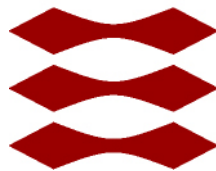


Prototype implementation of a social network application for the Polidoxa Project

Antoine Chamot

DTU



Kongens Lyngby 2012
IMM-M.Sc.-2012-111

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-M.Sc.-2012-111

Summary

Nowadays with the development of media people have access to an huge flow of information coming from various sources. This gives to everyone a wide range of viewpoints on a desired topic. However the streaming of information is mostly unidirectional, i.e. there is no possibility for the audience to control the process in any ways. Indeed with traditional media such as television or radio news are filtered step by step to ensure that only relevant information will be displayed to the public. Although such process is necessary to guarantee the information quality it is impossible for the receiver to give feedback or have any active interactions like select the source of the information or choose topics he/she wants to expand. This gives those media the power to influence publics agendas by putting forward stories they consider as newsworthy and give them prominence and space.

Internet offers an alternative since it is possible for each person to control the information he/she accesses, to choose the content he/she reads, and to interact with others. However people need some know-how to efficiently access relevant and trusted information they are looking for since the control is limited so it is usual to find any kind of hoax or garbage. It requires to be active and can be time consuming to get reliable news.

Taking account previously described limits and the fact that search engines like Google or social networks like Twitter, Facebook are for most people the starting point of much of their research emerged the idea carried by the Polidoxa project. Polidoxa aims to merge qualities of a news search engine and information coming from a trusted social network so as to offer a new searching experience. This would be done by putting user at the center and letting him influence the ranking algorithm to get more results susceptible to interest him.

Starting from this, the goal of this master is to design and implement a search engine prototype to illustrate and prove the interest of some core concepts described in Polidoxa. This application based on a chosen social network should evaluate and take advantage of the user network activities to give priority to links within a shorter relational distance. Moreover each user must be able to influence the process leading to result displaying. This running application would constitute a base for further investigations on the subject.

Preface

This thesis was prepared at the department of Informatics and Mathematical Modelling at the Technical University of Denmark in fulfillment of the requirements for acquiring an M.Sc. in Informatics.

This report consists of four parts : background and requirements, social network choice, prototype implementation and conclusion.

- Background and Requirements is composed of the introduction of the Polidoxa project on which is based this master. The second part is the description of the requirements for the application.
- Social Network Choice contains the choice of the social network made for the development. The chosen one is then described and some relevant characteristics in the scope of this project are detailed.
- Prototype implementation is explained in three four chapters. The first one details the database schema chosen for the application. The second describes the part of the application which has no direct interaction with the user so called back-end. The second part focuses on the user interaction and graphical aspects. The last part is a lightweight functional test of the prototype after successful deployment.
- The remaining chapters present conclusions and future work.

Lyngby, 01-January-2012

A handwritten signature in black ink, appearing to read 'Antoine Chamot', written in a cursive style. The signature is contained within a light gray rectangular border.

Antoine Chamot

Acknowledgements

I am grateful to Manuel Mazzara for helpful discussions during those month. I also thanks my supervisor Nicola Dragoni for his answers to few questions i had .

Contents

Summary	i
Preface	iii
Acknowledgements	v
1 Projects Descriptions	1
1.1 Polidoxa Project	1
1.1.1 Description	1
1.1.2 Algorithm Principle	3
1.2 Master Project Description	4
2 Requirements	7
2.1 Overview Use Case Diagram	7
3 Social Network Choice	11
3.1 Motivations	11
3.2 Twitter	14
3.2.1 Overview	14
3.2.2 Vocabulary	15
3.2.3 Application Registration	16
3.2.4 Existing Twitter Applications	17
3.2.5 Twitter as a news media	18
4 Database Design	21
4.1 Requirements	21
4.2 Physical Database	22

5	Back-end	27
5.1	Requirements	27
5.2	Data Collection	28
5.2.1	Twitter REST API	28
5.2.2	Task Queue	30
5.2.3	Cron Tasks executions	30
5.2.4	Cron Tasks Details	32
5.2.5	Performance issue and parallel execution	35
5.3	Data Processing	38
5.3.1	Daemon	38
5.4	Conclusion	39
6	Front-end	41
6.1	Symfony2	41
6.2	Controllers Structure	43
6.3	User Section	44
6.3.1	User Registration	44
6.3.2	Friend Static Trust	48
6.3.3	User Profile	49
6.3.4	Tweets Search	50
6.4	Administrator section	55
6.5	Testing	58
6.5.1	Deployment	58
6.5.2	Functional Testing	59
7	Future Work	65
8	Conclusion	67
A	Source Code of the Prototype	69
	Bibliography	71

Projects Descriptions

The objective of this thesis is starting from ideas developed in Polidoxa project to implement a prototype based on a social network. In order to have a clear idea of what is proposed by the Polidoxa project and provides foundations of this master thesis the first part details Polidoxa major concepts. In a second part the master project description is given.

1.1 Polidoxa Project

1.1.1 Description

Polidoxa is a project driven by scientists (Manuel Mazzara, Antonio Marraffa, Luca Biselli, Luca Chiarabini...) from different universities. In their presentation paper [7] Polidoxa is defined as ‘a Sinergic Approach of a Social Network and a Search Engine to Offer Trustworthy News’. It starts from the observation that people consume traditional media (Television, Radio...) passively. Basically they cannot have any interaction or mean to verify received information. It gives media power to influence people agenda by selecting what they consider as important and give it more space. Moreover since the control of the informa-

tion becomes more and more centralized it can potentially poses problems on the guarantee of impartiality.

Beside those media, the Internet has emerged as a new mean to access information. For most people search engine as Google or social networks as Facebook are the preferred means to search for news. These tools fix some problems by letting the user freely choose what they are interesting in and select the source of the information. However it still lack a mean for embed the notion of individual trustworthiness of a source. Thus the user has to do this work itself which is time consuming and required some know-how.

To solve this issue Polidoxa idea consists to combine the potential of both social networks and search engines to offer user new experience in document searching.

