heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Bachelor Thesis

GeoTwit – Progress Report

Miguel Santamaria

Table of contents

Table of illustrations

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 am 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. Generalities

Peers	Student / Developer: Miguel Santamaria Professor / Supervisor: Fatemi Nastaran
Name of the project	GeoTwit
Short description	GeoTwit is an application, which allows you to visualize real-time activities on Twitter's subjects with a map.
GitHub repository	https://github.com/edri/GeoTwit
Start date	22 February 2016
Progress report	17 June 2016
End date	29 July 2016
Actual duration	450 hours
Oral defense	Between 5 and 16 September 2016

1.2. 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 this 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.

1.3. 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 and resolving a problem, and by developing the solution.

In case of problems and to ensure a successful work, the student is supported by a supervisor – generally a professor from the HEIG-VD – who gives him advice and checks 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 final oral defense.

1.4. Abstract

GeoTwit is the application linked to my current Bachelor thesis, which allows an user to visualize real-time activities on Twitter's subjects with a map. As I am writing this abstract, it is the 16th week of the semester. I am thus in the middle of my work and have to produce a progress report about what I currently achieved to prove I properly worked until now. My objectives at the beginning of the semester were to complete all the specifications and the analysis parts of the project, and even to start developing an early of the application until this progress report. Because we had a lot of work during this semester (way more than expected, in fact), I had to use several approaches to reach my goals: firstly I always worked on this project during Mondays and often during Thursdays, and I also worked in the transports since I spend a lot of time in them. I began with both the specifications and the analysis parts, in particular with the analysis of the Twitter's APIs. To avoid losing a lot of time at the end of the semester, I also mostly kept the documentation up-to-date. I am glad to announce I reached my goals until now: all the specifications and analysis parts are completed, as well as some pages of the application (the home page, the connection process and a basic GUI for the search pages) and various prototypes applications, containing a lot of code I will reuse. For the second half of this project, I am firstly planning to finish the application and the documentation of course, but also to mostly focus on the data analysis since I think I am a little bit early. I will also develop the static mode of the application, which was noted as an application's extension.

2. Specifications

2.1. Description of the project

The goal of this project – GeoTwit – is to set up a web application allowing users to enter two subjects of their choice and to visualize real-time activities for these subjects both on Twitter and on a geographic map. The application could 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. Functionalities

The application will provide the following features:

- 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) and libraries, in order to allow an appropriated visualization by the selected zoom level.
- The generation of data charts during the results phase.

2.2.1. Tasks

Here is a chronological list of the various achievements I will operate during this project:

- Use of the Twitter's APIs, including:
  - Retrieval and filtering of Tweets by one or more keyword(s).
  - Retrieval of geographic information attached to a Tweet.
  - Analysis of the percentage of Tweets having 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.
- Development of charts in order to display interesting information about the results.
- Tests and validations.

2.2.2. Used Technologies

It was suggested to use the Scala language coupled with 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 Play Framework before starting developing the application.

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, and which could even simulate a streaming by analyzing the Tweet's publication dates.**

   After some discussion with my supervisor, we decided that these alternatives will be implemented.
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.).

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 mostly 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 around the world on a map.
4. **TweetMap** (https://worldmap.harvard.edu/tweetmap/): with this tool you can visualize Tweets that was 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 write these lines, it only supports static Twitter content, so I won't use it.

This project is thus not totally innovative, but still has some interesting features, like the possibility to collect Tweets in both dynamic and static ways and the possibility to access charts and perhaps other data analysis tools.

3.2. Technologies

This chapter contains analysis of the main used technologies within this 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 the logical continuation, 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.
In the following paragraphs, you can find basic instructions about the framework's use.

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 started 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 could be helpful during the conception of the application, both to the security and the optimization: https://dev.twitter.com/overview/general.

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 tools used to easily interact in a stable way with the social features of Twitter (sharing, connection, etc.).
- Twitter for Websites: widgets collection, used to easily integrates Twitter's widgets (“Post Tweet” and “Follow” buttons, display of Tweets, etc.) in a web site.
- **OAuth**: an authentication way used with the Twitter APIs 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 in order to use them in development. They work like most of the APIs provided by services of this kind and are analyzed below.
- **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 subject and thus to automatically show new Tweets matching with the filter. This technology is analyzed below as well.

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 me to use it.

3.2.3.b. OAuth

The current version (v1.1) of the Twitter APIs 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 the application and the end-user it makes API calls on behalf of.
   In terms of being specific, the user using the application must have a Twitter account, have to connect and then 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.

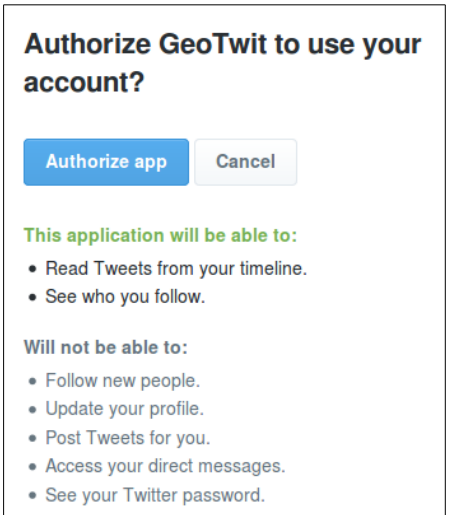


Illustration 1: example of Twitter application's authorization

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.

2. **Application authentication only**: the application executes the requests of the APIs 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 APIs. It is indeed possible to search for Tweets and get users information, but one cannot use either geographic data or streaming components...

Despite the ergonomic issues the first method (application-user authentication) could create, I will still keep it. Indeed, the second method does not allow me to use either geolocation or streaming features, which are 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-formatted responses. It allows one – 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:

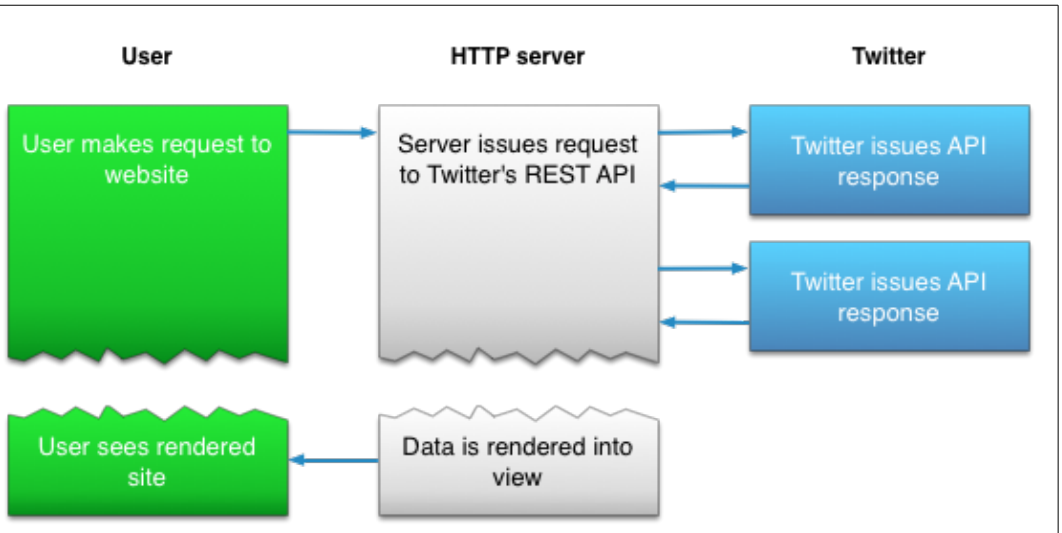


Illustration 2: use case of the REST API within a web application, taken from the documentation of the Twitter's APIs

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:

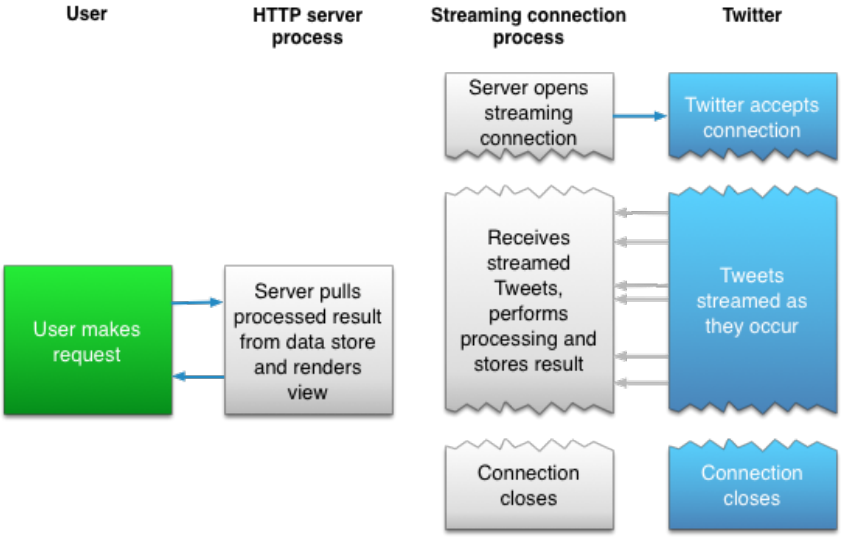


Illustration 3: use case of the Streaming API within a web application, also taken from the documentation of the Twitter's APIs

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 I will use (I will indee0d need to analyze all Twitter's public data).

In order to follow the specifications of the project, I will mainly use the Streaming API, in the sens that it has been asked to be able to visualize the activities of the Twitter's subjects in real-time. I 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 Tweets within GeoTwit.

