**: an authentication way used with the Twitter API for securing and authorizing the different executed requests. Notice that this is not a technology developed by Twitter but a free protocol. You can find more details about the this in the following chapter.
- **REST APIs**: provide a read and write access to the data of Twitter as a JSON format or via OAuth to use them in programming. They work like most of the APIs provided by services of this kind.
- **Streaming APIs**: those APIs continually send responses to the requests of the REST APIs trough a persisted HTTP connection. They can be useful to continually

display Tweets linked to a given hashtag and thus to automatically show new Tweets matching with the filter.

The various technologies I listed above are the ones that can potentially be interesting within the project. After a quick analysis it turns out that the tools that will be useful and helpful are the REST API and the Streaming API, coupled with the OAuth mechanism for the security. It is indeed currently (at least in the first instance) not planned to use mobile technologies and the widgets option is not wide and complex enough to allow us to use it.

### 3.2.3.b. OAuth

The current version (v1.1) of the Twitter API uses the version 1.0A of the OAuth protocol. The developer can choose between two ways of authentication within his application:

1. **Application-User authentication:** the most common form of resource authentication with APIs of this kind (Instagram, Facebook, Twitter, etc.). Signed requests both identify the identity of our application as well as the end-user we make API calls on behalf of.

In terms of being specific, the user using the application must have a Twitter account and have to connect it, then to authorize the application to access data through his account (according to the rights asking by the developer: Tweets reading, Tweets posting, etc.) in order to get an access token. Later, all executed requests will be signed by this token, linked to the current application and user. In order to use this method, the user must therefore have a Twitter account and have to trust enough the application to grant reading and/or writing rights on the Twitter's data.

#### Authorize GeoTwit to use your account?



##### This application will be able to:

- Read Tweets from your timeline.
- See who you follow.

##### Will not be able to:

- Follow new people.
- Update your profile.
- Post Tweets for you.
- Access your direct messages.
- See your Twitter password.

*Illustration 1: example of Twitter application's authorization*

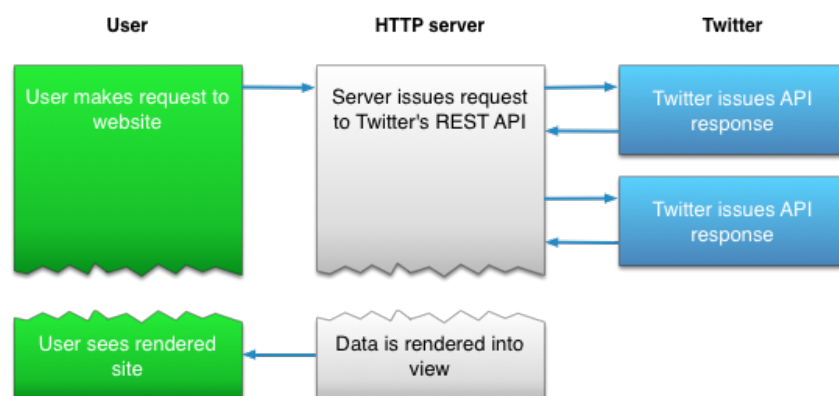
2. **Application authentication only:** the application executes the requests of the API on its own behalf without a user context. This method could at the first sight seem interesting but is unfortunately limited in the list of possible calls to the API. It is indeed possible to search for Tweets and get users information, but we cannot use either geographic data or streaming components...

Despite the ergonomic issues the first method (application-user authentication) could create, we will still keep it. Indeed, the second method does not allow us to use either geolocation or streaming, which is essential to the real-time visualization of Twitter's activities.

### 3.2.3.c. *REST / Streaming APIs*

#### Introduction

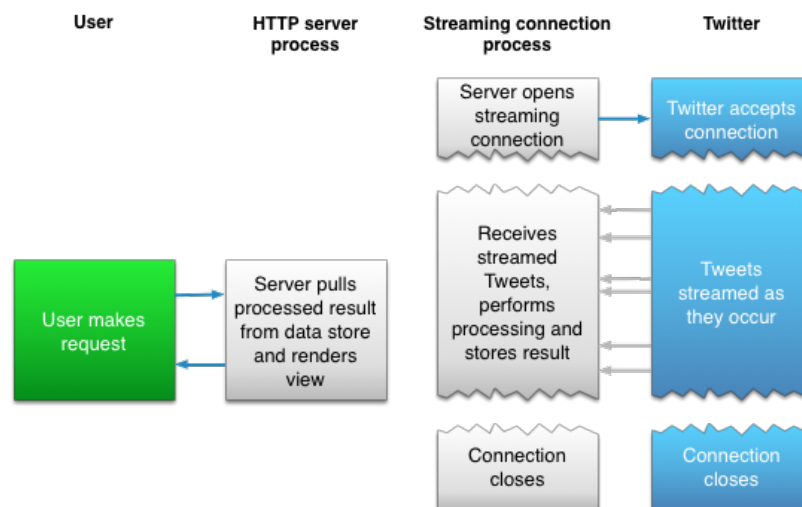
As a reminder, the REST API identifies the Twitter applications and the users by using OAuth and by providing JSON-formated responses. It allows us – among other things and according to the granted rights – to read and post Tweets, read the profiles of the Twitter's users, etc. The last released version (namely v1.1) works in the following manner:



*Illustration 2: use case of the REST API within a web application, taken from the documentation of the Twitter's APIs*

In a nutshell, the user makes requests executed by the server, which receives responses by Twitter and forwards them to the client.

Now let's have a look at the Streaming API's operation (which also uses OAuth and JSON), used to dynamically show results of search in real-time:



*Illustration 3: use case of the Streaming API within a web application, also taken from the documentation of the Twitter's APIs*

The operation is here a little more complex: the code maintaining the Streaming is executed in a separated process while another one is processing the HTTP requests.

There is 3 types of endpoints:

1. **Public streams:** stream of the public data flowing through Twitter, which can be filtered by keywords, users, etc.
2. **User streams:** single-user stream, roughly containing all of the data corresponding to a *single* user's view of Twitter.
3. **Site streams:** the *multi-user* version of the user streams.

For obvious reasons the public stream is the endpoint we will use (we need to analyze all Twitter's public data).

In order to follow the specifications of the project we will mainly use the Streaming API, in the sense that it has been asked to be able to visualize the activities of the Twitter's subjects in real-time. We though won't set aside the REST API, because it could be useful for the comprehension of the Streaming, which search algorithms are based on it. After some discussion with my supervisor, users will also be able to statically search Tweet within GeoTwit.

## Streaming API

To initialize a stream based on a filter we just have to execute a specific HTTP request (detailed above), indicating the Twitter's servers we want to connect. We then only need to maintain the connection open so the servers will send us data at regular intervals.

To know we are still connected, the Twitter's servers send us each 30 seconds a message indicating that the connection is still active (in case there would be no new actualities). If

no confirmation is received after 90 seconds we need to connect again to the stream.

Since there is no other way to test the Streaming API than developing a little application, I won't come to the question here. You can however go in the "Prototype Applications" chapter, in which I introduce and explain various prototype applications I made to test both Twitter APIs and maps libraries. Notice that search examples are though presented in the "REST API" part of this document, which you can find above.

## Limitations

In order to prevent the Twitter's servers to be overloaded by "churns" (a large number of requests of connections' opening and closing), Twitter restricts the possible number of connection requests in a certain time frame. Once a client exceeds this number, it will receive a HTTP response containing the 420 error code; for unclear reasons, Twitter privately keeps the maximum number of possible connections as well as the time frame value. Notice that if the limit is regularly exceeded, Twitter will block our IP address for a while, so we have to take this problem into account.

Please also notice that the streaming is discouraged with mobiles using cellular network, because of the "disconnection → connection" loop's high risk, which can create "churns".

Finally, notice that it is only possible to start two instances of the same application following a Twitter streaming with the same account and at the same time.

For more information about the connection methods, you can go here:

<https://dev.twitter.com/streaming/overview/connecting>.

## Subscription to Twitter's subjects

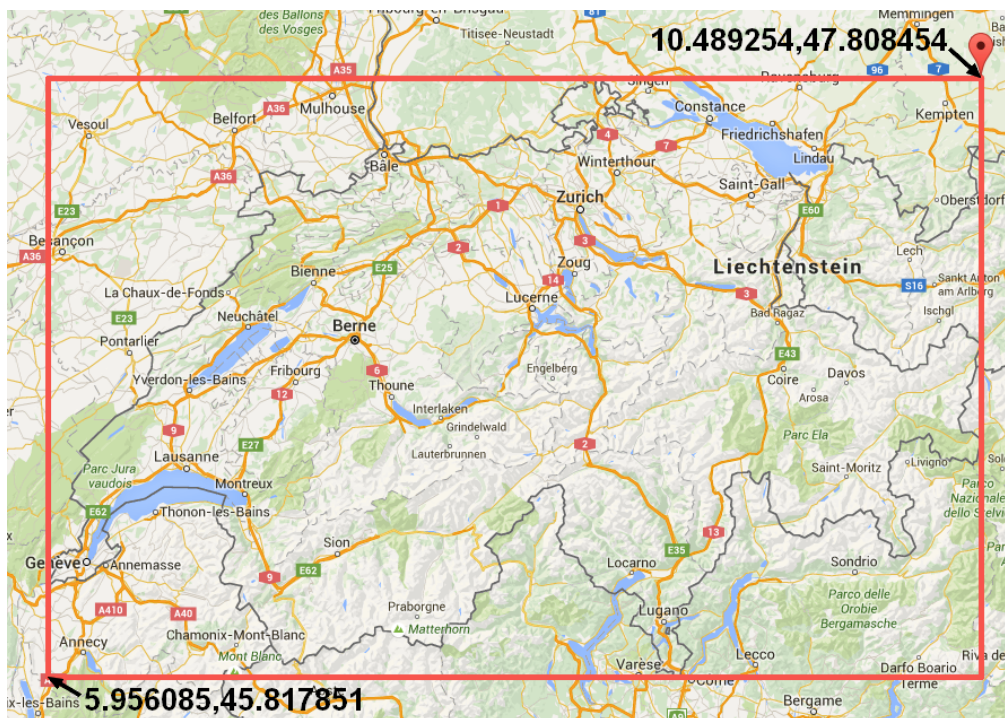
As stated above, it is necessary to make a HTTP request to indicate the servers of Twitter we want to subscribe to a particular stream. This request must/can have several parameters, which notably allow one to give various parameters to the server, like the wanted language of the Tweets, the filters (keywords, hashtags, etc.), the desired location, or other minor parameters. Those parameters are detailed here:

<https://dev.twitter.com/streaming/overview/request-parameters>.