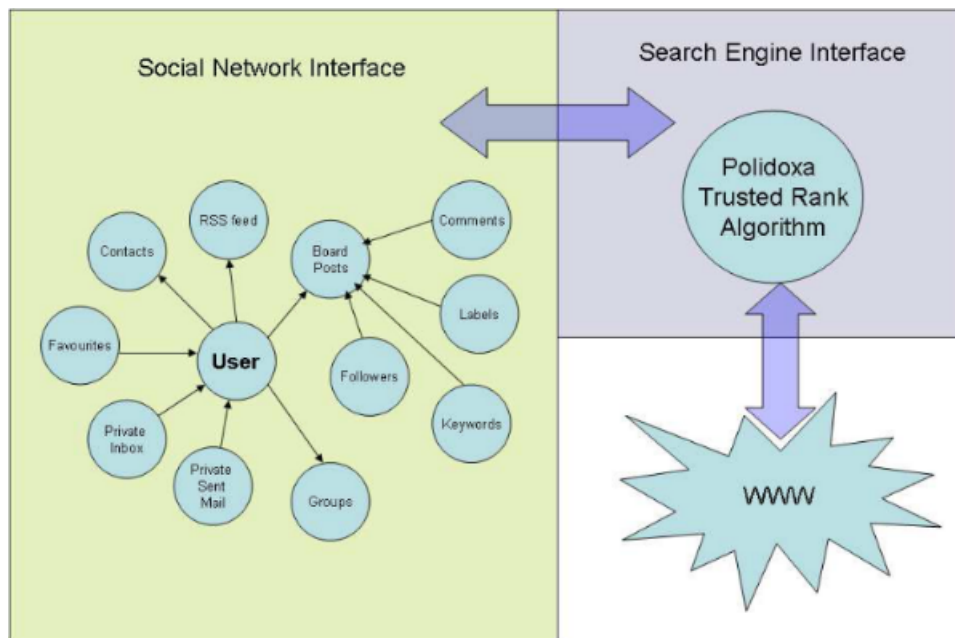


Figure 1.1: Polidoxa Platform

This new Polidoxa tool unlike Facebook or Google embed the notion of individual trustworthiness of a source. Its philosophy mentioned in the paper is the following : *we believe first in what we can directly verify, then in what our closest contacts have verified. We doubt about what people we do not know say about things we have never seen (it does not matter if this is coming from official sources) until our network of trusted contacts allows us to trust it because*

it has been verified directly by them. Somehow Polidoxa distributes the tasks of source checking amongst your network. Trusted relations are considered to be appropriate source of information. It reduce the work that a single user has to do to find reliable news.

1.1.2 Algorithm Principle

The algorithm on which Polidoxa is based favorite sources that user trust. Thus the user is implied in the process and is not simply passive consumer. The detailed algorithm is beyond the scope of this thesis. However the basic algorithm principles are following (algorithms comes from Polidoxa paper [7]):

The introduction of a static parameter representing the trust (static trust) enforce user to have active part in the process.

Algorithm 1 Configurable Static parameters

- 1 : Evaluate Trustworthiness of Contacts: by creating a contact with another user of Polidoxa, the user is asked to weight the trustworthiness of that contact.
- 2 : Evaluate Trustworthiness of a Web page: by configuring the search engine, the user is asked to weight the trustfulness of specific Web pages.

The introduction of a parameter taking advantage of the user activity. User or web pages with a large amount of like will get more trust.

Algorithm 2 Dynamic Parameters depending on activities and degree of separation

- 1: Evaluate like and dislike: the more like an article gets the more important it is
- 2: Evaluate comments in like thread
- 3: Evaluate amount and frequency of share function within a temporal interval: a high frequency within a temporal interval is an indicator of a hot and important news
- 4: Evaluate the number of comments of the post
- 5: Evaluate the number of private messages exchanged with the poster.
- 6: Evaluate keywords, labels match
- 7: Evaluate if the poster belongs to a shared group and the activities on that group
- 8: Evaluate the freshness of a document/article/post

Altogether with those two parameters the notion of distance between users is

also taken into account. The close contacts have more influence while the other see a reduction of their influence. The way it can be done is still a point to discuss. The aim of those algorithms is to embedded the notion of trust and evaluate his value as precisely as possible.

1.2 Master Project Description

The objective of this project is not to implement the Polidoxa tool, it is beyond a master work and would be premature considering early stage of the project. Nevertheless the central principle of trustworthiness of a source developed in Polidoxa is the object of this thesis. Indeed in order to illustrate how the concept of trust can be introduce in the scope of social networks a running prototype is developed and tested in this thesis.

The implemented application is based on a chosen social network which is the object of an upcoming chapter. Once this choice is done the prototype is designed to introduce the notion of trust in source. As for Polidoxa two type of trust are introduced to involve the user in the process : static and dynamic ones.

The static one is chosen directly by the user. Each user which has friends on the selected network must be able to grant them a trust value based on their own judgment. The second type is dynamic in the sense that it should evolve according to the activity of the user on the network. For instance giving extra trust to users for whom you have done many ‘like’ action or other interactions that characterize user interest for one particular source. The way those information are collected partly depends of course on the type of the network.

These trust parameters introduced are then useful to help user in his searching information process. This is done in the application by integrating trust parameters into a search engine. The search engine enable the user to search into his friends documents posted on the social network. Those other account are regarded as the sources of information that are more or less reliable. Thus the result given by the search engine and display to the user will favorite document provided by trustworthy friends (i.e have high trust value). To sum up (the requirements are incremental) :

- Messages are visualized/ranked/ordered according to static trust values of users’ network
- Adding of swarm intelligence for dynamic trust : evolution of the trust based on follower’s activity on the network

- Introduction of more complex mechanism for trust evolution such as hastags association

So the project aims to illustrate the possibility of integrate trust concept in a social network and use it while searching information.

CHAPTER 2

Requirements

After a rough description of the project and in order to make proper implementation choices it is necessary to have a clear idea of what are the functional requirements. The requirements are what is expected to be accomplish by the application in interaction with external actors. For this project actors are the user who is the application subscriber and the administrator which is an user with special access rights. They must be able to perform actions describe in this section.