Streaming API

To initialize a stream based on a filter, the developer just has to execute a specific HTTP request (detailed above), indicating the Twitter's servers he wants to connect on. He then needs to maintain the connection opened so the servers will send him data at regular intervals. To know one is still connected, the Twitter's servers send each 30 seconds a message to the application, indicating that the connection is still active (in case there would be no new actualities). If no confirmation is received after 90 seconds the developer needs 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 introduced and explained 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.

In those prototype application, I noticed something important when I did my tests: Twitter seems to restrict the number of Tweets transiting through the streaming, which means that the Streaming API does not send all posted Tweets! When I indeed tried to post some Tweets when listening to a streaming (in particular during the Euro2016 event), I did not systematically received them through the streaming (even if they had geolocation tags, etc.). Thus, the developer only receives samples (which still seems to represent a view of reality) when listening to a streaming.

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 the IP address of the client for a while, so I 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 the developer wants 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 pair; 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 one wants to search Tweets in Switzerland, it has 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 EXPRÈS… 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 obviously coming from all around the word, which restrict the use cases within the application…

2. **Search with the location only** → I obtained 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 this 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☻ : @TylerDurfee11 @SexualGif ILL BE HOME SOON😩
```

Illustration 6: search of Tweets with location only

3. **Search with keyword AND location** → at first sight one 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 have a geolocation tag (I did some tests and obtained a rate of about 20-30% of Tweets having geographic data), so I 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 I need to search Tweets by a keyword AND a location in the application. Therefore, I had no other option but to code my own location filter, which is 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 geolocalion 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 geolocalion 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 geolocalion 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 geolocalion 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 I received about 3 to 4 times less Tweets than before, I still received more interesting Tweets. Thus, this solution will be the one I will keep.

Within GeoTwit, it could also be possible to propose default locations (for example by countries). I made a prototype application ("Leaflet Countries' Borders" – see the "Prototype Applications" chapter) in which you can double-click on countries to draw borders all around them. Many libraries/API exists and allows one 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 the developer 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 one's 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 sending data in a series of "chunks" without knowing the length of the content before starting transmitting a response to the receiver. The 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 windows. Notice that if one's 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 I 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 one 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 the given request.
- X-Rate-Limit-Remaining : indicates the number of remaining requests for the type of the 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 one 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 exist 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, the interesting ones are 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: one has to use a specific geocode in the “latitude,longitude,radius” format, where *radius* is the search radius in miles (*mi*) or 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 one wants to search Tweets in a radius of 50 kilometers around the main station of Yverdon-les-Bains, 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, the developer is redirected on a web page containing a URL, which is built on a particularly format. For example, with a search about the @twitterapi user, one will get the following URL of redirection: https://twitter.com/search?q=%40twitterapi. Once done, the developer 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 this example → https://api.twitter.com/1.1/search/tweets.json?q=%40twitterapi).