Regarding to the location, the Streaming API uses bounding boxes. In other words, it is a rectangular area defined by a pair of geographic coordinates (a longitude and latitude pairs; be aware of the fact that we are more used to see coordinates in the "latitude, longitude" format), with the southwest corner of the bounding box coming first, followed by the northeast coordinate. It is possible to have many bounding boxes in one request, separated by commas.

For example, if we want to search Tweets in Switzerland, we have to give the following coordinates: 5.956085,45.817851 and 10.489254,47.808454.





*Illustration 4: graphical representation of the coordinates pair bounding Switzerland, taken from Google Maps*

Beware however, the location does not act as a filter for the other filters, which means the request “track=twitter&locations=5.956085, 45.817851, 10.489254, 47.808454” will search Tweets containing the “twitter” keyword OR being located in Switzerland.

To better understand the operation of the location “filter”, I conducted several tests. It is important to notice that those tests has been realized with the “twitter4jDesktop” prototype application (detailed on the “Prototype Applications” chapter above) and with the Streaming API. The used keyword was “michel” and the location was a rectangle bounding the U.S.A. ({-124.411668, 24.957884}, {-66.888435, 49.001895}). This tuple of keyword/location was used to avoid retrieving too much results and to (unsuccessfully) isolate information.

1. **Search with keyword only** → the obtained results are correct and correspond to the keyword:

```
hockey30.com : On dirait que Michel fait EXPRES... https://t.co/GUFreKJ5F2
ElyGoblin : #Blackfish "Maybe we have learned all we can from keeping them captive"~Jean-Michel
Patrick : RT @pathycvlcnt: "Lute pelos sonhos, busque seus objetivos; batalhe pelos seus ideais
Michel Telles
```

*Illustration 5: search of Tweets with keyword only*

However they are coming from all around the word, which restrict the use cases within the application...

2. **Search with the location only** → we're obtaining anything and everything here: indeed, most of the returned Tweets don't have geographic coordinates. This can be explained because of this “OR” operator applied on the search: the application is going to display all Tweets coming from the location area OR the Tweets corresponding to the keyword (namely an empty keyword in our case, so any Tweet...). In summary, absolutely all Tweets are displayed...

```
NO GEOLOC: Alex Herrera : I feel like a have a fever or something ☹
NO GEOLOC: deed : deed is making me hike angels landing today & I'm so scared ☹
-73.6137874,40.656401 => TMJ-NY Retail Jobs : CVS Health: Retail Store Positions
NO GEOLOC: sam☹ : @TylerDurfeell @SexualGif ILL BE HOME SOON☹
```

*Illustration 6: search of Tweets with location only*

3. **Search with keyword AND location** → at first sight we can expect this method to provide good results, but no... It is indeed a simple application of the operator OR: the application is going to display all Tweets having the keyword “michel” OR written in the U.S.A. (without taking in account the keyword...), which will give incongruous results again.

```
jas : I need to stop leaving the house with wet hair :-)
val : @hayl_x0x0 believe it cause it's the truth
Pat Green #13 : Shoot the house up and then Dip Dip Dip https://t.co/0qFjsbB2be
```

*Illustration 7: search of Tweets with keyword and location*

The Tweets don't necessary having a geolocation tag (I did some tests and obtained a rate of about 20-30% of Tweets having geographic data), we also want to filter the received results on-the-fly to only accept those which have one.

None of these search methods gave appropriate results... This is a huge problem, because we wish to search Tweets by a keyword AND a location in the application. Therefore, I will have no other option but to code my own location filter, which will be applied to the results of a search by keywords. In this example I used the “job” keyword in the U.S.A.:

```
-122.0363496,37.36883 => San Jose Eng. Jobs : #Engineering #Job in #Sunnyvale, CA: CAD Design
-86.7844432,36.1658899 => TMJ-BNA Retail Jobs : We're #hiring! Read about our latest #job ope
-93.269686,44.9754804 => RH Finance Jobs : Robert Half Finance & Accounting #Accounting #Job:
```

*Illustration 8: search of Tweets with keyword and a manual filtering of the location*

Two different approaches are possible here:

1. Search Tweets by keywords and (in reality OR) geolocation, and thereafter apply a manual filtering by keywords AND by geolocation for countering the logic filter OR. Here are the results I got in two periods of 5 minutes with the “job” keyword and in the U.S.A. (by around 15:30, Swiss time zone):

*“I received 8061 Tweets in 300 seconds, including 314 ones WITH geolocation tags*

*and 284 ones with the wanted geolocation.*

***This means 3.9 % of the received Tweets with the "job" tag(s) owned a geolocation tag and 3.52 % contained the desired location.***

*I received 6980 Tweets in 300 seconds, including 225 ones WITH geolocation tags and 215 ones with the wanted geolocation.*

***This means 3.22 % of the received Tweets with the "job" tag(s) owned a geolocation tag and 3.08 % contained the desired location."***

Results are here disappointing with only about 3-4% of Tweets having a geolocation tag.

2. Search Tweets by keywords, and then apply a manual filtering of the geolocation.

*"I received 2290 Tweets in 300 seconds, including 548 ones WITH geolocation tags and 516 ones with the wanted geolocation.*

***This means 23.93 % of the received Tweets with the "job" tag(s) owned a geolocation tag and 22.53 % contained the desired location.***

*I received 2774 Tweets in 300 seconds, including 571 ones WITH geolocation tags and 535 ones with the wanted geolocation.*

***This means 20.58 % of the received Tweets with the "job" tag(s) owned a geolocation tag and 19.29 % contained the desired location."***

Results are way more convincing here, with about 20% of Tweets having the desired geolocation tag. Even if we received about 3 to 4 times less Tweets than before, we still received more interesting Tweets. Thus, this solution will be the one I will keep.

Within GeoTwit, it could also be also possible to propose default locations (for example "in the German part of Switzerland", which would point a little on the East of Switzerland with pre-calculated coordinates; we could also imagine default locations by country or states). Many libraries/API exists and allows us to easily get coordinates by a pointed location for example. The most well known of them undoubtedly is Google Maps, but other alternatives like OpenStreetMap exists and will be analyzed below.

In addition of receiving JSON messages containing Tweets' data (formatted in the following way: <https://dev.twitter.com/overview/api/tweets>), it is also possible to receive other kind of messages from the servers of Twitter, for example to indicate a deletion of Tweets or of geolocation tags, a disconnection, a limit reached, etc. All messages' types are available right here: <https://dev.twitter.com/streaming/overview/messages-types>.

## Optimizations

In order to reduce the bandwidth, it is possible to ask compressed Gzip data to the servers with the following HTTP header: "Accept-Encoding: deflate, gzip".



Twitter also warns us about the fact that it is possible to receive a huge amount of data during worldwide or important cultural events. It is thus advisable to make tests in order to control if our application can carry this massive flow and take measures in consequences.

Also, for optimization reasons, the servers use the “Chunked transfer encoding” data transfer mechanism, which – according to Wikipedia – means they start to send data in a series of “chunks” without knowing the length of the content before starting transmitting a response to the receiver. Our client must so be compatible with this type of transfer.

Be finally aware of the fact that the received messages are not necessarily ordered and that a message can be received several times.

## REST API

Here are some analysis below about the REST API, which will still be useful in the future of the project, since there will be a static search.

## Limitations

First of all, it should be noted than the API has some limitations to ensure the security of the platform, and to avoid the Twitter's servers to be overloaded by requests. Limitations are mainly used on the number of requests that can be performed by periods of 15 minutes and by window. Notice that if our application is too regularly limited it may be put on black list, which will consequently forbid the access to the API and therefore the possibility to make other requests. For this reason we will need to properly optimize the number of sent requests.

When an application exceeds the rate limit of authorized requests, the Twitter's API send back a “429: Too Many Requests” HTTP response code, which allows us to indicate the user he reached the limit of requests imposed by Twitter for the current type of request.

The HTTP headers contain, among other things, information concerning the current rates of made and remaining requests:

- X-Rate-Limit-Limit : indicates the rate limit of requests for the type of given request.
- X-Rate-Limit-Remaining : indicates the number of remaining requests for the type of given request and for the current period of 15 minutes.
- X-Rate-Limit-Reset : the remaining time before the next reset of the 15 minutes period.

The limit of each type of request can be accessed here:

<https://dev.twitter.com/rest/public/rate-limits>.

## Tweets search

The search API belongs to the Twitter's REST API. It allows us to search Tweets by various criteria (latest or popular Tweets containing or not keywords or hashtags, filtered by language and by geolocation, etc.). Due to the massive quantity of data to process and therefore for performances reasons, search only takes in account Tweets published during the 7 last days preceding the request.

Many search filters exists and are accessible on the following web page: <https://dev.twitter.com/rest/public/search>. Most of these parameters are also compatible with the Streaming API. Among them, we will mainly be interested about the ones regarding the search of keywords and hashtags.

In addition of these filters, other parameters allowing a better control on the search exist, such as the type of results (the latest and/or most popular Tweets), the geolocation, the language of the Tweet or even parameters allowing to iterate on Tweets in an easily way (*count*, *until*, *since\_id*, *max\_id*).

Regarding the location, this API directly uses the geolocation. First thing first, it is strictly speaking not possible to search near a specific location: we have to use a specific geocode in the "latitude,longitude,radius" format, where *radius* is the search radius in miles (*mi*) or in kilometers (*km*), having for center point the given latitude and longitude. The API will firstly try to get Tweets containing information about the wanted latitude, longitude and radius, and then in a second time if there is no result, will search for Tweets whose users' profiles match with the desired geolocation parameters.

For example, if we want to search Tweets in a radius of 50 kilometers around the main station of Yverdon-les-Bains (whose coordinates have been found on Google Maps), the parameters will have the following values: "46.783177,6.640630,50km". Ideally within the project, the user should just point a location on a map and indicate the radius he wants, resulting an automatically acquirement of the longitude and latitude values associated to this location in the background.

Notice that it is also possible to search for Tweets by the ID of locations. For example, the Tweets emitted from the Twitter's headquarters can be found with the "place %3A247f43d441defc03" parameter. More information can be found here: <https://dev.twitter.com/rest/public/search-by-place>. This parameters will certainly be less useful than the geolocated search, but still deserves to be on this documentation.