2.1 Overview Use Case Diagram

According to the project description given in the previous chapter the actions that must perform the application are represented in this use case diagram :

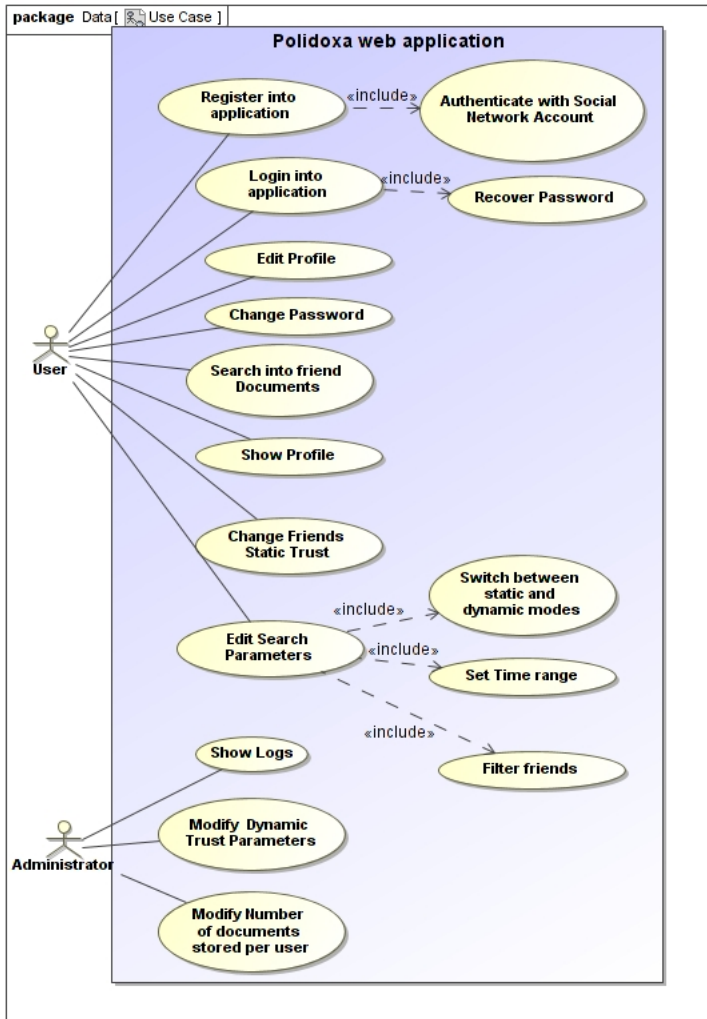


Figure 2.1: Overview use case

This diagram contains following action :

User Actions

- **Register Into Application :** An non logged-in person can register to use the application and become an user. So as to do that he has to get a valid

social network account which is used to authenticate each person during the registration process. For more details the use case is the following :

- **Description :** The user register himself to use the application
- **Actor :** User
- **Preconditions :** User has a valid social network account
- **Main Scenario :**
 1. User Identify himself using his social network account
 2. User Fill out the registration form and post it
 3. User receive confirmation email with validation link
 4. User validate his account
- **Alternative Scenario :**
 - 2.1 The Social network account is not valid
 - 2.1.1 The user is notify
 - 2.1.2 The registration process is stopped
 - 3.1 The Social network account is already used
 - 3.1.1 The user is notify
 - 3.1.2 The registration process is stopped
 - 4.1 The Registration is not validate
 - 4.1.1 The user account is removed after some time
- **Login into application :** user log-in using credentials. It includes the possibility for the user to recover his password if it was lost.
- **Show and Edit Profile :** Each user has access to his profile and can modify some parameters as email address or name.
- **Change Password :** A login user can change his password whenever he wants by entering his old one.
- **Change Static Trust :** As specify in the project description users have a static trust value associate to each of their friend. This value can be modified.
- **Search into Friend Documents :** The user can use search engine to perform keyword search on his friends documents. The documents are the ones posted by friends on the social network. Each search result ,if some, is displayed ordered following friends trust.
- **Edit Search Parameters :** The search action previously described is done according to some parameters which can be modified. The way the trust is calculated can be changed from static to dynamic (including user

network interactions). The time range on which the searching is done can be chosen. Eventually amongst all user friends it must be possible to restrict the search to only some of them.

Admin Actions

- **Show logs :** The administrator should be able to see logs of tasks running in background.
- **Modify dynamic trust parameters :** The dynamic trust is calculated using a formula with coefficients. The application administrator must have the possibility to freely modify coefficient values.
- **Modify Number of documents stored per user :** If some documents must be store then the administrator should be able to set their quantity to limit database size.

Altogether with those functional requirement some non-functional can be included : The application should be intuitive and do not require any installation process for people who want to use it.

Social Network Choice

According to both description and requirement the thesis is based on a social network. The choice of this network influences the design part and therefore must be done carefully. In this chapter the selected network and the reasons that lead to this choice are detailed. In the last part a specific consequence of this choice on the nature of the application prototype is discussed.

3.1 Motivations

Currently there exists many different kind of social networks on the Internet. The two most popular of them are Facebook and Twitter but many others exists such as LinkedIn, MySpace, Google Plus+, Ning etc. The application developed in this project has to be based on one of those networks, so it is important to know which one would be the best choice for a search engine implementation.

The comparison between those networks is made taking into account both technical criterias such as provided programming interface (API) limits, content structures and non-technical such as the number of active users which implies more interaction between users and so better data collection by the application. To achieve this objective pro an cons arguments for each of those social networks

have been listed so as to eventually choose the one that would best fit project requirements. In order to avoid overloading this section, the comparison will focus on three chosen examples. The first two are the most popular (Facebook and Twitter) since it enables to reach more people active on these networks and have quite well developed and documented API to access users data which are two crucial points. The last one is a maybe less known alternative network called Ning.

Facebook :

- Pro :
 - Most Popular of the networks (over 900 million active users).
 - Well documented client libraries to request twitter have available in various languages
 - Not very strict restriction on number of API calls (around 600 calls/600 sec)
- Cons :
 - Search API limited to two weeks back
 - Retrieve posts using Facebook Query Language is limited. Basically it is only possible to retrieve data that are displayed to the user by consulting their friends profile and clicking on the "get older information".
 - Information is contained into heterogeneous elements (post, comment, messages...) and many of them contain only images.
 - Facebook is not recent so it is more difficult to be innovative

Twitter :

- Pro :
 - On track for 250 millions active users at the end of 2012
 - Quite new so more possibilities of innovation
 - Available client libraries to request twitter in various languages
 - Relatively uniform and light documents (tweets of up to 140 characters)
 - Tweets messages contain tags and mentions that could be useful for data mining

- Essentially written information
- Cons :
 - Twitter REST API limit number request per identified user per hour by around 350.
 - Number of results return per call is often limited (no more than 200 result)
 - Regarding the search API provided it is limited to the previous 6-9 days
 - Timeline retrieve for each user is limited to 3200 tweets
 - Provided API is often changing so need to keep up to date.
 - Site Streaming still in beta version with restricted access

Ning :

- Pro :
 - No explicit restriction on the number of call that can be addressed (however it still exists)
 - Less user than its competitors but still 32 million users of the 1.5 million Ning-built social networks.
- Cons :
 - No so well documented client libraries as most popular networks
 - Documentation specifies that only "recent" posts are return.
 - Information is contained into various element in which some are only picture.