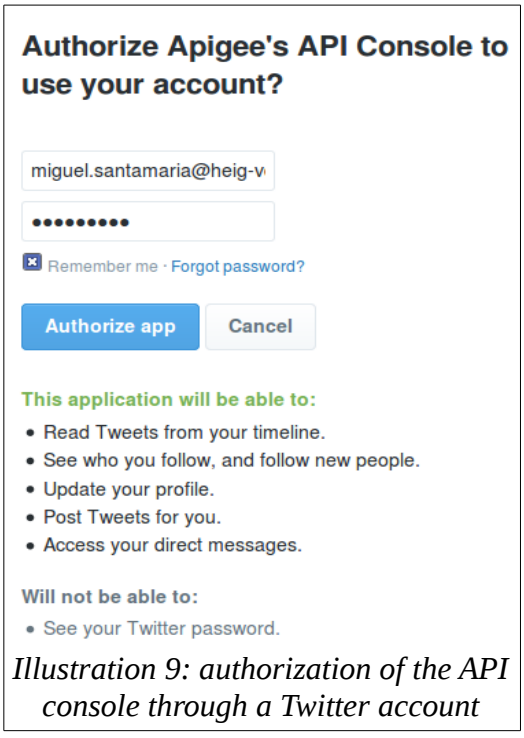


Illustration 9: authorization of the API console through a Twitter account

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, one has to authorize the console to access its own Twitter account. This access is automatically asked as soon as one tries to make the first request.

Here is a list containing a few examples, 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
    - Results:

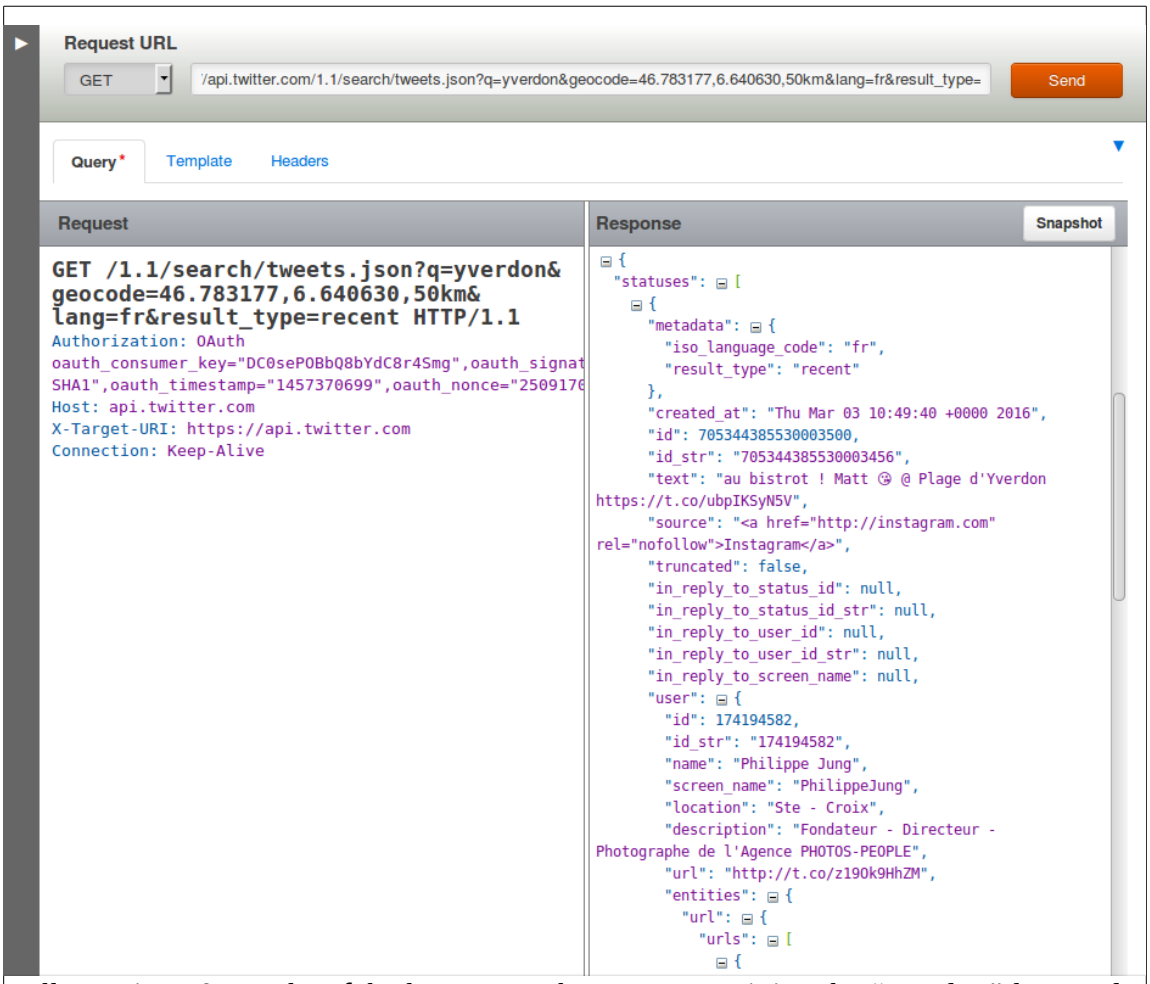


Illustration 10: results of the latest French Tweets containing the “yverdon” keyword and (OR) being located near Yverdon-Les-Bains in the API console

- Results on Twitter:

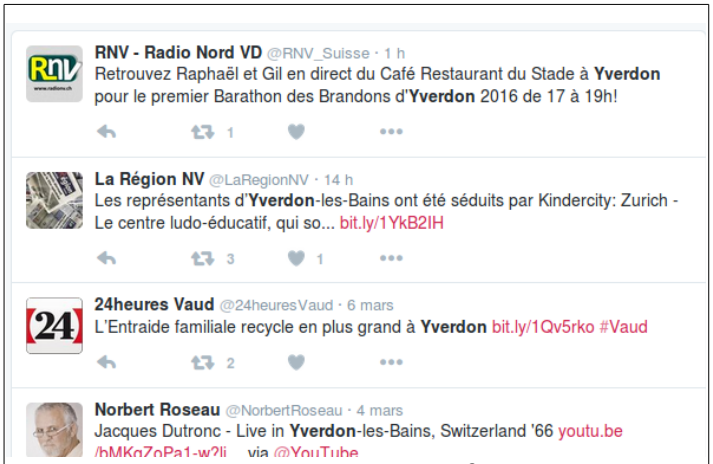


Illustration 11: results on Twitter of the latest French Tweets containing the "yverdon" keyword and (OR) being located near Yverdon-Les-Bains

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:

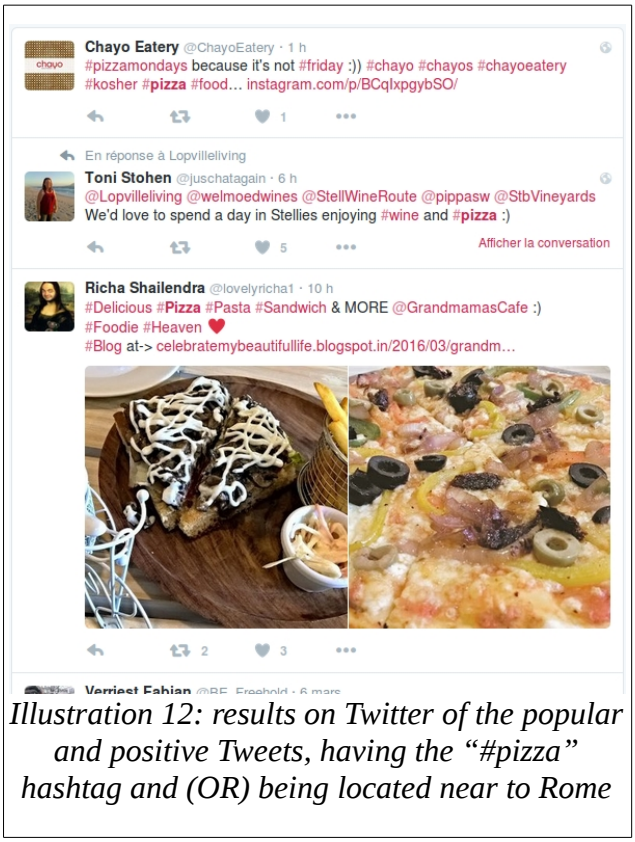


Illustration 12: results on Twitter of the popular and positive Tweets, having the "#pizza" hashtag and (OR) being located near to Rome

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 25th 2016 between 13:00 and 16:30 (Swiss hours) for the Europeans countries and during the next day between 09:00 and 10:00 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
london		**15**
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	U.S.A., ({-124.411668, 24.957884}, {-66.888435, 49.001895})	3
job		**618**
sport		0
people		4
vote		1
trump		3
clinton		0
logitech		0 or 1
nestlé		
novartis		
swatch		
mcdonald		0
java		0

scala		0

As regards the recurring themes related to Switzerland, I though 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 that 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 Europeans 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 in their respective countries.

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 the same results results again! I also tried more typical and current American subjects – like “vote”, “trump”, “clinton”, “mcdonald” – and computer subjects like “java” or “scala” but they were less interesting.

In conclusion, results are pretty bad. I 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 the zero rate of Tweets having geolocation tags.

After some consideration and discussion with my supervisor, we decided that it would be better to try these keywords as well as other ones during hours/days instead of minutes. Those tests will be done once the application will be developed, because it will be more convenient to do long search and I will have accesses to graphs as well.

In addition to these few examples, other interesting cases linked to particularly events are possible: for example, 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.

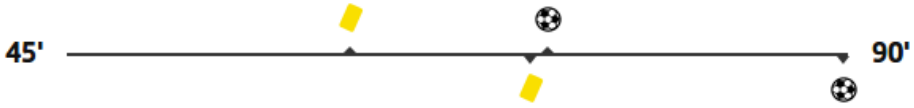
Finally, here is a little “planning” of hot and interesting coming topics until the end of the semester:

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	69th 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
01.07.2016 to 16.07.2016	50th Montreux Jazz Festival
02.07.2016 to 24.07.2016	“Tour de France” 2016
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
On **11.06.2016** From **22:00** to **23:00** During the second half of the **England-Russia match** Country: **UK** Keyword: **euro2016**	 *Illustration 13: tweets results for the UK-RU match* Score: **1-1** **47** Tweets with geolocation tags Average of **12'000-14'000** Tweets per 10 minutes until the first goal (73'), then **20'000-24'000** after it and until the end Between **0.01%** and **0.07%** of Tweets had a geoocation tag (between 2 and 20 Tweets per 10 minutes). I received a massive flow of 16'000 Tweets in a few minutes after the goal of the 92'.  *Illustration 14: progression of the EN-RU match, taken from the RTS web site*

On **15.06.2016**
From **17:55** to **19:55**

During the Switzerland-Romania match

Country: **Switzerland**
Keyword: **euro2016**

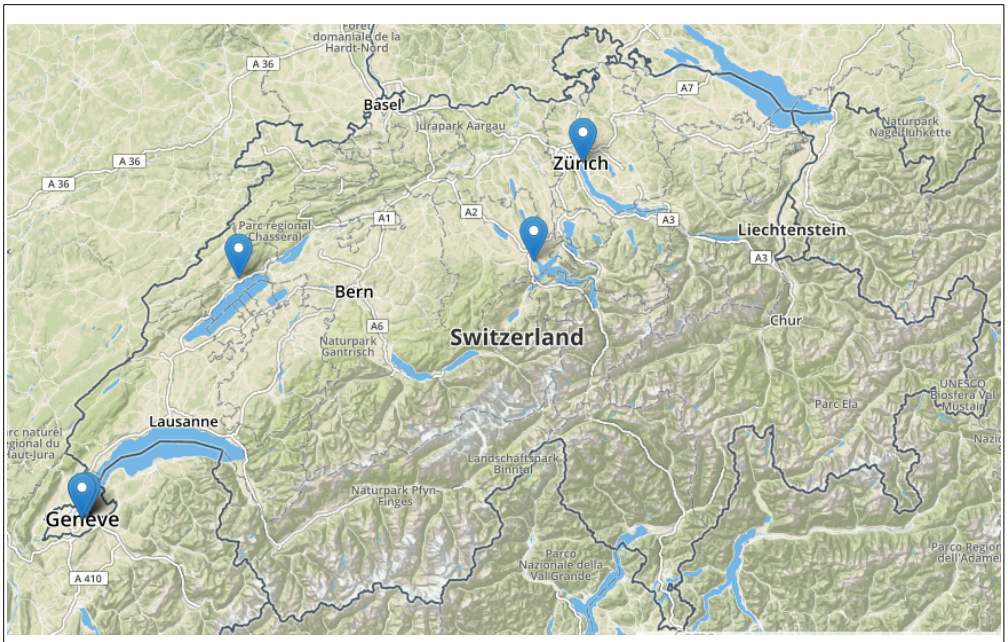


Illustration 15: tweets results for the CH-RO match

Score: **1-1**

5 Tweets with geolocation tags

Average of **6'500-8'000** Tweets per 10 minutes, and **10'000-11'000** at the end of the match

Between **0.01%** and **0.05%** of Tweets had a geoocation tag (between 1 and 4 Tweets per 10 minutes).

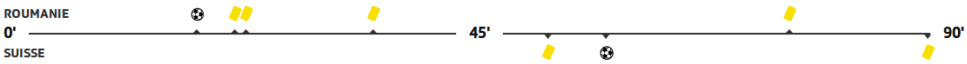


Illustration 16: progression of the CH-RO match, taken from the RTS web site

On **15.06.2016**
From **20:55** to **23:10**

During the France-Albania match

Country: **France**
Keyword: **euro2016**

Illustration 17: tweets results for the FR.AL match

Score: **2-0** for the France

126 Tweets with geolocation tags

Between **0.01%** and **0.09%** of Tweets had a geoocation tag (between 5 and 15 Tweets per 10 minutes).