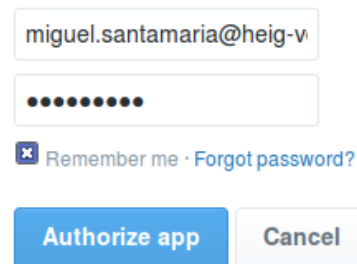
## Examples

For the purpose of being able to easily generate search requests, the advanced search tool (<https://twitter.com/search-advanced>) was used for the coming couple of examples. Whenever a search is made, we are redirected on a web page containing a URL, which is

built on a particularly format. For example, with a search about the @twitterapi user, we get the following URL off redirection: <https://twitter.com/search?q=%40twitterapi>. Once done, we can get the valid URL to use in the API by replacing “https://twitter.com/search” by “https://api.twitter.com/1.1/search/tweets.json” (in our example → <https://api.twitter.com/1.1/search/tweets.json?q=%40twitterapi>).

In addition, the various tests have been conducted with the API console tool provided by Twitter right here: <https://dev.twitter.com/rest/tools/console>. Beforehand and in order to be able to make requests, we have to authorize the console to access our own Twitter account. This access is automatically asked as soon as we try to make the first request.

## Authorize Apigee's API Console to use your account?



miguel.santamaria@heig-v

.....

☒ Remember me · [Forgot password?](#)

**Authorize app** Cancel

### This application will be able to:

- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.
- Access your direct messages.

### Will not be able to:

- See your Twitter password.

*Illustration 9: authorization of the API console through our Twitter account*

Here is a list containing a few examples, and which can turn out to be subsequently useful:

1. Search of the latest French-written Tweets having “yverdon” as keyword and (in reality OR, like we saw before) being located in a radius of 10 kilometers from the Yverdon-les-Bains' main station.
  - Request: [https://api.twitter.com/1.1/search/tweets.json?q=yverdon&geocode=46.783177,6.640630,10km&lang=fr&result\\_type=recent](https://api.twitter.com/1.1/search/tweets.json?q=yverdon&geocode=46.783177,6.640630,10km&lang=fr&result_type=recent)

- Results:

**Request URL**

GET /api.twitter.com/1.1/search/tweets.json?q=yverdon&geocode=46.783177,6.640630,50km&lang=fr&result\_type=recent HTTP/1.1

**Request**

```

Authorization: OAuth
oauth_consumer_key="DC0seP0BbQ8bYdC8r4Smg",oauth_signature="SHA1",oauth_timestamp="1457370699",oauth_nonce="2509176"
Host: api.twitter.com
X-Target-URI: https://api.twitter.com
Connection: Keep-Alive
  
```

**Response**

```

{
  "statuses": [
    {
      "metadata": {
        "iso_language_code": "fr",
        "result_type": "recent"
      },
      "created_at": "Thu Mar 03 10:49:40 +0000 2016",
      "id": 705344385530003500,
      "id_str": "705344385530003456",
      "text": "au bistrot ! Matt @ Plage d'Yverdon https://t.co/ubpIKSyNV",
      "source": "<a href='\"http://instagram.com\"' rel='\"nofollow\">Instagram</a>",
      "truncated": false,
      "in_reply_to_status_id": null,
      "in_reply_to_status_id_str": null,
      "in_reply_to_user_id": null,
      "in_reply_to_user_id_str": null,
      "in_reply_to_screen_name": null,
      "user": {
        "id": 174194582,
        "id_str": "174194582",
        "name": "Philippe Jung",
        "screen_name": "PhilippeJung",
        "location": "Ste - Croix",
        "description": "Fondateur - Directeur - Photographe de l'Agence PHOTOS-PEOPLE",
        "url": "http://t.co/z190k9HhZM",
        "entities": {
          "url": {
            "urls": [
              {
                "url": "http://t.co/z190k9HhZM",
                "expanded_url": "http://t.co/z190k9HhZM",
                "display_url": "t.co/z190k9HhZM",
                "indices": [0, 23]
              }
            ]
          }
        }
      }
    }
  ]
}
  
```

Illustration 10: results of the latest French Tweets containing the “yverdon” keyword and (OR) being located near Yverdon-Les-Bains in the API console

- Results on Twitter:

**RVN - Radio Nord VD** @RVN\_Suisse · 1 h  
Retrouvez Raphaël et Gil en direct du Café Restaurant du Stade à **Yverdon** pour le premier Barathon des Brandons d'**Yverdon** 2016 de 17 à 19h!

**La Région NV** @LaRegionNV · 14 h  
Les représentants d'**Yverdon**-les-Bains ont été séduits par Kindercity: Zurich - Le centre ludo-éducatif, qui so... [bit.ly/1YkB2IH](http://bit.ly/1YkB2IH)

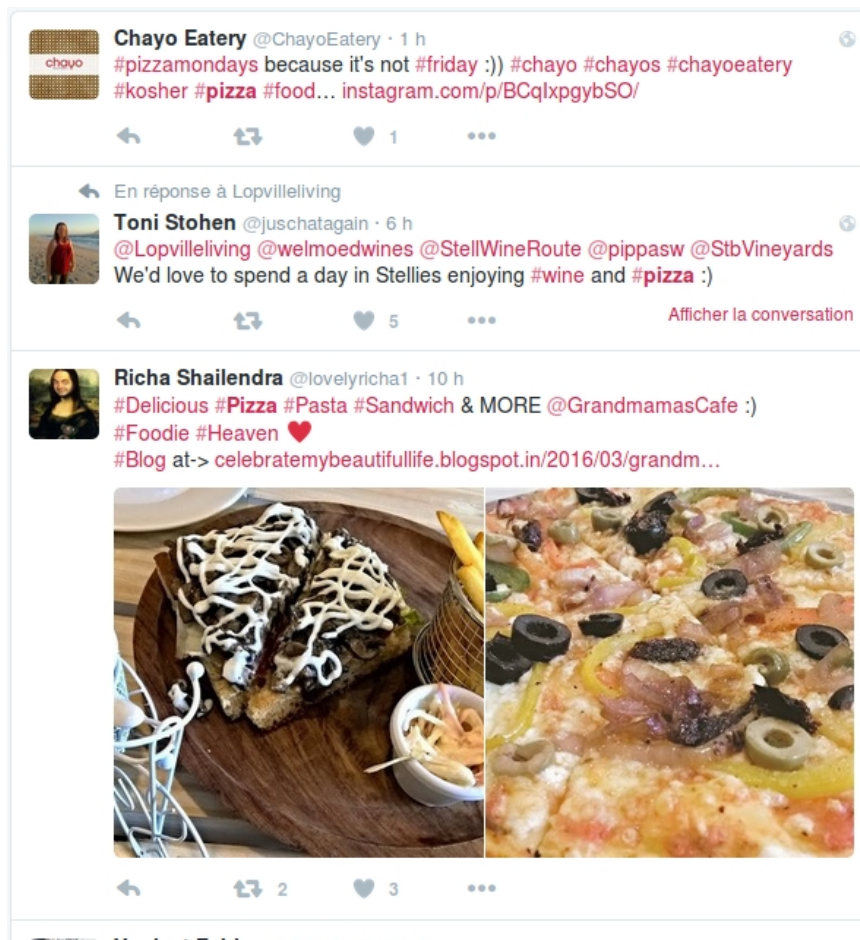
**24heures Vaud** @24heuresVaud · 6 mars  
L'Entraide familiale recycle en plus grand à **Yverdon** [bit.ly/1Qv5rko](http://bit.ly/1Qv5rko) #Vaud

**Norbert Roseau** @NorbertRoseau · 4 mars  
Jacques Dutronc - Live in **Yverdon**-les-Bains, Switzerland '66 [youtu.be/hMKa7nPa1-w?li](http://youtu.be/hMKa7nPa1-w?li) via @YouTube

Illustration 11: results on Twitter of the latest French Tweets containing the “yverdon” keyword and (OR) being located near Yverdon-Les-Bains

2. Search of popular and positive (containing a happy “:)” smiley) Tweets, having the “#pizza” hashtag and (OR) being located in a radius of 100 kilometers from the center of Rome (without taking care of the language).

- Request: <https://api.twitter.com/1.1/search/tweets.json?q=%23pizza%20%3A%29&geocode=41.894143,12.495479,100km&src=popular>
- Results on Twitter:



*Illustration 12: results on Twitter of the popular and positive Tweets, having the “#pizza” hashtag and (OR) being located near to Rome*

### 3.2.3.d. Interesting Twitter's Subjects

This section contains various Twitter's subjects, which can be interesting in the project. These tests have been conducted with the “twitter4jDesktop” application on April 25<sup>th</sup> 2016 between 13:00 and 16:30 (Swiss hours) for the Europeans countries and on **TODO** for the U.S.A.

Subject	Geolocation (location, coordinates)	Nb. of new Tweets in 5 min.
zurich	Switzerland, ({5.956085, 45.817851}, {10.489254, 47.808454})	0
fondue		0
logitech		0
nestlé		1
swatch		0
job		1
sport		0
people		0
containing a 'e'		0
bbc	UK, ({-8.306947, 49.696022}, {1.801128, 59.258967})	0
<b>london</b>		<b>15</b>
paris	France, ({-4.805145263671875, 42.34528267746347}, {8.232879638671875, 51.09052797518529})	5
roma	Italy, ({6.6357421875, 47.09805038936004}, {18.6328125, 36.577893995157474})	0
pizza		0
pizza		3
<b>job</b>		<b>618</b>
sport		0
people		4
vote		
trump		
clinton	U.S.A., ({-124.411668, 24.957884}, {-66.888435, 49.001895})	
logitech		0 or 1
nestlé		
novartis		
swatch		
mcdonald's		
java		
scala		

As regards the recurring themes related to Switzerland, I thought about two main categories: the Swiss medias and the active companies. I conducted some bad-resulting tests about companies with the following keywords: “coop”, “glencore”, “logitech”, “migros”, “nestlé”, “novartis” and “swatch”. I deliberately did not put all these words in the table above, because of the insufficient refreshment rate (zero or one Tweet per 5 minutes).