Conclusion :

The comparison between social networks to fit the project requirements puts forward several relevant points. First most of them provide API to search trough all documents exchanged between user, however due to the amount of data it is quite limited in time. Therefore it is difficult to use such API to search into friends' documents for this project.

The second major restriction common to all networks is related to the restriction on the number of call that can be addressed using API. Even if this limit is not the same for each it leads to a common problem of retrieving data while avoiding overload. To workaround this problem few social network as Facebook

or Twitter (in beta version) provides streaming API that enable developer to open permanent stream to collect information.

The last point is related to the structure of exchanged data. On one side in “traditional social networks” such as facebook the exchanged datas are composed by various elements such as comment, post, messages, photos. On the other side twitter users exchange homogeneously formatted documents : tweets mainly composed of text even if it is possible to attach photos or videos.

So it seems obvious that getting information on user activities and data mining would be easier using such network as twitter because of both the nature of data exchanged and the exchange rate which is generally higher than with other form of network either personal (twitter, ning..) or professional (LinkedIn..). To sum up the choice has been done in favor of twitter for the following reasons :

- Formated and textual data exchanged are easier to retrieve, store and analyze. Moreover tweets contain special tags that could be use for data mining.
- Numerous active users
- Relatively new network so open for innovations
- Well documented API and libraries available in various programming languages
- Higher exchange rate than other networks
- Many news are exchanged on this network (see section [3.2.5](#))

Now that the choice is done the following section investigates in greater detail Twitter.

3.2 Twitter

3.2.1 Overview

Twitter is an on-line social networking service and micro-blogging service that enables its users to exchange text-based messages known as "tweets". It was created only six years ago (March 2006). This web service is now very popular with over 500 million active users in 2012 which exchange around 340 million tweets daily. Unregistered users can read tweets whereas registered users can also post new ones. Some statistics on the twitter usage have been published

and notably by sysomos company[6] in which they analyzed data of 11.5 million Twitters accounts. It underlines some point that can be relevant for the application :

- 85.3 % of all Twitter users post less than one update/day
- 21 % of users have never posted a Tweet
- 93.6 % of users have less than 100 followers, while 92.4 % follow less than 100 people

Those information are interesting in the scope of our application. The fact that most people have less than 100 friends and each of them post around 1 tweet per day means that the application should not have to handle too much traffic per user.

3.2.2 Vocabulary

Twitter has a special vocabulary to nominate elements that are offered by the service. These elements will appear all along this report so it is necessary to describe them for those who are not familiar with this jargon.

- A **tweet** is a text based message exchanged between twitter users. In contains up to 140 characters.
- A **hashtag** is a word or phrase contained in a tweet and prefixed with the symbol #.
- A **mention** is a twitter username contained in a tweet prefixed with the symbol @.
- A **friday follow** is the combination of the hash tag #ff with mentions. It is used every friday by user to suggest people to follow (those indicate by the mention). Ex: #ff @antoinecahmot
- The **Home Timeline** is the one user see when they login to twitter.com. This is the place user receive most recent statuses posted by the authenticating user and the users they follow. The most recent tweets appear at the top.



Figure 3.1: Home Timeline

- A **follower** is another Twitter user who has followed you. To follow someone on Twitter is to subscribe to their tweets or updates.
- A **friend** on twitter can have several definitions. In this project users' friends are the people you follow. A more restrictive definition could impose that users have to follow each other which is not the case here.

3.2.3 Application Registration

To develop a twitter application it has to be first registered with Twitter. Indeed before developing any application it is necessary to get a production URL. The process begin by opening the link to the developers website (<http://twitter.com/apps>) and click the Register a new application link. Once the registration process is terminated consumer key and consumer secret are generated. These unique credential are necessary for the application to interact with Twitter. Thus they are a fixed parameters for the application.



Figure 3.2: Application Registration Page

What is interesting to notice are the callback URL and the access type. The callback URL must be the production address of the web application. During the authentication process it is also used to redirect user after success. Access type defines operations that the application needs to do (read and/or write). For project purpose the read only access type is enough.

3.2.4 Existing Twitter Applications

Many applications have been developed since twitter provides API to access data. It is difficult and not relevant to describe all existing ones. Thus the focus is done on applications which aim is to search through tweets like the project application. This list is not exhaustive and introduce most popular examples :

- **SnapBird** SnapBird.org is an application that enables to search either in someone timeline, friend's tweets, someone favorites, and some other possibilities. This application is based on twitter REST API, thus this search is restricted by the API limit (3200 tweets). It display tweets order by time. The interface is easy to use and the access is free.
- **Topsy** Using Topsy.com's free Advanced Search, it is possible to search in an user tweets and subscribe to the results via RSS. The display results can be broken up into past hour, past week, past month or all time. It is also limited by the API to the last 3200 tweets. But it is not only for twitter, indeed it is possible to search through Google plus. Moreover the

video and photos associated with tweets can be search within the entire web.

- **SocialSearching** SocialSearching is designed for both Twitter and Facebook. This application has a simple interface where user can enter the account on which perform the keywords search.
- **ThinkUp** This tool capture all the user activity on an user network and register them in a database. Then all information are display to the user in various way (graphics, maps). It has a search function that enable the user to search into his tweets, they are ordered by created time. This application is restricted for experts since it has to be install on a web server together with mysql.

So as one can see they are already different applications that are available for searching. However most of them are limited to the last 3200 tweets back in time. *ThinkUp* is different since it registers your own tweets in a local mysql database but require user to do quite heavy installation. Moreover none of them embed the notion of trust or use the network activity of the user to make more than just a simple keyword search ordered by date.

3.2.5 Twitter as a news media

Twitter is a micro-blogging service which is regarded as a social network but can it also be considered as a news media ?. This is the discussion of the paper entitled *What is twitter, a Social Network or a News Media ?* [12]. In this document the entire Twitter site has been crawled to obtain millions of user data. By analyzing twitter space they underlines some point that are relevant in the scope of this application. Indeed the Polidoxa project is made for people to search news on the web while basing trust on social network. So a legitimate question is to know if twitter is more than a social network or even further if twitter is more a news media than a social network. The answer to this question gives also an hint on the nature of the developped prototype. Is it a tool to only search within friends' personal tweets or an efficient mean to access news ?