Average of **15'000** / 10 minutes at the beginning of the match, **9'000-12'000** during it, and **24'000-28'000** at the end when the France scored twice.

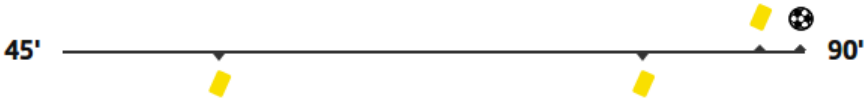


Illustration 18: progression of the FR-AL match, taken from the RTS web site (there was two goals at the end)

There was a lot of Tweets in Lille, Paris and Marseille. Since the match took place in the “Stade Vélodrome” of Marseille, many Tweets were posted there.

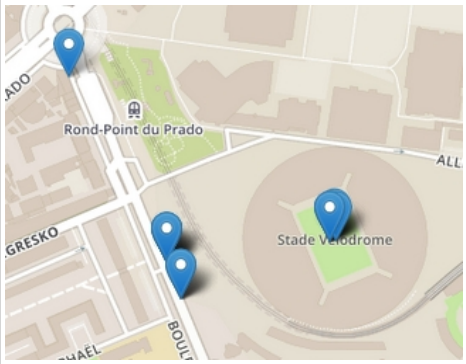


Illustration 19: tweets results for the FR-AL match in the "Stade Vélodrome" of Marseille

On **16.06.2016**
From **14:55** to **17:05**

During the England-Wales match

Country: **UK**
Keyword: **euro2016**

Illustration 20: tweets results for the EN-WLS match

Score: **2-1** for the England

135 Tweets with geolocation tags

Average of **18'000** Tweets / 10 minutes at the beginning of the match, then **10'000-14'000** until the end and finally **22'000-24'000** each time the England scored (56' and 91').

Between **0.02%** and **0.14%** of Tweets had a geoocation tag (between 3 and 20 Tweets per 10 minutes).

There was a lot of Tweets in London, Norwich and Manchester, but just a few in Wales.

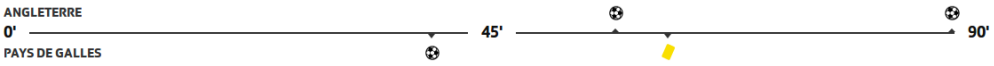


Illustration 21: progression of the EN-WLS match, taken from the RTS web site

On **16.06.2016**
From **20:55** to **22:55**

During the Germany-Poland match

Countries: **Germany** and **Poland**
Keyword: **euro2016**

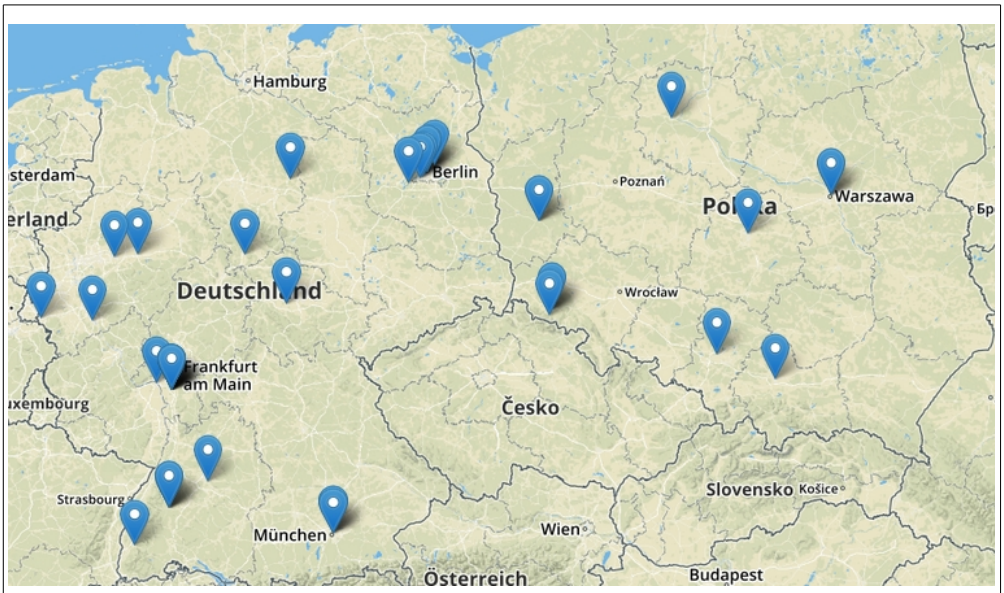


Illustration 22: tweets results for the DE-PL match

	Score: **0-0** **40** Tweets with geolocation tags Average of **18'000** Tweets / 10 minutes at the beginning of the match, then **11'000-14'000** until the end. Between **0.01%** and **0.08%** of Tweets had a geoocation tag (between 1 and 6 Tweets per 10 minutes, 15 in the beginning).  *Illustration 23: progression of the DE-PL match, taken from the RTS web site*

Discussion

After these first results, I can conclude that England and France are the countries whose people post the most Tweets numbers during the Euro2016 event, among all the analyzed countries. The France case can easily be explained: this is the country in which the event takes place, so many people are watching live matches and are twitting. England people just seem to like Twitter and thus post a lot; also, it is the country having the highest percentage level of Tweets having geolocation tags. Concerning Switzerland, there is still too few Tweets for me to be interesting: the application indeed received only 5 Tweets in a 2 hours period.

Tweets are generally grouped in big cities and in the stadiums in which the matches takes place. We can obviously see a certain correlation between the goals moments and the peak of Tweets. These analysis are very pleasant compared to the previous analysis I did in Europe, because of the higher Tweets rate I received. I am really looking forward to watch and analyze the Euro2016's final match.

3.2.4. Twitter4J

Twitter4J (http://twitter4j.org/en/index.html) is a non-official library, which – as its name implies – allows one to easily use the features of the Twitter's APIs with Java and which was proposed by my supervisor, Mrs. Fatemi. 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.

You can find useful basic instructions about the use of this library in the following paragraphs, allowing you to better understanding what I did in order to develop the prototype applications, as well as the GeoTwit application.

Installation

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 asynchronicity), add the other libraries too (like “lib/twitter4j-stream-4.0.4.jar” for the streaming).

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.

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 the parameters of your application.

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.

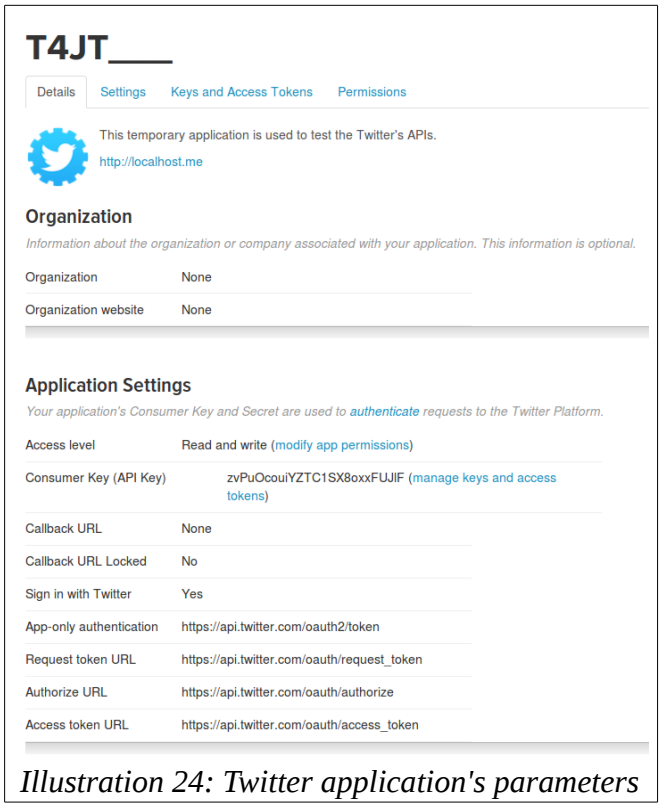


Illustration 24: Twitter application's parameters

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. You can find an example in the “GeoTwit/prototypes/twitter4jDesktop” folder.

For the mentioned reasons one has to use the PIN-based OAuth (the user has to give a PIN number generated by Twitter in order to use the API); 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).

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

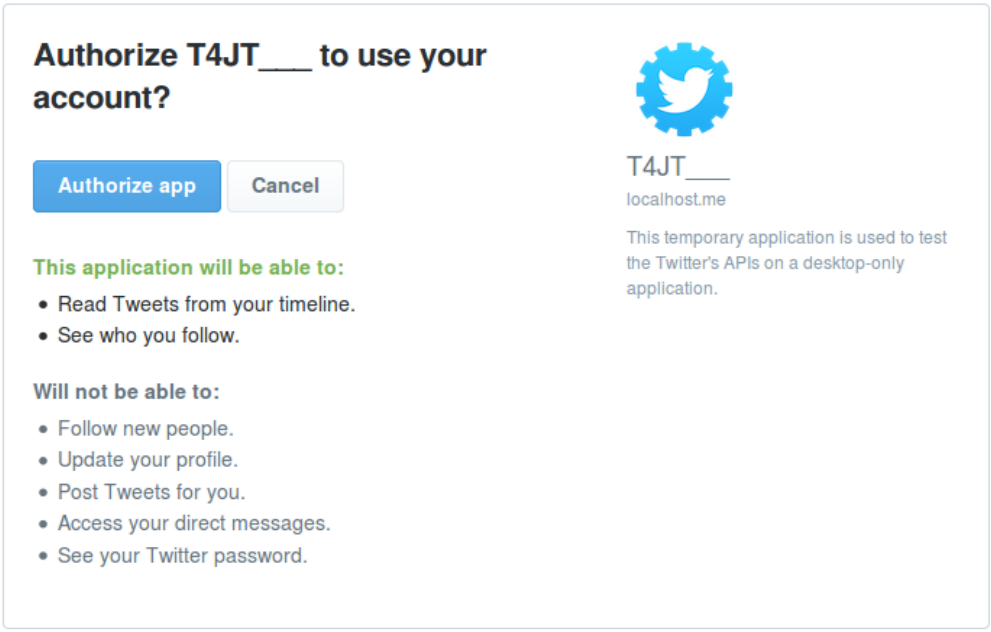


Illustration 25: Twitter application's authorization's page

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

Illustration 26: 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=v6inWQAAAAAAuPXSAAABU5no9GU
Enter the PIN(if aviailable) or just hit enter.[PIN]:6993695
Yay! Token successfully got: AccessToken{screenName='Migwelsh28', userId=533045969}
BUILD SUCCESSFUL (total time: 29 seconds)
```