Concerning the medias, I endeavored to think about recurrent keywords, which could be used in the whole Switzerland (French, German and Italian parts) like “job”, “sport” or “people”, but without success: results were indeed bad again. In a desperate attempt to find Tweets with geolocation in Switzerland, I searched for Tweets containing the 'e' letter (which is pretty common) and received here again zero Tweets, so I decided to move to other countries. I took other European countries first, like UK, France or Italy: most of the tested keyword were still unsuccessful, but I found some of them giving better results, like some of the main European cities.

After all these failures I decided to move to the U.S.A., which are pretty known to use social medias in a stronger way than Europe. I was testing again all the previous keywords when I came across the holy grail: “job”. I received more than 600 results in a 5 minutes period. Surprised, I tested again and received 618 results again! I also tried more typical American subjects – like “vote”, “trump”, “clinton”, “mcdonald's” – and computer subjects like “java” or “scala” but they were way less interesting. TODO

In conclusion, results are pretty bad. We got an average of better results when searching in the U.S.A., especially for the “job” keyword, which is the only keyword giving good results. Switzerland has been quickly eliminated of the interesting countries as well as most of the other European countries, due to zero rate of Tweets having geolocation tags.

In addition to these few examples, we could think about other interesting cases linked to particularly events: a keyword corresponding to a subject of Swiss votes during the Sunday in which results are given, the name of a candidate in the U.S.A.'s elections during the results day (be careful about the massive flow of data!); the name of a show during its transmission to the TV; the name of a sporting event's winner; etc.

Finally, here is a little “planning” of hot and interesting coming topics until the end of the semester:

Date	Subject
17.03.2016	Hockey : LHC – HC Ambri-Piotta
17.03.2016	St. Patrick
27.03.2016	Easter
21.04.2016	90 years of the Queen Elizabeth
23.04.2016	400 years of Shakespeare
23.04.2016 to 24.04.2006	20KM of Lausanne
08.05.2016	Football : FC Sion – BSC Young Boys
09.05.2016	Transit of Mercury in front of the Sun
11.05.2016 to 22.05.2016	69 <sup>th</sup> Cannes Festival
14.05.2016	Eurovision in Stockholm



16.05.2016 to 05.06.2016	Tennis : Roland-Garros
29.05.2016	Football : Final of the “Coupe de Suisse”
05.06.2016	Swiss votes
10.06.2016 to 10.07.2016	UEFA Euro 2016
29.06.2016 to 04.07.2016	“Tour de France” 2016
01.07.2016 to 16.07.2016	50 <sup>th</sup> Montreux Jazz Festival
19.07.2016 to 24.07.2016	Paléo Festival Nyon
20.07.2016	“Independence Day Resurgence” film release
22.07.2016	“Star Trek Beyond” film release
01.08.2016	National day of Switzerland
04.08.2016 to 14.08.2016	“Fêtes de Genève”
05.08.2016 to 21.08.2016	Olympic Games of Rio de Janeiro
07.09.2016 to 18.09.2016	Paralympic Games of Rio de Janeiro
25.09.2016	Swiss votes
29.10.2016 to 30.10.2016	Lausanne's marathon
08.11.2016	Elections in the U.S.A.

This non-exhaustive list can be completed with other events during the whole semester.

### **Euro 2016**

I thought it was interesting to do data tests on Twitter during the Euro 2016's games. In order to achieve this, I used the “leafletAndTwitter4J” prototype application (see the well-named “Prototype Applications” chapter for more information):

Search information	Results
<p>On <b>11.06.2016</b>  <b>22:00 → 23:00</b>  During the second half of the match</p> <p>Country: <b>UK</b></p> <p>Keyword: <b>euro2016</b></p>	<div data-bbox="475 365 954 1025"> </div> <p>Score: <b>1-1</b></p> <p><b>47</b> Tweets with geolocation tags.</p> <p>Average of <b>6'000-7'000</b> Tweets per 5 minutes until the first goal (73'), then <b>10'000-12'000</b> after it and until the end.</p> <p>I received a massive flow of 16'000 Tweets after the goal of the 92'.</p> <p>Between <b>0.01%</b> and <b>0.07%</b> of Tweets had a geolocation tag (between 1 and 10 Tweets per 5 minutes).</p> <div data-bbox="475 1131 1422 1238"> </div> <p><i>Illustration 13: tweets results for the UK-RU's match</i></p> <p><i>Illustration 14: progression of the UK-RU's match, taken from the RTS web site.</i></p>

### 3.2.4. Twitter4J

Twitter4J (<http://twitter4j.org/en/index.html>) is a non-official library, which – as its name implies – allows us to easily use the features of the Twitter's APIs with Java. Given that Java classes are fully compatibles with the Scala classes and vice-versa, it turns out that this library is easily usable with the Scala programming language.

It works both with the OAuth technology and the Gzip compression and is 100% compatible with the version 1.1 of the Twitter's API. The advantages of this library lies in the fact that it significantly simplifies the APIs uses (for both Rest and Streaming APIs), at the level of both data reading and writing. The documentation of this library have examples of using for each specific case, as well as many useful explanations.

## Installation

When using Play Framework, just add **"org.twitter4j" % "twitter4j-core" % "4.0.4"** in the library dependencies in the `"/build.sbt"` configuration file of your application.

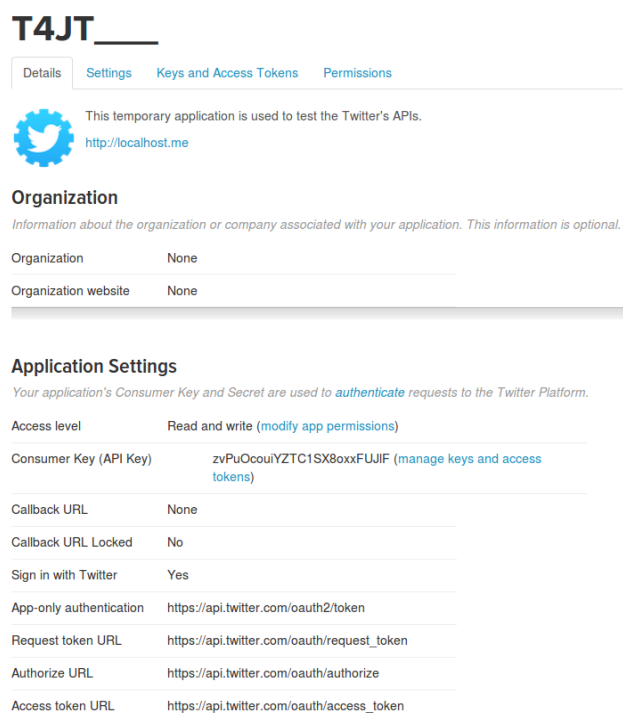
If you are using a Java or Scala desktop application, download the latest stable version of Twitter4J, then just add `"lib/twitter4j-core-4.0.4.jar"` to your application class path. If you want to use other features (like streaming or asynchronism) add the other libs too (like `"lib/twitter4j-stream-4.0.4.jar"` for the streaming).

## Create a New Application

In order to use the Twitter's APIs on a new application you firstly have to create and register a new application on Twitter (<https://apps.twitter.com/app/new>).

After you did this, you will access a web page, which contains all your application's parameters.

First be sure the access level of the application is the one you want. Then you can easily get all keys and access tokens in the well-named tab, in order to use them in your application.



The screenshot shows the Twitter application settings page for an application named 'T4JT'. The page has four tabs: Details, Settings, Keys and Access Tokens, and Permissions. The 'Settings' tab is active. It displays the following information:

- 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 Consumer Key and Secret are used to [authenticate](#) requests to the Twitter Platform.
 

Access level	Read and write ( <a href="#">modify app permissions</a> )
Consumer Key (API Key)	zvPuOcoiYZTC1SX8oxxFUJIF ( <a href="#">manage keys and access tokens</a> )
Callback URL	None
Callback URL Locked	No
Sign in with Twitter	Yes
App-only authentication	<a href="https://api.twitter.com/oauth2/token">https://api.twitter.com/oauth2/token</a>
Request token URL	<a href="https://api.twitter.com/oauth/request_token">https://api.twitter.com/oauth/request_token</a>
Authorize URL	<a href="https://api.twitter.com/oauth/authorize">https://api.twitter.com/oauth/authorize</a>
Access token URL	<a href="https://api.twitter.com/oauth/access_token">https://api.twitter.com/oauth/access_token</a>

Illustration 15: Twitter application's parameters

## Desktop Applications

Desktop applications are a little bit trickier to use with the Twitter's APIs, because you have to manually get accesses and you have to indicate the API that you don't want it to redirect you on a web page after the connection phase.

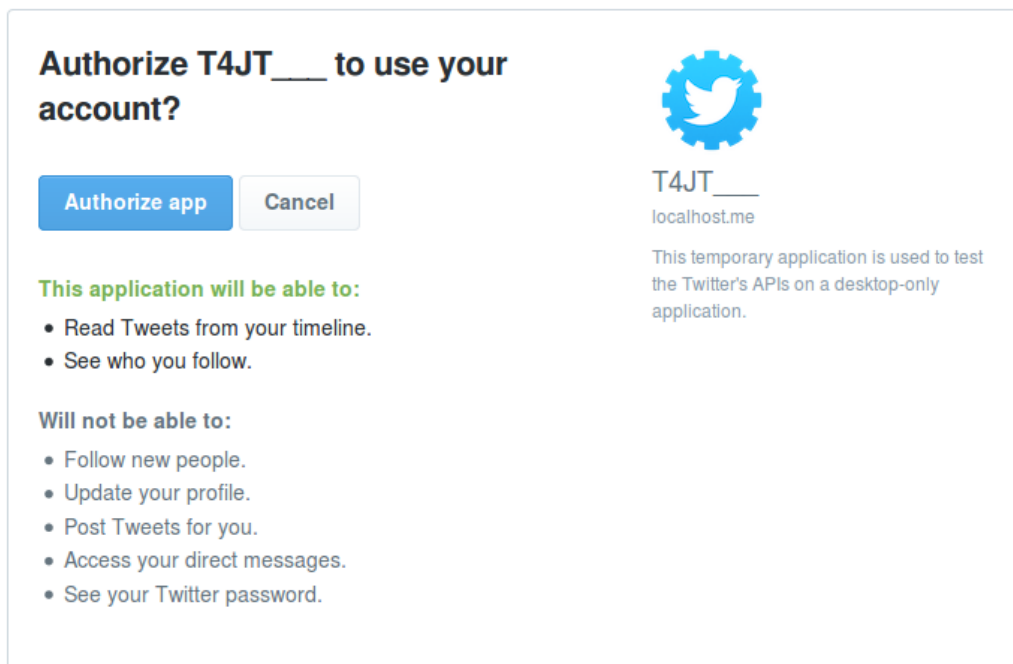
For the mentioned reasons we have to use the PIN-based OAuth; you can find more information about the ways to do it in these two links: <https://dev.twitter.com/oauth/pin-based> and <https://dev.twitter.com/web/sign-in/implementing>. Here is the procedure I used:

1. Create a new app on the "Twitter for developers" web site (see "Creating a new

application” chapter above).