According to the previously mentionned paper the twitter trending topics are in majority (over 85%) headline news or persistent news in nature. Moreover a close look at the reciprocity in twitter shows thats 67.6% of users are not followed by any of their followings in Twitter. So one can conjecture that these users regard twitter rather as an information source than a social network site. Another characteristic describe in this paper is the degree of separation which is the average length to connect any two people. According to the study it is

around 4 which is quite short for such a big network as Twitter. This could bear out the idea of Twitter's network other than only social network. An idea on how the information is spread on twitter network is also available. The retweet mechanism introduced by twitter seems to play a prominent part. Thanks to retweet mechanism users can spread an information of their choice , which gives the power of individual user to dictate which information is important. Regardless the number of followers a user has, once tweets starts spreading via retweets they are expected to reach a certain number of audience. Thus it can be regarded as the rise of a collective intelligence that solves the problem of traditional media dictating the headlines. The user is involved in the process in a sense that you can choose to broadcast what is really important for you. In conclusion all these aspects underlines that it is not senseless to see twitter as a new news media.

Therefore an application for searching through his friends tweets is also a quick mean for users to find some fresh news likely to interest them. It gives an hint on the interest of a tool that could facilitate the search by putting forward trustworthy news that have been relayed by your friends or directly created by a news account that the user follow.

Database Design

4.1 Requirements

In order to fulfill application requirements a database is obviously necessary. To design database schema correctly this section describes the data that should be stored :

- Each person who want to use the application need to register using his/her twitter account. This implies to get a table containing each user and containing information to identify the associated twitter account (twitter id, twitter username). Moreover each user friend (twitter accounts follow by the user) must be stored.
- One of the requirements is to be able to search within friends' tweets. Unfortunately the provided twitter search API. imposes quite strict limitations on how long back in time it is possible to perform searching. This restriction is not suitable for this application if the objective is to get a quite large data set for each user. So as to overcome this limit the choice of storing user friends tweets in the database has been made. This implies that a table should contains tweets associated to their sender.
- The possibility of choosing a static trust associated to his/her friend has to be

integrate to the database schema. It is easily done by created a one to many association with attribute between users.

- The dynamic trust calculation requires user activities to be tracked. Those activities are relevant in terms of how user perceive his/her friends. For instance if an user follows a newspaper account on twitter and regularly retweets news delivered by this friend then it is an indication that show that he/she trusts this source. The activities that have been chosen to be monitored and stored are the following : retweet, favorite, mention and friday follow.

- Besides application requirement some other table are necessary to ensure application running according to choices made in terms of technology. That is why one table is necessary to contain cron tasks that are executed. Another one is devoted to sphinx and eventually a cache table to store rough tweets. The reasons of those choices are detailed in the next chapters.

4.2 Physical Database

Following requirement the database schema created is the following :

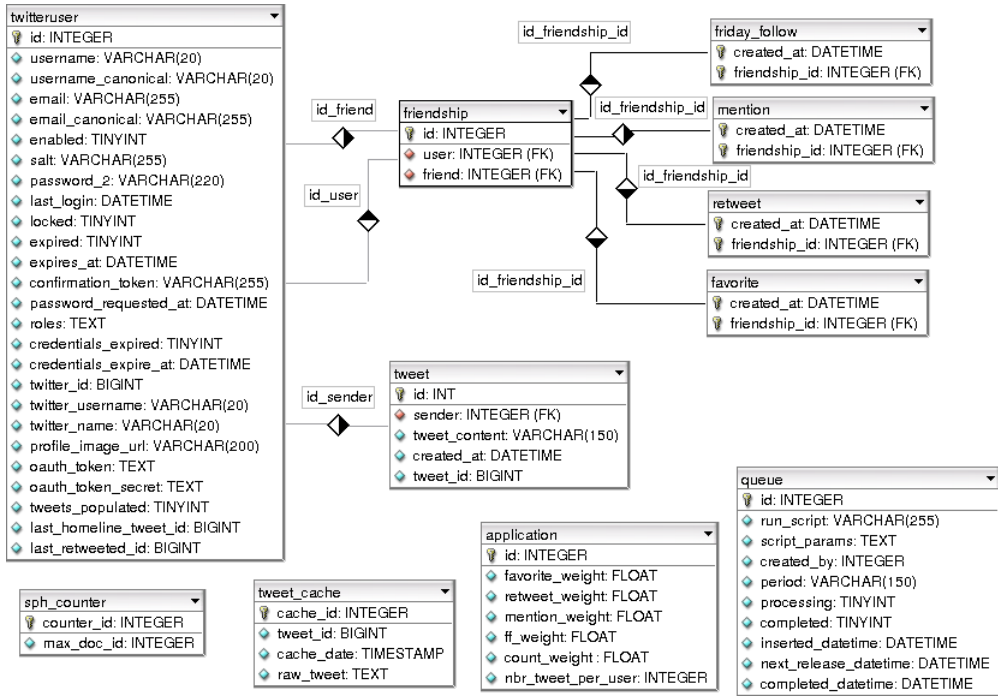


Figure 4.1: Database Schema

This is composed of the following tables :

- **User** : This table contains the application users together with their twitter friends. It is composed of the following rows :
 - username, cononical_username, email, password : Those information are provided by the user when he fills the registration form to use the application.
 - twitter_username, twitter_name, twitter_id, profile_image_url : Those information are collected during the registration process when the user is identified with his twitter account. The twitter id is unique so an user can only be register one time with the same twitter account
 - last_hometimeline_id, last_retweeted_id : Those two ids are used to remember last data retrieve from twitter. It is usefull for cron jobs to avoid useless call.
 - enable : This boolean is used to check that the user confirmed his inscription by email

- expires_date : The date when the user will be removed from the system. The user can be removed because he didn't confirm his application subscription, he has been inactive for too long or he revoked the application.
- populated : This boolean is used when an user is created. The value changed from 0 to 1 when first 3200 tweets are recovered from twitter and stored in the database. It enables to pre-populate database with tweets sent by this user.
- **FriendShip** : This table contains friendship relations between two users. This enables to store one to many relations between the users contained in the database. The trust attribute represents the static trust granted by an user to his friend.
- **Tweets** : This table contains users tweets retrieved from twitter. The information stored are voluntarily minimized to keep only what is really necessary for the application and prevent database from being overloaded. It is composed of the following columns :
 - sender : It contains the id uniquely identifying the user who sent this tweet and thus is its owner.
 - twitter_id : This big integer is a unique number given to identify each tweet and prevent duplicate data.
 - tweet_content : Textual content of the tweet, it can contains up to 140 char.
 - created_at : Creation date of the tweet.
- **Retweet** : Table containing retweets associated to each friendship relation.
 - friendship_id : Foreign key references the friendship associated to the retweet.
 - created_at : Creation date of the retweet
- **Favorite** : Table containing favorites associated to each friendship relation.
 - friendship_id : Foreign key references the friendship associated to the favorite.
 - created_at : Creation date of the favored tweet
- **Mention** : Table containing mentions associated to each friendship relation.
 - friendship_id : Foreign key references the friendship associated to the mention.