Illustration 27: 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, I 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 I indeed want to use the site in a proper way, I 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).
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 these applications right here: 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 my case, the interest of this library – in addition to the main features offered by the map – especially lies in the fact that it allows one to easily communicate with the OpenStreetMap's API. One can also easily draws 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, I am going to deal with two major massive data issues :

1. When sending data from my server to my 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 short time, making my web browser crash… To avoid this case I 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 perhaps be thousands of Tweets to display on the map, I 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 one 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), 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 (in the “GeoTwit/prototypes” folder). Please notice that there is no inputs controls and the GUIs are very poor, because they are only 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.

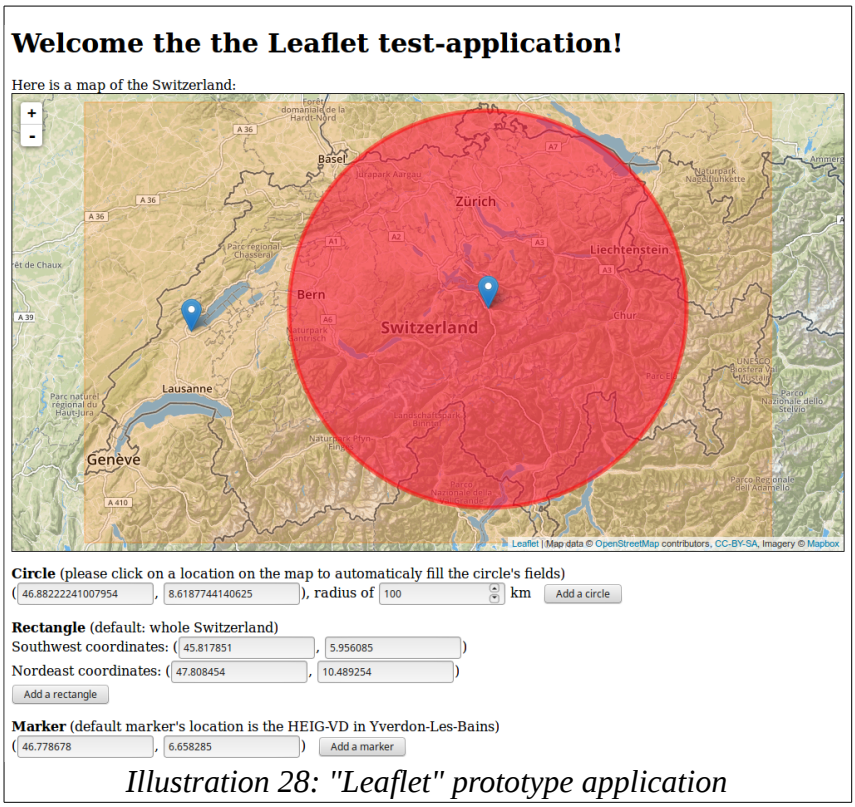


Illustration 28: "Leaflet" prototype application

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 29: 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, 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 30: 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 the JavaScript's packages manager “npm” (https://www.npmjs.com/) in 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 a Node.js server (https://www.npmjs.com/package/browserify).
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 data set so you won't be able to draw a polygon around them.

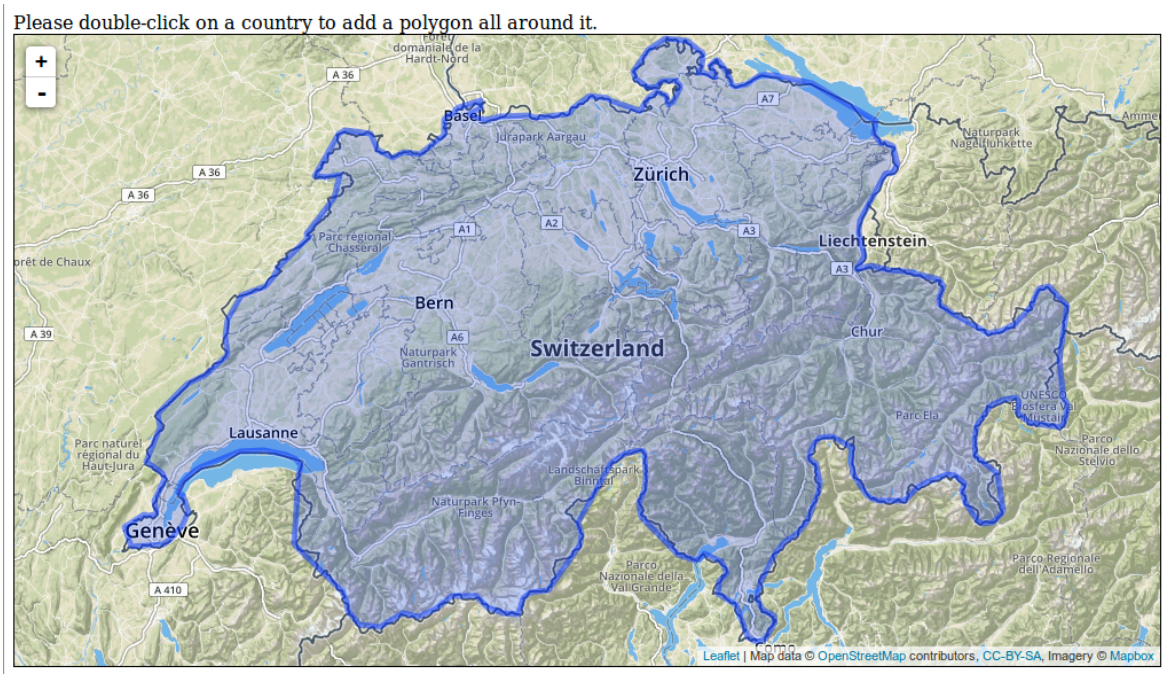


Illustration 31: "LeafletCountriesBorders" prototype application

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 in a binary format (Shapefile or ".shp" – https://en.wikipedia.org/wiki/Shapefile).
- Since the content of this file is a binary content, JavaScript cannot read it directly and thus has 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 (eg. CH or FR) by a longitude and latitude pair and to get a country's name by its ISO (eg. 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" npm package, which allows the developer 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, the application first gets 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 with the REST API or subscribe to the Twitter's Streaming API (by default).
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 geolocalion 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 32: "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 application 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.

Is 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. You can as well change the duration of the analysis below.
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” in order to deploy and run the GlassFish server.

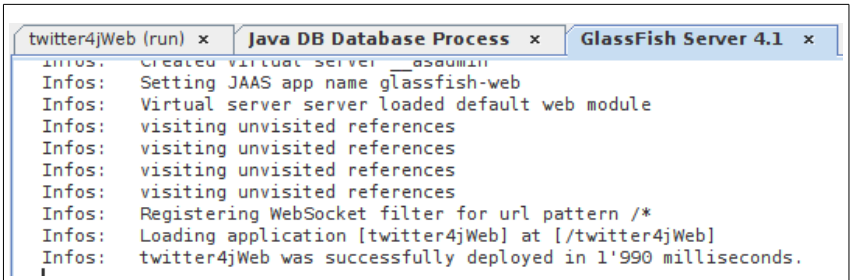


Illustration 33: deployment of the GlassFish server in the “twitter4JWeb” prototype application

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.

Illustration 34: "LeafletAndTwitter" prototype application, using the “job” keyword in the U.S.A.

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 am 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.

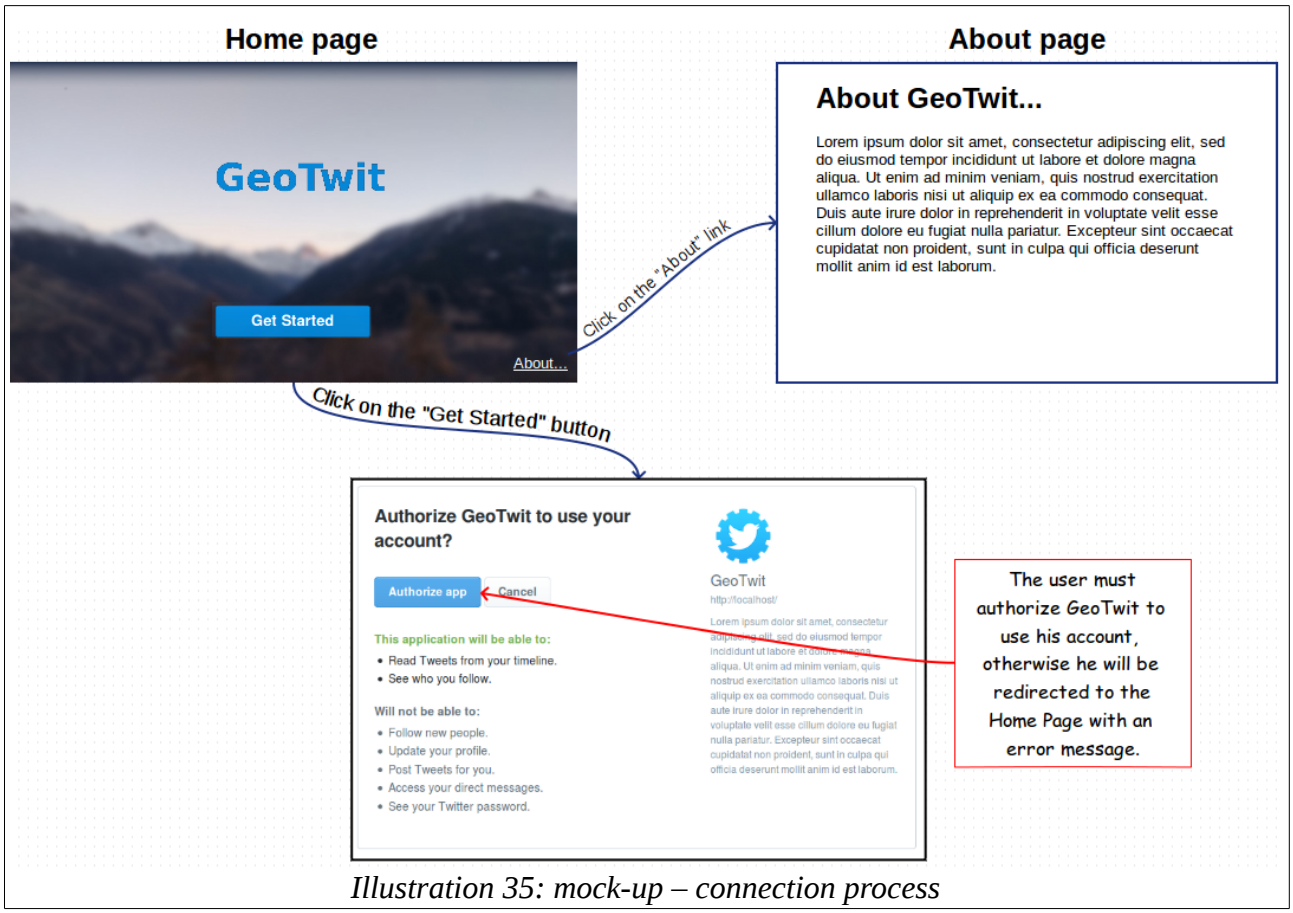


Illustration 35: mock-up – connection process

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.