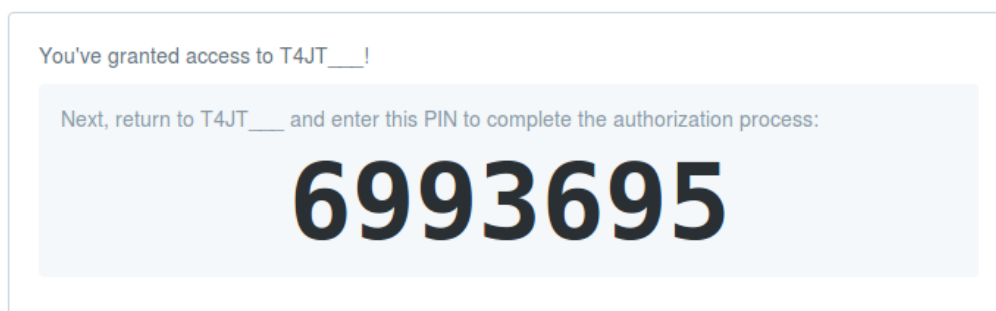
2. Use the Twitter4J library to make a token request and get an access token (<http://twitter4j.org/en/code-examples.html>, on the “OAuth support” section). You can find an example in the “GeoTwit/prototypes/twitter4jDesktop” folder.

If you're doing it right you have to ask the current user to grant an access to your application through a web page's link. Once he clicked on it, he will access an authorization's page:



*Illustration 16: Twitter application's authorization's page*

Once he authorized the app, he will get a PIN to write in the application:



*Illustration 17: PIN code generated from Twitter*

Here is an example of a right output:

```
debug:
Open the following URL and grant access to your account:
https://api.twitter.com/oauth/authorize?oauth_token=v6inWQAAAAAuPXSAABU5no9GU
Enter the PIN(if available) or just hit enter.[PIN]:6993695
Yay! Token successfully got: AccessToken{screenName='Migwelsh28', userId=533045969}
BUILD SUCCESSFUL (total time: 29 seconds)
```

*Illustration 18: output example of a Twitter's desktop application*

3. Once you got the token, you can simply use it for your next requests!

### **3.2.5. Geolocation Features**

In order to make the web site as ergonomic as possible and thus to be able to display and work with geographic maps, we will need to use a library. Two technologies will be analyzed in this section: the unmistakable Google Maps, and one of its main open-sourced rival, OpenStreetMap.

#### **3.2.5.a. Google Maps**

The Google Maps' API (<https://developers.google.com/maps/?hl=fr>) is very intuitive and user friendly (as I already used it in the past), but is however very problematic at the limitations level: if we indeed wish to use the site in a proper way, we will need to pay for this service in one way or another, in order to not be limited at the requests rate level, which is hardly practicable within a Bachelor thesis. For this reason, I won't draw further analysis on this library.

#### **3.2.5.b. OpenStreetMap**

OpenStreetMap (currently in version 0.6) – proposed as an alternative by Mrs. Fatemi – is a direct and open-sourced competitor of Google Maps, and also have an API ([http://wiki.openstreetmap.org/wiki/API\\_v0.6](http://wiki.openstreetmap.org/wiki/API_v0.6)). In fact, there is two different APIs : the live-API (<http://api.openstreetmap.org/>) and another used for tests (<http://api06.dev.openstreetmap.org/>).

Although this technology is open sourced, this does not mean that it is obsolete or slower: for example, it also uses the now well-known OAuth protocol and is used in a lot of applications, like Leaflet and MapQuest (you can find a short list of applications right here: [http://wiki.openstreetmap.org/wiki/List\\_of\\_OSM-based\\_services](http://wiki.openstreetmap.org/wiki/List_of_OSM-based_services)).

This technology gets very interesting for this project when coupled to a JavaScript library – also open-sourced and very light – allowing to easily manipulate maps and add effects on them: Leaflet (<http://leafletjs.com/index.html>). In our case, the interest of this library – in addition to the main features offered by the map – especially lies in the fact that it allows us to easily communicate with the OpenStreetMap's API. We can also easily draw circles (by a latitude, a longitude and a given radius), rectangles or custom polygons on the map (usable for geolocation features; you can find an example here: <http://leafletjs.com/examples/quick-start.html>) and add annotations as panels. Also, it is fully compatible with the mobile word, which is interesting for scalability issues.

Due to the above reasons, these technologies will be the ones I will use within the project for all the geographic requirements.

### 3.2.6. Data Grouping

Within the project, we are going to deal with two major massive data issues :

1. When sending data from our server to our client: when I did my tests, I sometimes got issues when I lost my Internet connection for a few second. As soon as I got my connection again, the server started to send one by one thousands of Tweets to the client in a very limited time, making my web browser crash... To avoid this case we will have to use an algorithm to group data before sending them to the server, if there is more than a certain number of data to send.
2. When displaying data on the map: since there will maybe be thousands of Tweets to display on a map, we will have to group them again with a grouping algorithm. I analyzed some existing solutions, and one of the most interesting one was without a doubt the marker-cluster plug-in proposed by Leaflet again (<https://github.com/Leaflet/Leaflet.markercluster>), which allows us to group data by circles on the map. This library provides a lot of interesting functions, is fully compatible with the Leaflet library (because Leaflet wouldn't develop libraries incompatible with their main library, you know...), and can handle up to 50'000 marker in one map! You can find examples here: <http://leaflet.github.io/Leaflet.markercluster/example/marker-clustering-realworld.10000.html> and <http://leaflet.github.io/Leaflet.markercluster/example/marker-clustering-realworld.50000.html>.

## 3.3. Prototype Applications

In order to test various libraries that will be used within the project and as discussed before in this documentation, I developed some prototype applications containing interesting tests. Please notice that there is no inputs controls and the GUIs are very poor, because they only are test-applications.

### 3.3.1. Leaflet

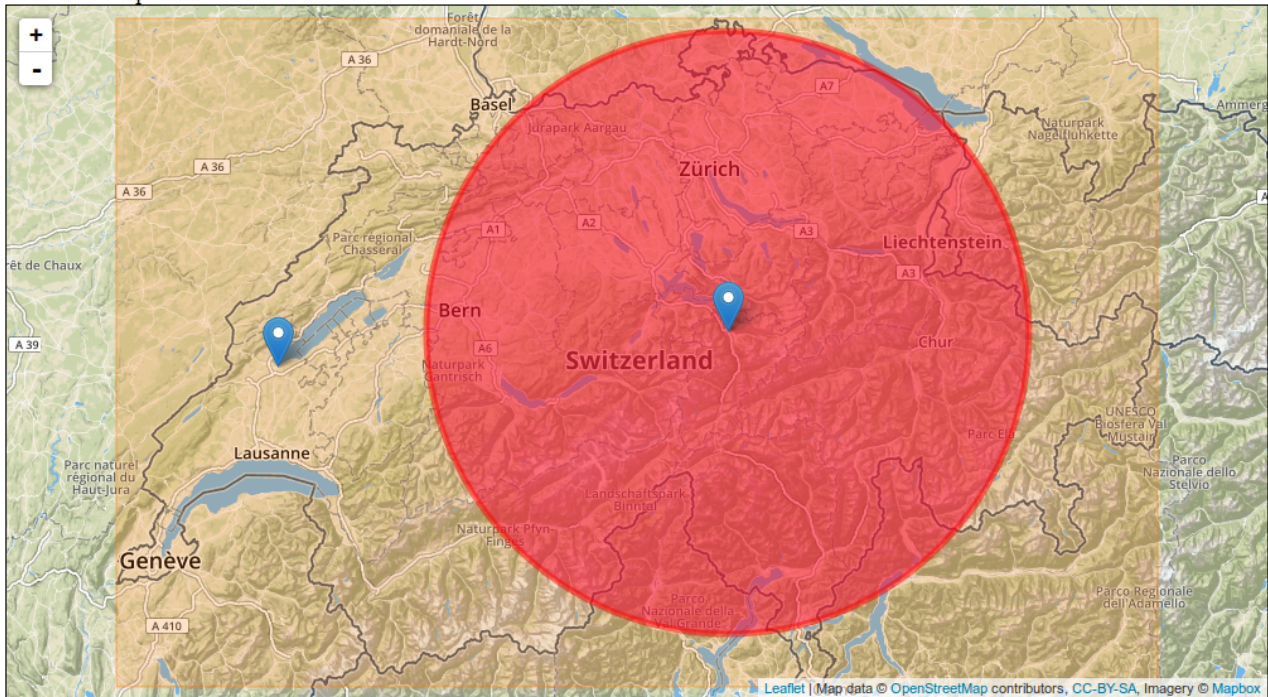
This application contains a simple map drawn with **Mapbox's** imagery and **OpenStreetMap's** data all together through the **Leaflet** JavaScript library. In this basic application, you can pin markers and draw rectangles and circles on a map. The "js/map.js" file contains all the map's code.

Just open "index.html" in your web browser and it will work.



## Welcome the the Leaflet test-application!

Here is a map of the Switzerland:



**Circle** (please click on a location on the map to automatically fill the circle's fields)

( 46.88222241007954 , 8.6187744140625 ), radius of 100 km

**Rectangle** (default: whole Switzerland)

Southwest coordinates: ( 45.817851 , 5.956085 )

Northeast coordinates: ( 47.808454 , 10.489254 )

**Marker** (default marker's location is the HEIG-VD in Yverdon-Les-Bains)

( 46.778678 , 6.658285 )

Illustration 19: "Leaflet" prototype application

In this application, the Leaflet library uses both OpenStreetMap for the map's data and Mapbox for the map's imagery. The Leaflet library acts like a wrapper on the OpenStreetMap's API. Mapbox (<https://www.mapbox.com/>) is a platform, which provides map's design as blocks to easily integrate location into web applications. Notice that this platform has a limited requests rate, which I noticed too late... In GeoTwit, I will directly use the OpenStreetMap imagery.