- created_at : Creation date of the tweet containing the mention
- **Friday Follow** : Table containing friday follows associated to each friendship relation.
 - friendship_id : Foreign key references the friendship associated to the friday follow.
 - created_at : Creation date of the tweet containing the friday follow
- **Application** : Table containing only one row with some application parameters i.e the coefficients used to compute dynamic trust and the number of tweets to store per user.
- **Queue** : Table containing tasks to be executed by the cron
 - period : Define the execution period of the corresponding task by the cron
 - next_release_time : The next date when the task can be executed
 - processing : boolean variable used to lock the task and avoid multiple instance running at the same time
 - completed : Indicate if the task is completed or should be executed again by the cron
 - created_by : Identify the user that created this task. It is used to restart completed tasks.
- **sph_counter** : This table is necessary for the sphinx search engine. This component is evoked in the front-end chapter.

The relational database management system used in this project is mysql and the storage engine is InnoDB.

Back-end

To fulfill overall requirements the application needs to collect, process and store information from twitter. This is basically what is done by the application and which is not visible by the user. That is why those functions have been grouped in a piece called *Back-end*. Different aspects and challenges regarding the implementation of such processes are discussed in this section.

5.1 Requirements

Before exposing the implemented solutions it is necessary to clarify which are the requirement specific to this part based on the global ones.

First, as the main function offered by the application is to be able for the user to search through its friends tweets it is necessary to get the list of friends for each user together with a piece of information on them (names, profile image...) and then of course obtain their tweets content to perform searching on it.

Regarding the second aspect of the application which is the analyze of the interactions between the user and its friends to provide dynamic trust, extra collects are necessary : favorites and retweets. For more advance analyze of user activity data mining on tweets content is also expected.

Eventually the database need to be cleaned by removing inactive user or checking

validity of stored credentials for instance. Altogether the requirements are the following :

- Retrieve friends tweets
- Register Application users with associated data (tokens,username,..)
- Collect information on twitter user friends (names,twitter id ..)
- Collect and process data on user network activity (retweets,favorites)
- Perform some data mining on tweets.
- Clean database (remove users,check tokens validity...)

5.2 Data Collection

5.2.1 Twitter REST API

So as to perform search within tweets posted by friends the first option would be to use the provided twitter search API. However according to the documentation *The Search API is not complete index of all Tweets, but instead an index of recent Tweets. At the moment that index includes between 6-9 days of Tweets.* It means that with such tool it is impossible to search within a large number of tweets or period which is quite restrictive. To workaround this problem the chosen solution is to use a local storage.

A MYSQL database setup on the server is feed with friends user tweets, so the limitation is transfered from twitter to the database storage capacity.

This database is filled by resources gained from twitter using one of the two existing API :

- The REST API enable to periodically query twitter databases.
- The Site Streaming API which open a permanent data stream with each user.

The second solution would be the more practical because it does not suffer from limitations and is a real time solution. However the site streaming API is still in beta version an therefore its usage is limited and requires special authorizations. So the first option has been chosen and implemented in this project.

The REST service is available by sending GET requests containing parameters. In response JSON data are returned. Moreover to perform twitter calls several client libraries are available in various languages. The one chosen for this project is *Twitter-async* written in PHP.

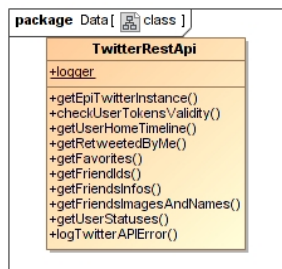
Numerous resources are offered to the developers. Amongst them the ones useful for the project have been selected and listed as following :

- **GET statuses/home_timeline** : Returns the most recent statuses, posted by the authenticating user and the users they follow
- **GET statuses/retweeted_by_me** : Returns the most recent retweets posted by the authenticating user.
- **GET statuses/user_timeline** : Returns the most recent statuses posted by the authenticating user
- **GET friends/ids** : Returns an array of numeric IDs for every user the specified user is following.
- **GET users/lookup** : Return up to 100 users worth of extended information, specified by either ID, screen name, or combination of the two
- **GET favorites** : Returns the 20 most recent favorite statuses for the authenticating or specified
- **GET account/verify_credentials** : Returns an HTTP 200 OK response code and a representation of the requesting user if authentication was successful; returns a 401 status code and an error message if not

Each of those request return formatted JSON object as the extract below :

```
{
  'name' : 'Matt Harris',
  'id_str':'777925',
  'followers_count':1025,
  'profile_background_tile':false
  ...
}
```

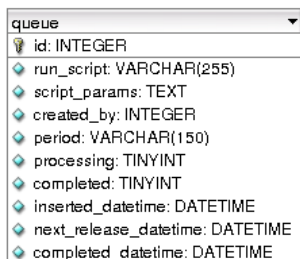
It contains high quantity of information which need to be filter to keep only relevant ones. It is also possible for the request to fail, in such case an error code is return. Those previous function are partly implemented by the *Twitter-RestApi* class which is used as interface between the external library and the rest of components :



This class is in charge of making successive GET requests to retrieve information from twitter. In case it fails the exception is caught and the error logged in a file. It doesn't prevent the execution from continuing and perform other calls. The different methods return arrays build by parsing JSON format.

5.2.2 Task Queue

The major issue using REST API is to stay below the limit imposed by twitter of 150 request per non identified user or 350 for an identified one (i.e using identification tokens). To keep control on the number of request an efficient strategy is necessary. First the identification tokens specific to each user need to be stored in the database during the registration process. This way they can be reused later to identify the user making the call to twitter and give him an higher credit. Secondly, as the limitations are made based on a period of one hour, a good practical to optimize the given credit is to spread requests over this period. This is done using periodic tasks and choosing appropriate period to avoid overloading. This solution has been implemented by combining a cron table together with a task queue represented as a table in the database.



This *queue* table contains for each task the name of the associated scripts together with a list of information necessary to their executions. Thus for each run of the cron task ready to be executed in the table are identified and started. This flow is detailed in the following section.