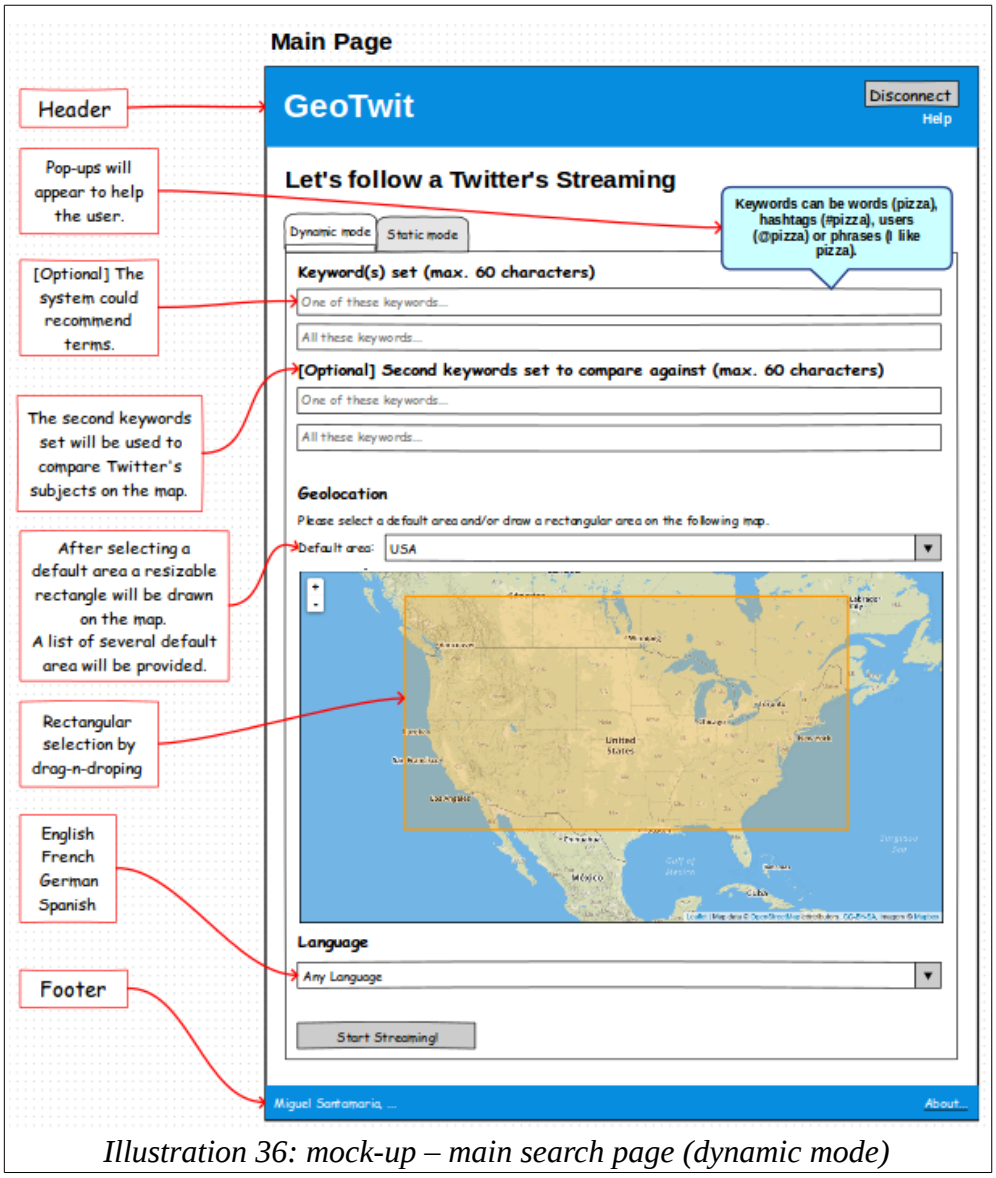


Illustration 36: mock-up – main search page (dynamic mode)

The user has to fill the fields, then click on the “Start Streaming!” button to begin the streaming process.