Before interacting with all the Leaflet's components, I had to create an account on the Mapbox platform (<https://www.mapbox.com/studio/signup/>) and then I had to:

1. Get the default API's public token (<https://www.mapbox.com/studio/account/tokens/>) and write it somewhere.

2. Create a MapBox project (<https://www.mapbox.com/studio/classic/projects>), which is used by the Leaflet library to provides imagery. Once done I had to get the project's ID and write it somewhere.



*Illustration 20: a MapBox project*

3. Finally, I created this simple web-application and extracted the downloaded Leaflet library into it, then followed the quick-start tutorial (<http://leafletjs.com/examples/quick-start.html>), which is well explained. In order to call the map's layer with MapBox; my code looks like this:

```
L.tileLayer('https://api.tiles.mapbox.com/v4/{id}/{z}/{x}/{y}.png?access_token={accessToken}', {
  attribution: 'Map data &copy; <a href="http://openstreetmap.org">OpenStreetMap</a> contribut
  maxZoom: 18,
  id: 'edri.pjdcfni6',
  accessToken: '[MY-PUBLIC-TOKEN-KEY]'
}).addTo(mymap);
```

*Illustration 21: MapBox integration into Leaflet*

### 3.3.2. Leaflet – Countries' Borders

This application provides the same functionalities as the **Leaflet** one, with the possibility to double-click on a country to draw a polygon all around it. In order to make the application work, please do the following:

1. Ensure JavaScript's packages manager "npm" (<https://www.npmjs.com/>) is installed on your computer.
2. Run "npm install -g browserify" to install "browserify", which is a library used to use npm's packages within a application without having Node.js.
3. "npm install" to install npm's packages.
4. Open the "index.html" file in your web browser.



5. Double-Click on any country on the map in order to draw a polygon all around it. Notice that some small countries are not present in the dataset so you won't be able to draw a polygon around them.

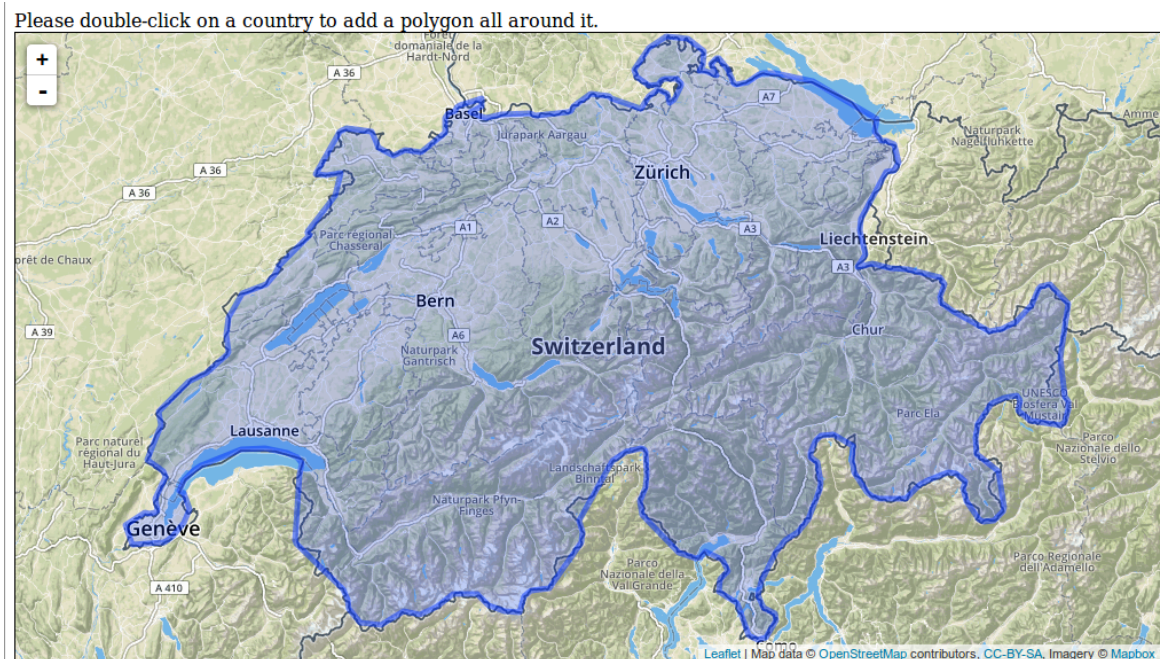


Illustration 22: "LeafletCountriesBorders" prototype application

All implementation's details are well commented in the code, but here are the main things to know:

- The data of all the countries' borders are contained in the "/data" folder as the binary Shapefile format (".shp" – <https://en.wikipedia.org/wiki/Shapefile>).
- Since the content of this file is a binary content, we cannot read it directly with JavaScript and thus have to parse it and convert it to JSON (specifically to GeoJson, which is an agreed format used for encoding a variety of geographic data structures in JSON; more information here: <http://geojson.org/>). In order to do this, I used the calvinmetcalf's shapefile-js library (<https://github.com/calvinmetcalf/shapefile-js>), which I randomly found with Google and which perfectly works by writing `shp( 'data/TM_WORLD_BORDERS-0.3' ).then(function(geojson) { ... })`.
- I also used two npm's packages to respectively get a country's ISO (e.g. CH, FR, ...) by a longitude and latitude pair and to get a country's name by its ISO (e.g. CH => Switzerland): which-country (<https://www.npmjs.com/package/which-country>) and world-countries (<https://www.npmjs.com/package/world-countries>). Since these packages are npm packages (which must normally be used in a Node.js server), I used the wonderful "browserify"

(<https://www.npmjs.com/package/browserify>) package, which allows us to use npm's packages within a simple JavaScript application. As soon as you updated the "js/map.js" file (which manages the double-click event on the map), you have to type "browserify map.js -o bundle.js" in a console to generate the "js/bundle.js" file that contains a working version of the map.js with all the npm's packages.

- When the user double-clicks on a map, we first get the clicked point's coordinates, then search for the clicked country's ISO and name with the coordinates, and finally get the borders' coordinates in the GeoJson data by the country's name in order to draw the polygon with Leaflet.

### 3.3.3. Twitter4JDesktop

This is a simple Java application in which you can search for Tweets or subscribe to the Twitter's Streaming API (by default) or ask the REST API.

In order to use this application please do the following: uncomment the country in which you want to search Tweets in the "main" of the "Twitter4j.java", then compile the application, launch "Twitter4j.java" and follow the output instructions.

---

```
run:
Trying to get the persisted token...
Persisted token successfully got!
Please enter some tags to search: job
How long do you want the streaming to run (in seconds)? 10
[Sun Jun 12 17:09:52 CEST 2016]Establishing connection.
[Sun Jun 12 17:09:53 CEST 2016]Connection established.
[Sun Jun 12 17:09:53 CEST 2016]Receiving status stream.
I received 79 Tweets in 10 seconds, including 14 ones WITH geolocation tags and 13 ones with the wanted geolocation.
This means 17.72 % of the received Tweets with the "job" tag(s) owned a geolocation tag and 16.46 % contained the desired location.
```

*Illustration 23: "Twitter4JDesktop" prototype application*

Notice that you may have to provide a PIN for the application if this is the first time you opened it on your computer (see the above "Twitter4J" chapter).

I used all the Twitter4J documentation in order to develop this application and everything is well-commented so I won't put other explanations here.

### 3.3.4. Leaflet and Twitter4J

This application receives streams of Tweets and display them on a map.

It contains two applications: **twitter4JWeb**, which is a Java application containing the **twitter4jDesktop** code and a web sockets server used to communicate with the second application **leafletAndTwitter**. This one is a JavaScript application displaying the received Tweets from the web socket server in a map. The well-named "js/websocket.js" file contains the web socket client part.

In order to correctly use them, please do the following:

1. Uncomment the country in which you want to search Tweets in the beginning of the "readStreaming" method in the "Streaming.java" file of the "twitter4jWeb"

application.

2. Write the keyword you want to search at the end of the “initializeConfiguration” method in the “Streaming.java” file. Try with the “job” keyword in the U.S.A. to get a lot of results.
3. Check that NetBeans and the GlassFish server are properly configured on your computer.
4. Run the Java server “twitter4jWeb”.

```

twitter4jWeb (run) x  Java DB Database Process x  GlassFish Server 4.1 x
Infos: Created virtual server glassfish-admin
Infos: Setting JAAS app name glassfish-web
Infos: Virtual server server loaded default web module
Infos: visiting unvisited references
Infos: visiting unvisited references
Infos: visiting unvisited references
Infos: visiting unvisited references
Infos: Registering WebSocket filter for url pattern /*
Infos: Loading application [twitter4jWeb] at [/twitter4jWeb]
Infos: twitter4jWeb was successfully deployed in 1'990 milliseconds.

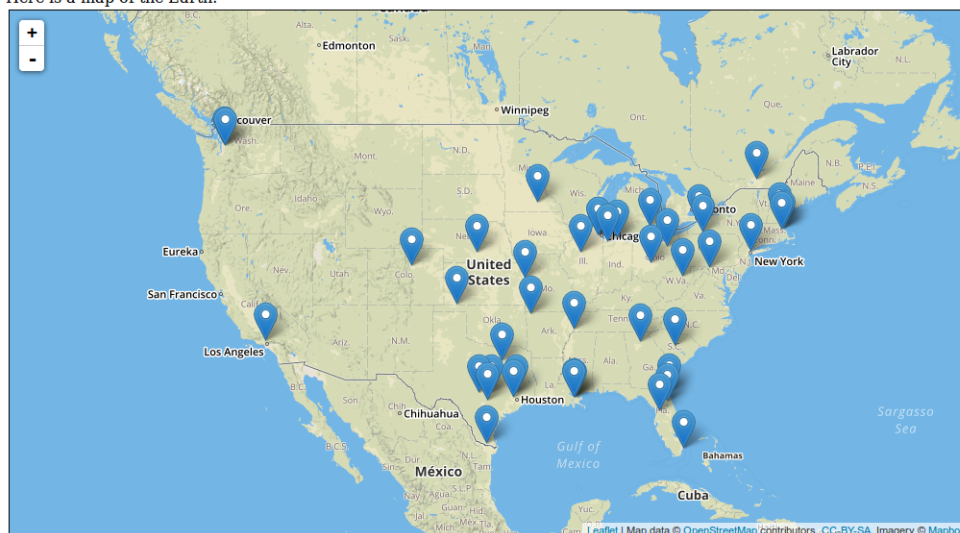
```

*Illustration 24: deployment of the GlassFish server in the “twitter4JWeb” prototype application*

5. Open the “leafletAndTwitter/index.html” file in your web browser when the server is correctly started.
6. Then just click the “Start Streaming” button in the web application to start the streaming.