5.2.3 Cron Tasks executions

As mention in the previous section each cron task is created by adding a row in the queue table. Once this is done a cron will periodically execute a script which perform actions represented in the following activity diagram :

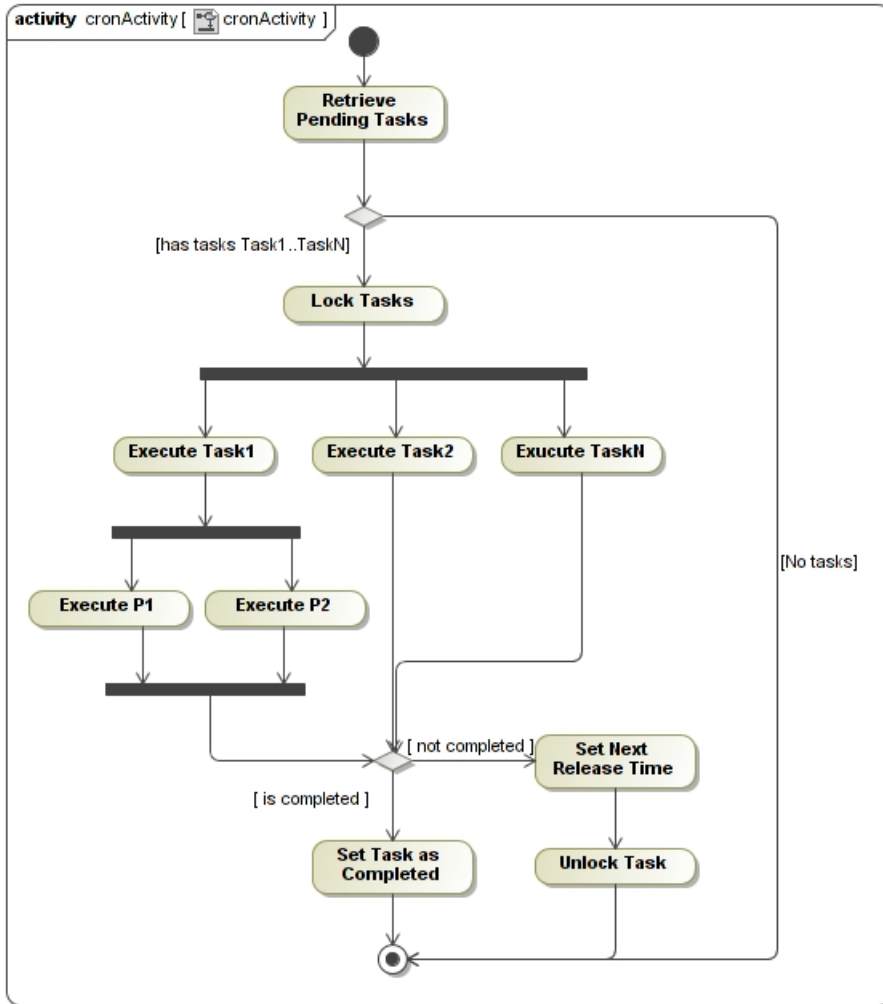


Figure 5.1: cron Activity

First the pending task are selected from the queue table. A task is considering as pending when it is ready to be executed which means that it is in an idle state and its next release time is greater or equal to the current time. To check that a process is in an idle state two boolean are used. The first one *processing* enable to lock the task when its is currently executed by one process, it prevents the same task to have several instances running at the ame time. The second variable *completed* is set to 1 when the task is finished. So a task is ready if

those two columns *processing* and *completed* are both 0.

Then if some tasks have been selected they are locked (i.e *processing* equal to one) before being executed in parallel. The PHP script fork the main process into as many child processes as it is necessary. They are executed independently because they can have very various execution time. Some of the task are themselves forked again to reduce waiting time during twitter call (This point is discussed further in a next section).

Once one task finishes its execution two case are possible. Either the task is completed which means that its doesn't need to be executed again in the future. This is for instance the case of the script in charge of populated friends with first 3200 tweets. In this case the task is marked as completed (i.e *completed* equal to 1) and the execution terminate.

Otherwise the task is not completed which is the case of the ones that should run periodically forever. In such case the next release time is calculated using the *period* column value and the *processing* value is reset.

5.2.4 Cron Tasks Details

In the previous section the tasks execution strategy by the cron have been presented. This part focuses on the task content and the design of the php classes associated. Indeed the first part deals with the functional description of the scripts associated to each tasks before entering into more technical discussion in the second part.

5.2.4.1 Description

The tasks that must be executed by the application can be divided into two group : the one where tasks have to be executed one or several times before completion and the one with periodic tasks that are executed indefinitely.

The first group is composed of :

- **SetupUserAccountTask** : This task is executed one time right after the registration of a new user. It first recovers user's friends information from twitter to created relationships in the database. Then it executes in parallel the following actions. Populate the database with first tweets from the user home timeline and retrieve both retweets and favorites from twitter. Eventually another task called *RetrieveUTTask* associated to the newly created user is inserted in the queue.

- **RetrieveUTTask** : This job generated by the previous one is executed one time per hour until completion. It is associated to one user and is in charge of pre-populate friends with the maximum number of tweets it is possible to get (i.e 3200). Since each friends required until 16 requests the number of friends pre-populated for each run is limited to around 10 to respect the API quota. The task is completed when all friends have been populated.

The second group is composed of :

- **RetrieveRTTask** : This job is in charged to periodically request new retweets made by the application users. For each of them identification tokens contained in the database are used to ask twitter retweets made from the *last_retweeted_id* of the user. If some are return then they are stored in the cache table and the *last_retweeted_id* is updated for the next execution.
- **RetrieveHTTask** : This job is in charge to periodically request users plus friends new tweets. For each of them identification tokens contained in the database are used to ask twitter tweets made from the *last_home_timeline_id* of the user. If some are return then they are stored in the cache table and the *last_home_timeline_id* is updated for the next execution.
- **RetrieveFVTask** : This job request application users favorites. The user favorites contained in the database are updated with the new ones.
- **RetrieveNTTask** : The goal of this script is to update user friendships to keep database synchronized with twitter. Indeed the relationships created during the user registration can evolve. For instance if a new friend is added on the twitter account it must be reflected in the application.
- **DatabaseCheckOutTask** : This task is executed only one or two time per day. It is in charge of various things. Firstly removing expired users from the system by correctly cleaning all associated tables in the database. Secondly checking if the access rights were revoked by an user (it is possible for each user to revoke access to his twitter account) or if the user was inactive for more than one year. In both case a email is send to notify him that his account will be removed within one day if he doesn't log in the application before. Moreover the retweets, mentions and friday follow older than one year are removed from the data tables.