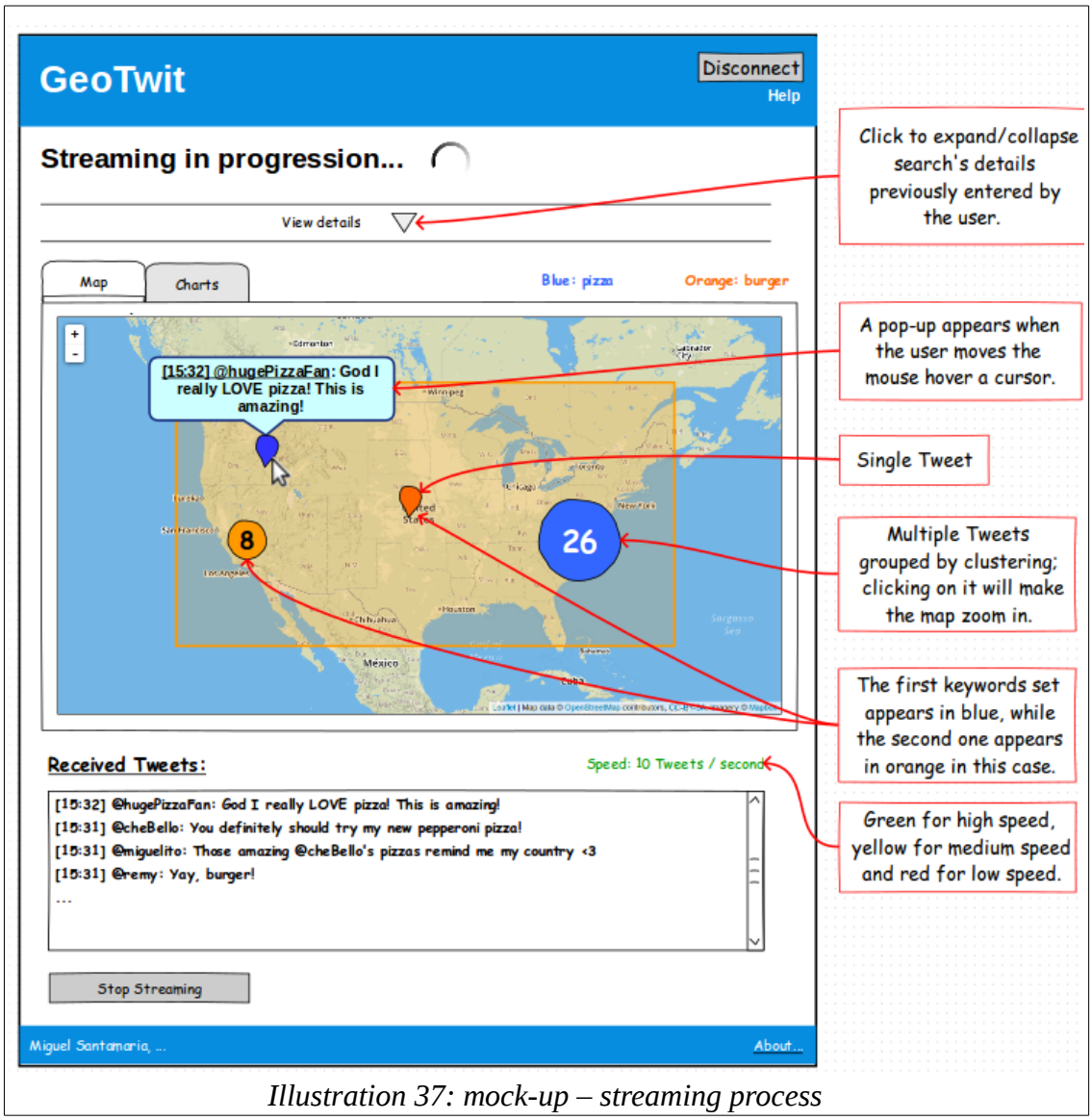


Illustration 37: mock-up – streaming process

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

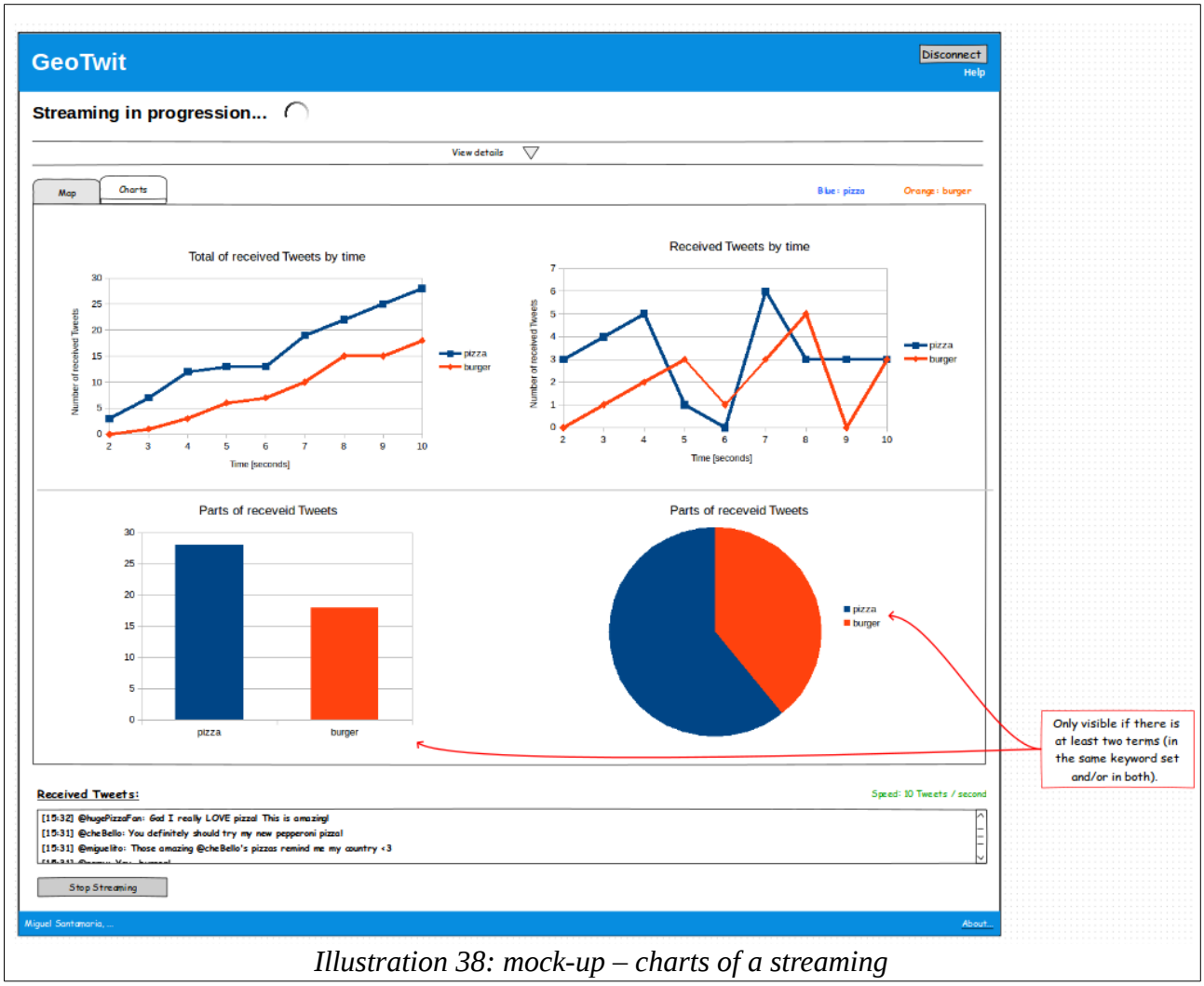


Illustration 38: mock-up – charts of a streaming

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

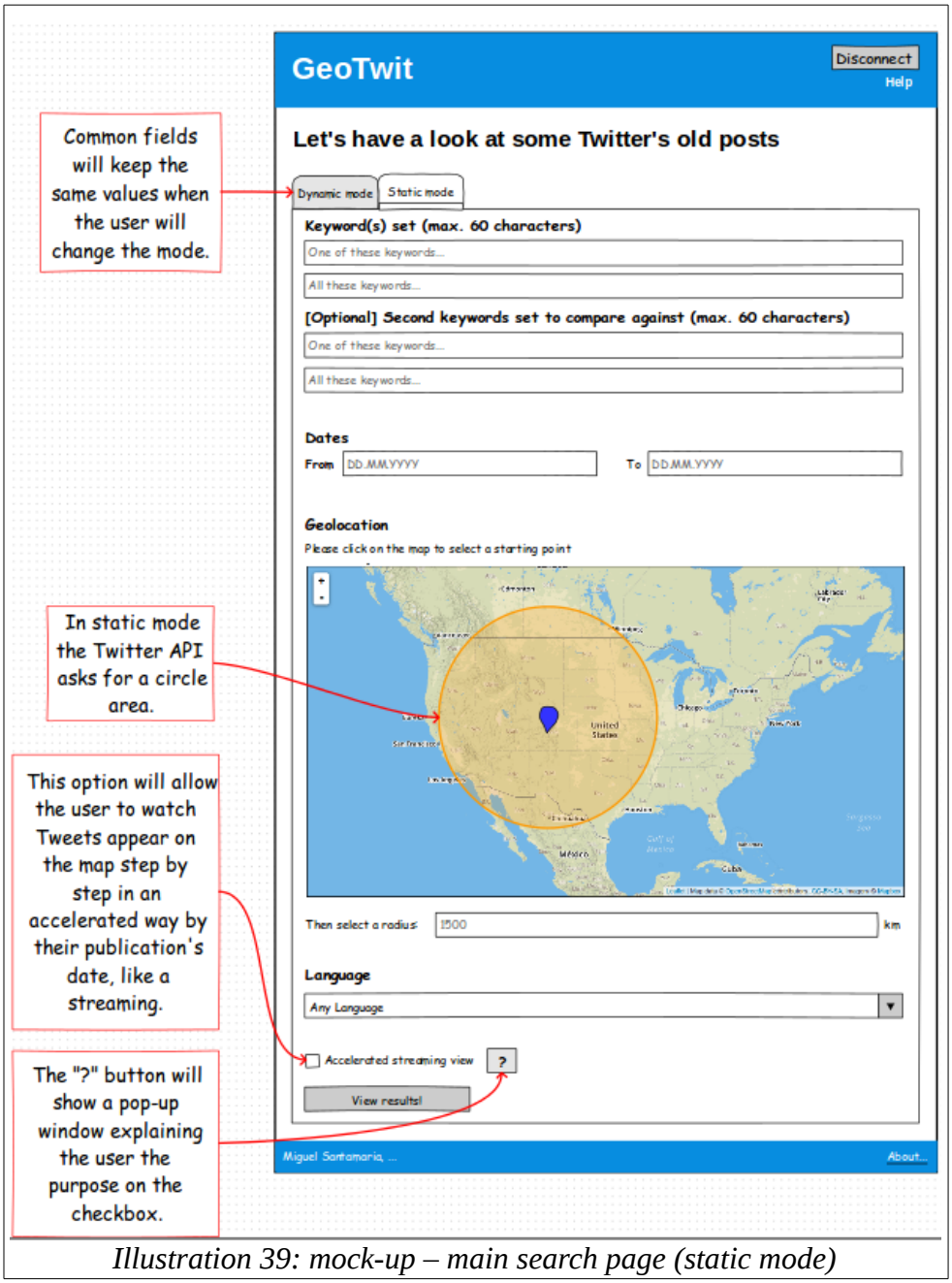


Illustration 39: mock-up – main search page (static mode)

Here again the user has to properly fill the fields and click on the “View results!” button to have a look at the results.

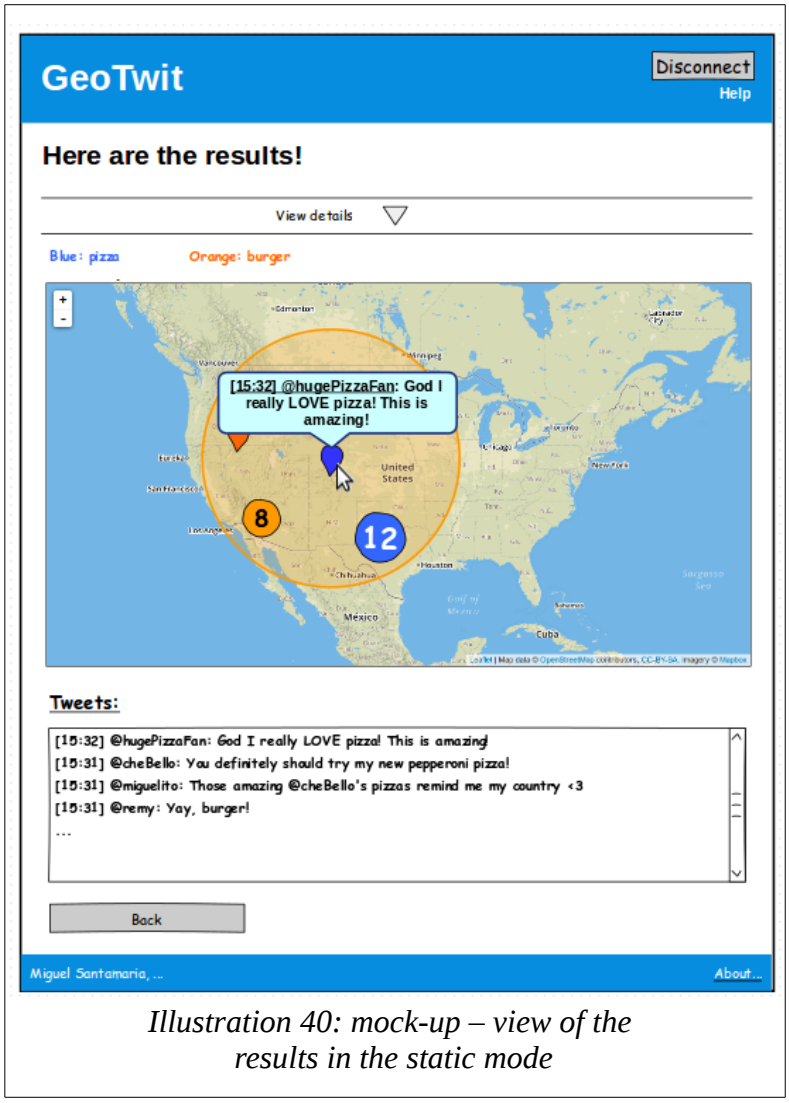


Illustration 40: mock-up – view of the results in the static mode

In the static mode, you cannot access charts.