## Welcome the the Leaflet test-application!

Here is a map of the Earth:



### Tweets Details:

[17:32:36] Denver, CO Jobs : This #job might be a great fit for you: Underwriting Assistant - <https://t.co/OwVlgOuK0A> #Denve:  
 [17:32:36] Illinois Acct Jobs : We're #hiring! Read about our latest #job opening here: Assistant Treasurer - <https://t.co/MqTIY>.  
 [17:32:36] TMJ-TX HRTA Jobs : #Fredericksburg, TX #Hospitality #Job: Assistant Manager at SONIC Drive-In <https://t.co/nPG>  
 [17:32:36] TMJ-CA HRTA Jobs : Interested in a #Hospitality #job near #Pacoima, CA? This could be a great fit: <https://t.co/vaj>  
 [17:32:35] TMJ-CHI Labor Jobs : Can you recommend anyone for this #Labor #job? <https://t.co/V2dc52OiAP> #Elmhurst, IL #1

*Illustration 25: "LeafletAndTwitter" prototype application*

Like before, I used all the Twitter4J documentation in order to develop this application and everything is well-commented so I won't put other explanations here.

### 3.3.5. Discussion

These applications allowed me to properly test all libraries I will need within the project, especially the Leaflet and the Twitter4J libraries. By developed them, I already did a huge part of the project's work and they also allow me to better understand and document these libraries.

Notice that I'm particularly pretty glad about the “**Leaflet – Countries' Borders**” application. I maybe will put it on a project apart on GitHub, because some people would need to use it.

## 4. The GeoTwit Application

This chapter contains all documentation related to the **GeoTwit** application's code.

### 4.1. Mock-Up

In order to make the mock-up more comprehensible, I spitted it in various parts. You can find the whole mock-up schema in annex.

When the user first accesses the web site, he is redirected on the home page, in which he can connect with his Twitter account.

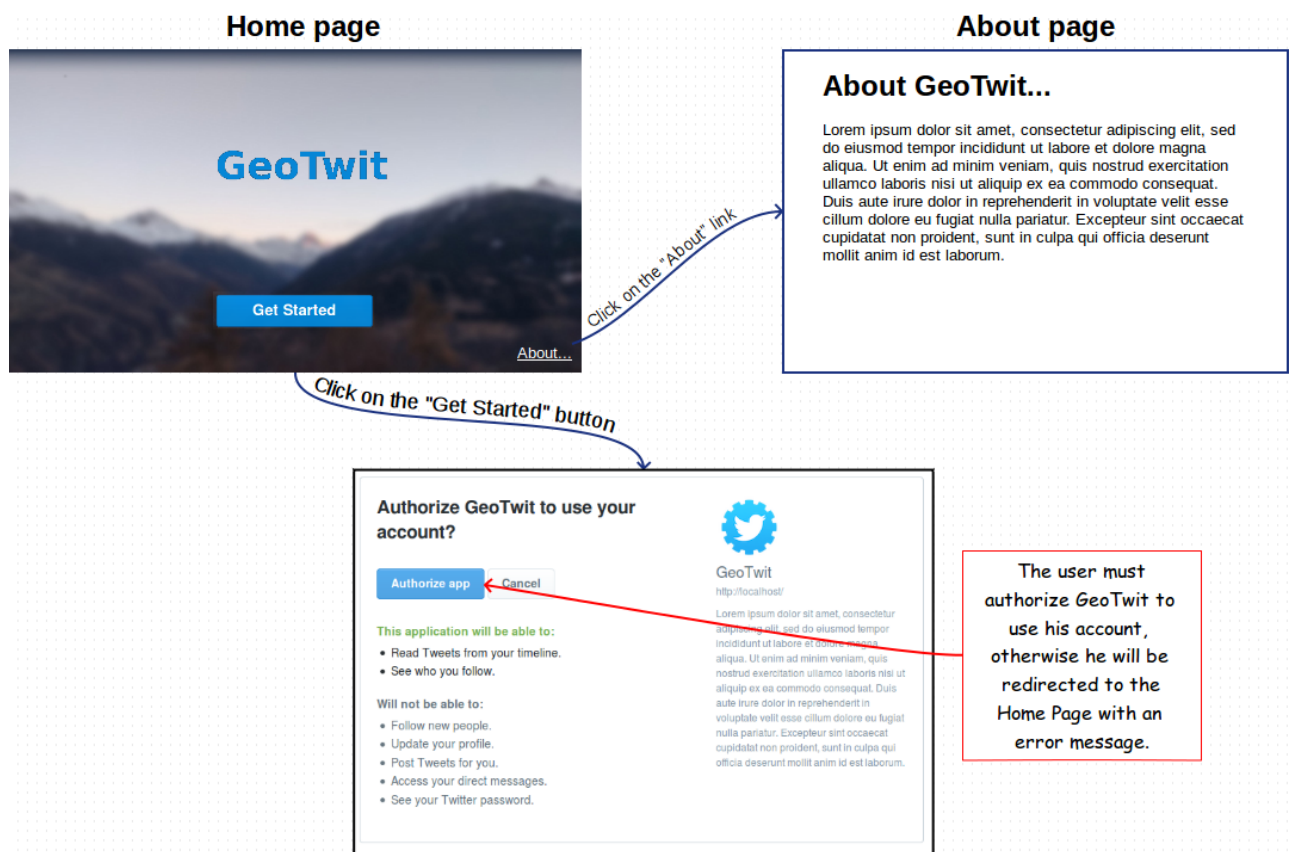


Illustration 26: connection process in the mock-up

Once the connection is established, the user accesses the search page and can search for Tweets either in dynamic or static mode. First have a look at the **dynamic mode**, which will use the Twitter's Streaming API.



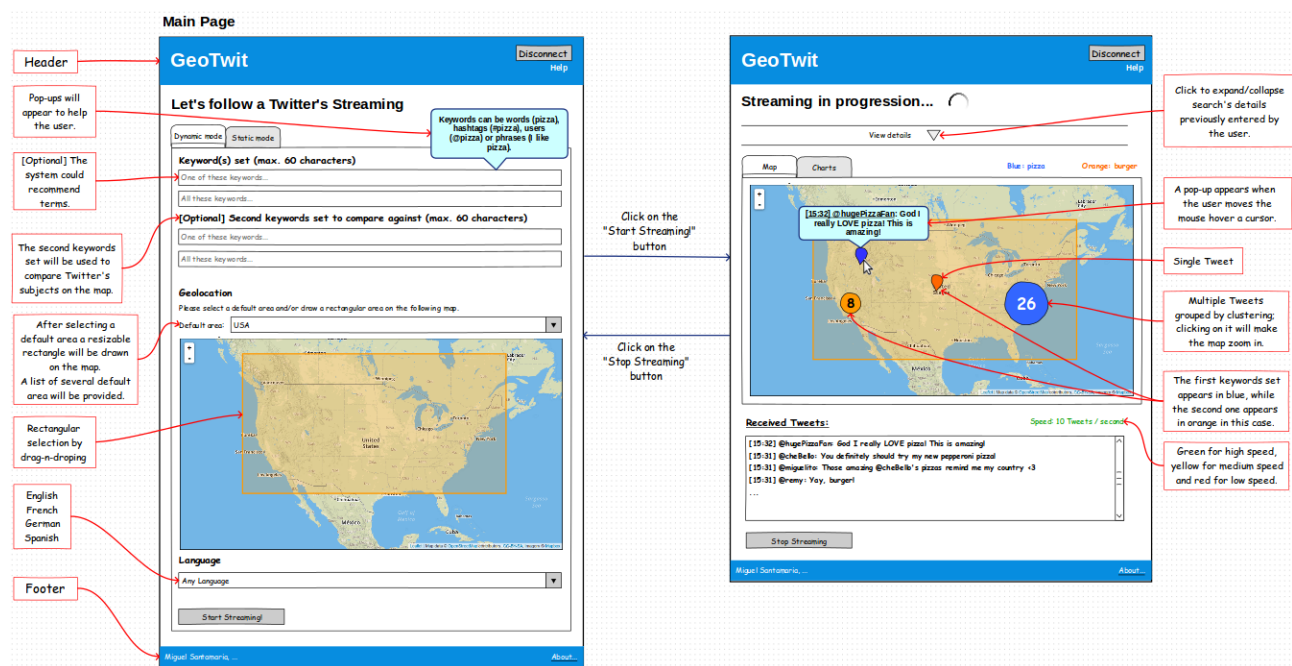


Illustration 27: main search page (dynamic mode) in the mock-up

When watching the results, the user can also access the “Charts” tab, in which he will found interesting charts about the current streaming.