4.2. UML Diagram of the Application

Here is the current version of the UML schema:

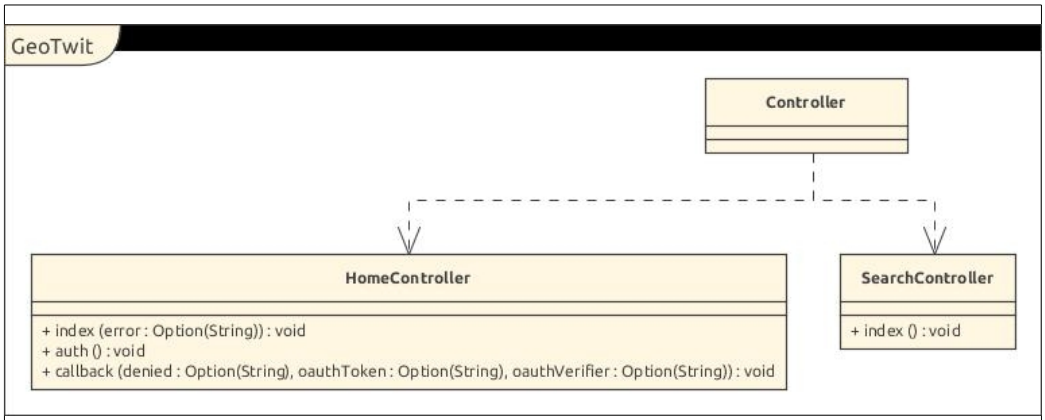


Illustration 41: current UML schema

There is currently two controllers: HomeController and SearchController.

HomeController has three actions:

1. **index(error)** → displays the home page and the error if the optional “error” parameter is set.
2. **auth** → occurs when the user clicked on the "Get Started" button of the home page. Redirects the user on the Twitter's connection page, giving it the URL of the callback function, which corresponds to the "callback" action below.
3. **callback(denied, oauthToken, oauthVerifier)** → redirects the user either on the search page if the connection was successful or on the home page if there was an error or the user denied the connection process. The status of the connection process depends on the three optional parameters: if “denied” is set, this means that the user denied the connection process and that the two following parameters are null. If “denied” is not set, this means that the two following parameters must have a value, respectively the token's string value (which will be used for the next requests to the APIs) and the “verifier” of the Twitter's OAuth process as a string (which can be asked by Twitter to verify the requests).
   This action is called anyway by the Twitter's API when the user leaves the Twitter's connection page.

SearchController currently has one action:

1. **index** → displays the main search page.

4.3. Anatomy of the Application

Here is the current anatomy of the application:

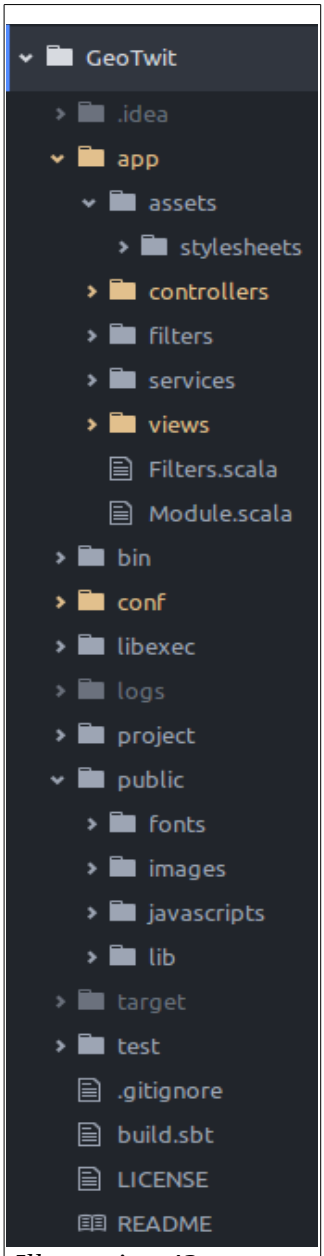


Illustration 42: current anatomy of GeoTwit

app: contains all components related to the server, like controllers, views, etc.

- **assets**: contains “less” (http://lesscss.org/) files, which are converted to “css” when the server compiles the code.
- **controllers**: contains all the application's controllers.
- **filters**: will contain filters that will be called before the processing of the HTTP requests. For example, I will check than the user is connected when he will try to access the search pages, otherwise he will be redirect on the home page.
- **services**: will maybe contain components injected in controllers.
- **views**: contains all views and sub-views of the application.

- **conf**: contains all the application's configurations, in particular the routes' configuration file.
- **logs**: contains the application's logs file.
- **public**: contains the client-side components, like fonts, images, JavaScript code (currently: JQuery and the map's code) and various libraries (currently: Bootstrap and Leaflet).
- **build.sbt**: this file is used to build the application and contains the application's description, library dependencies, resolvers and a filter used to automatically compile the Less code.

Other files and folders are not important and are in particular used by Activator and SBT to start and compile the application.

As a reminder, you can find more details about the anatomy of a Play application right here: https://www.playframework.com/documentation/2.5.x/Anatomy.

4.4. Implementation details

4.4.1. 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 home page. 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 content format, which allows me 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).

4.4.2. 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.

4.4.3. 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.

We 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 the application also gives 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.

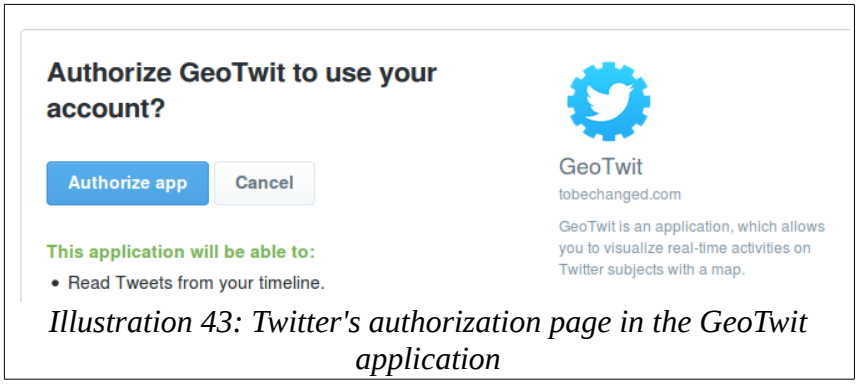


Illustration 43: Twitter's authorization page in the GeoTwit application

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 redirected him on the index page with an error.

5. Conclusions

5.1. Review of the Current State

In the beginning of the semester, I planned to complete the specifications and analysis parts of the project. I also planned to develop the basic functionalities of the application (retrieval and analysis of Tweets, basic displays on the map, etc.) as well as the documentation. Notice that this planning changed during the beginning of the semester, but I only updated it a few days ago, because I just forgot...

I succeeded in following this planning. In fact, I think I am even a little bit early: all the specifications and analysis parts of the documentation are finished, and I already created a little bit of the web site (home page, connection process and GUI of the search pages) and almost developed all the functionalities in the prototype applications (display of the Tweets on the map in a dynamical way, selection of the countries on the map, etc.). I also tried and succeeded to work on a pretty regular basis during the whole semester, as you can see in the following GitHub's graph.

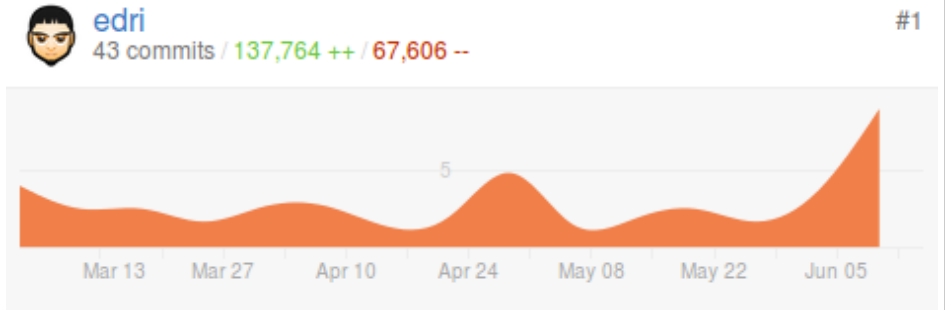


Illustration 44: graph of the commits I did during for GeoTwit the semester

Everything currently flows smoothly, then.

5.2. Encountered Problems

Here is the list of problems I encountered until now:

- The main problems until now were the time management and the personal organization: since there was way more work than I though in the other courses of the semester and since I also worked for an external client on a web site, I never was able to work during the 16 planned hours per week. However, I made myself sure to assiduously work during the free hours I had, to finally be able to follow the planning. I worked during every Mondays, often during Thursdays and sometimes during other days.
- I spent many hours trying to integrate the Twitter4J library in the Play project. It was indeed different when I tried in the pure Java prototype application, where everything was simple. I searched for a while to find the right path to include in the project's build file, and when I finally succeeded, I spent many other hours trying to complete the Twitter's connection process, since it was not properly documented in the library and some things changed with Scala.

- The learning of the different libraries and technologies took some time, but nothing really significant.
- I spend time translating the first documentation from French to English (since it was not planed in the beginning), but it was not a huge problem in the end.

5.3. Final Discussions

Even if I was a little bit scared at the beginning of this project, I think it went better than expected. Although my current work is indeed not perfect and I am sure my document should still contain typos, I still think I provided a good work until now. I hope to continue along this path until the end of the project.

For the second half, I am firstly planning to finish the application and the documentation of course, but also to mostly focus on the data analysis, by searching more interesting subjects and by running the application during days instead of minutes. I will also develop the static mode of the application, which was firstly noted as an application's extension.

6. General References and Annexes

References:

- https://github.com/: the application containing the repository of this project
- http://mashable.com/2010/08/06/twitter-mapping-tools/#yrksT7cG8Sqg: 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: 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
- http://lesscss.org/: CSS' Less pre-processor

Annexes:

- The whole mock-up schema
- Planning:
  - Initial Planning
  - Logbook
  - Hours