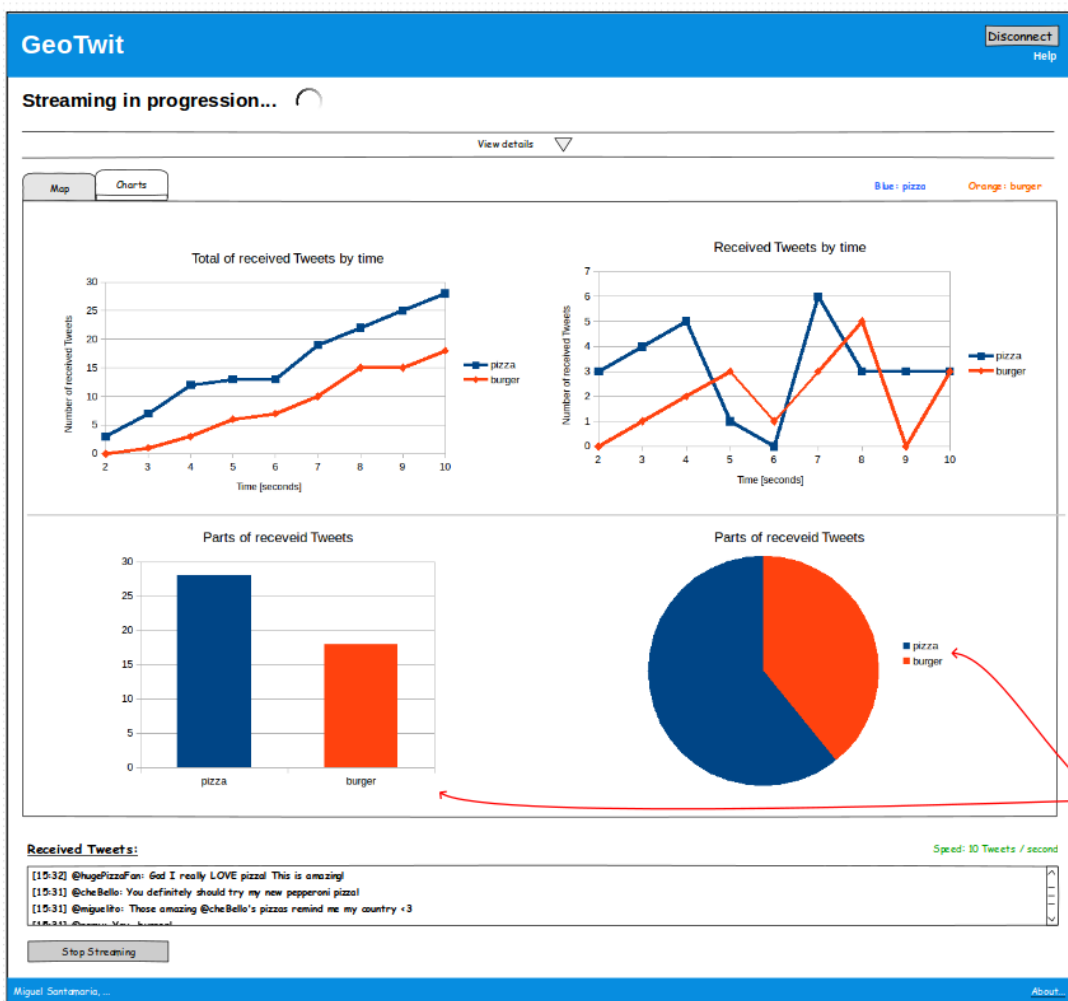


Illustration 28: charts of a streaming, in the mock-up

And now let's have a look at the **static mode**, which will use the Twitter's REST API.

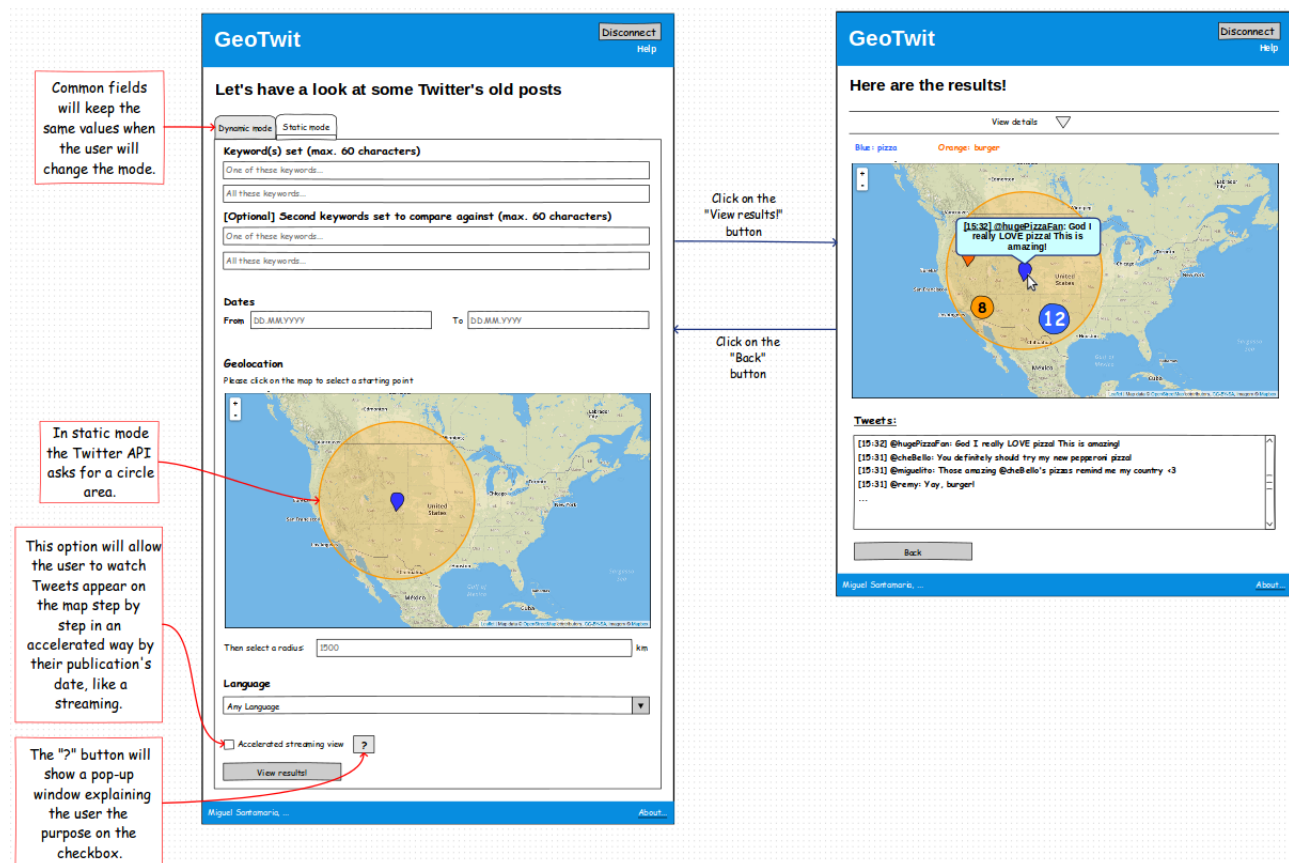


Illustration 29: main search page (static mode) in the mock-up

In the static mode, you cannot access charts.

## 4.2. UML Diagram of the Application

## 4.3. Implementation details

### 4.3.1. Behavior of the Views

All action's views inherit from one of these two interface templates:

1. "mainIndex.scala.html" if the current page is the index one. It contains only the header and footer partials and calls the page's content.
2. "main.scala.html" for all other pages; in addition to also containing the page's header and footer, it also contains a page format, which allows us to have the same page format (page's header, content, page's footer, etc.) in all pages.

As said above there is a header (header.scala.html) and a footer (footer.scala.html) partial templates, which are loaded in all templates. They respectively contain all HTML headers



(page's title, CSS and libraries loading, ...) and footers (closing tags).

### 4.3.2. Twitter4J

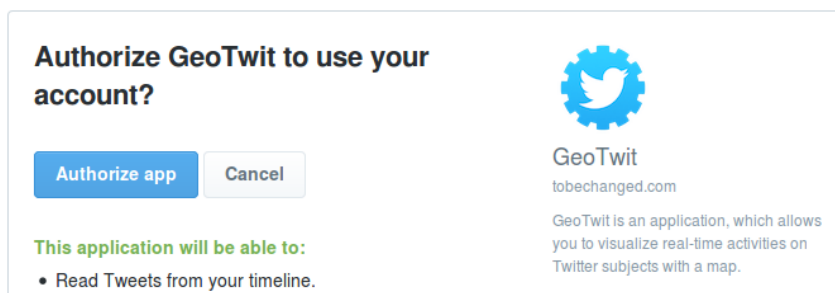
Twitter4J is used in the “HomeController” controller and is automatically loaded by SBT with the “build.sbt” file. As said before, this file indeed contains the Twitter4J's library dependency: `"org.twitter4j" % "twitter4j-core" % "4.0.4"`, where the first string is the library's location, the second one is the library's name and the last one is the library's version.

### 4.3.3. Connection Process

In order to properly connect to the Twitter's APIs the user must click on the “Get Started” button located on the index page.

When he clicks on it the following steps are operated:

1. The click calls the “auth” method of the “Home” controller.
2. This action uses the Twitter4J library to make an authentication's request in order to get the OAuth token and the authentication URL, which points to the Twitter's authorization page. When doing the request we also give the API a callback URL (/callback => action “callback” of the “Home” controller), which will be called at the end of the authentication.
3. Redirection of the user to the Twitter's authorization page.



*Illustration 30: Twitter's authorization page for the GeoTwit application*

4. Whatever the user does, he will be redirected on the callback action; if the user successfully authorized the application, the callback will memorize the OAuth token and redirect the user to the “search” action, otherwise he will be redirected to the index page with an error.

## 5. Conclusions

### 5.1. Abstract

(15ème de lignes)

Context (très court), Objectives, Approaches, Conclusions (ce que j'ai obtenu)

### 5.2. Review of the Current State

### 5.3. Encountered Problems

### 5.4. Final Discussions

(qu'est-ce que je vais faire, etc.)

## 6. General References

- <http://mashable.com/2010/08/06/twitter-mapping-tools/#yrksT7cG8Sgg>: existing Twitter mapping tools
- Wikipedia: description about Scala and the Play Framework, explanations about the “Chunked transfer encoding” data transfer mechanism and the Shapefile file format.
- <https://www.playframework.com/>: the Play Framework
- <https://dev.twitter.com/overview/documentation>: documentation of the Twitter's APIs, also used for illustrations
- <https://www.google.com/maps>: used for illustrations
- <http://www.rts.ch/sport/football/euro/2016/resultats/>: results of the UEFA Euro 2016's matches
- <http://twitter4j.org/en/index.html>: Twitter4J Java library
- <https://developers.google.com/maps/?hl=fr>: Google Maps' APIs
- [http://wiki.openstreetmap.org/wiki/API\\_v0.6](http://wiki.openstreetmap.org/wiki/API_v0.6): OpenStreetMap's API
- <http://leafletjs.com/index.html>: Leaflet
- <https://github.com/Leaflet/Leaflet.markercluster>: Leaflet's marker-cluster algorithm,

used to group data by circles on the map

- <https://www.npmjs.com/>: JavaScript's package manager “npm”
- <http://geojson.org/>: GeoJson information
- <https://github.com/calvinmetcalf/shapefile-js>: Shapefile to GeoJson conversion library.
- <https://www.mapbox.com/>: a platform used to get map's imagery within Leaflet

Planning (EN ANNEXE)

Initial Planning

Logbook

Hours