5.2.4.2 Database Access Class Diagram

As it is mentioned in the above tasks description each of them needs to interact with the mysql database. So as to facilitate this the **DAO Pattern** have been implemented using following classes :

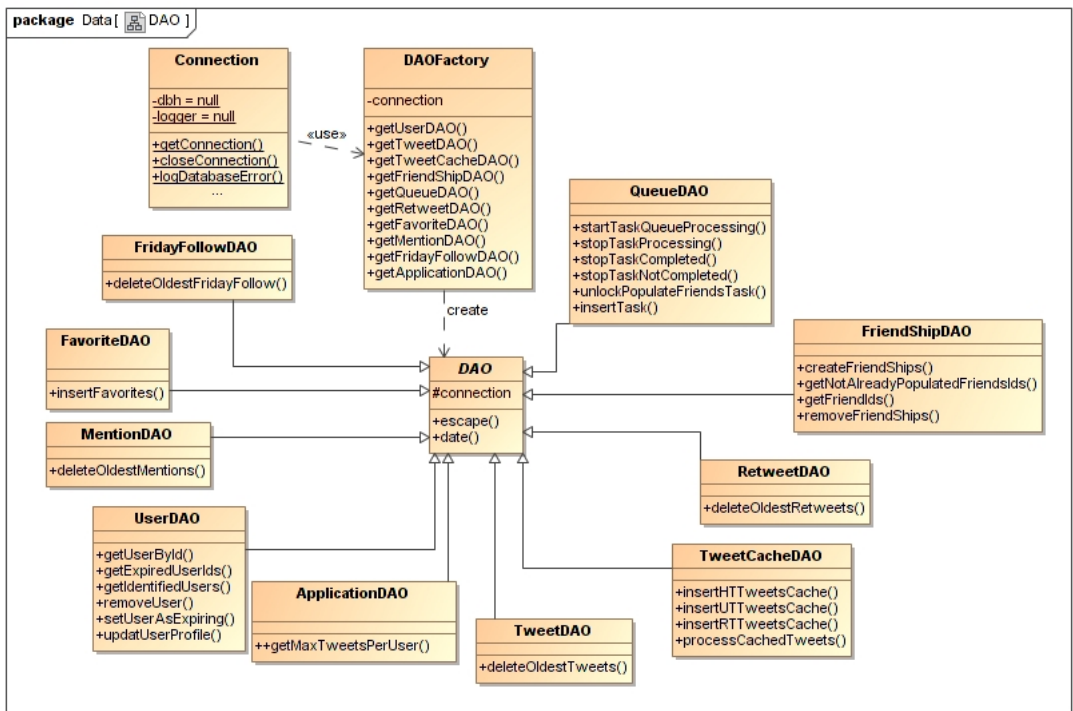


Figure 5.2: DAO class diagram

The data access object (DAO) is an abstract interface to database, providing some specific operations without exposing details of the database. These operations are the methods of class whose names have DAO prefix in the above schema. This isolation separates data the application has to access and data types from how these needs are be satisfied with the database schema. Thus those DAO classes make all SQL request necessary to perform operations on the database, in case of failure of one request the error in reported in a specific Mysql log file.

5.2.4.3 Cron Tasks Class Diagram

The DAO describe previously is specifically useful for classes representing the different tasks described in the first part. Altogether those elements can be represented as follows :

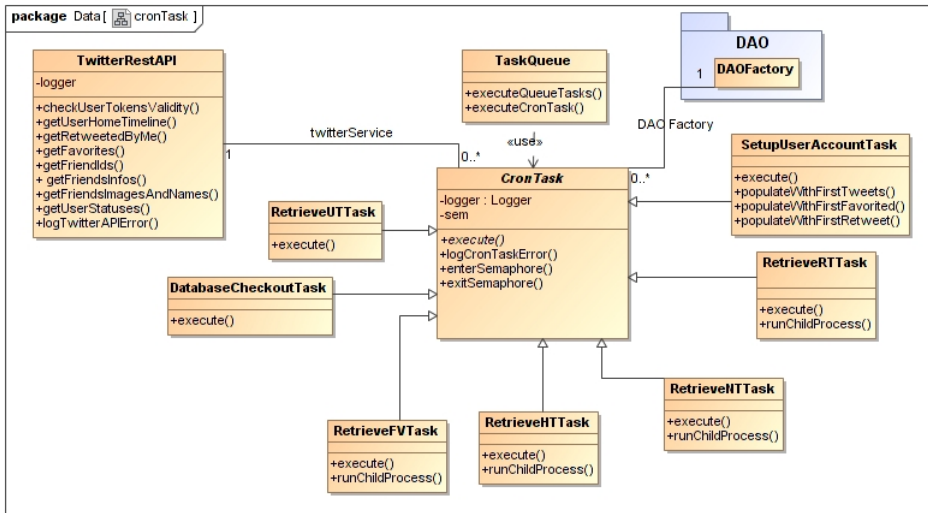


Figure 5.3: DAO class diagram

In the figure above is represented the *CronTask* which is associate to both the *TwitterRestAPI* and *DAOFactory* classes which realize the interface between respectively twitter and mysql. This abstract class contains an *execute* method which is redefined by each of its inheritors. This method is the one which launches each task execution from the *TaskQueue* class. A instance of this class is indeed used by the script started each minutes by the cron. This instance simply executes the *executeQueueTasks* method witch run all queue tasks following the activity diagram presented in the Cron Tasks execution section 5.2.3.

5.2.5 Performance issue and parallel execution

Since the tweets are retrieved by querying periodically twitter, the application will not be real time. The user will experience a delay between the time a tweet is send and the time when it is actually taken into account by the application. This delay is directly related to the frequency with which task are executed.

This is especially critical with *RetrieveHTTask* which is supposed to recover user and friends tweets. So it is important to launch it with quite small period. However each execution of this task consume twitter calls which are limited to 350. That is why it has been decided to use a period of 5 minutes. Knowing that the maximum number of request per execution per user is 4, it gives a number of request up to 48 ($12*4$) for one hour. It leaves more than 300 request for other jobs which is enough. However this reduction of the period raises another issue which is the following : the execution time of the task must not exceed his period. This is not a problem for a single user but becomes an issue when their number increase. Indeed assuming that the elapse time between the sending of the request to twitter and the response reception is at most of 2 second (time measured with php) then for 4 calls it leads to at most 8 second of time execution. This calculation is independant of the material used and implies that in a sequential process *RetrieveHTTask* can be executed for at most 37 users ($60*5/8$) to keep below 5 minutes.

The solution chosen to overcome this problem is to execute some process concurrently within task that needs higher execution speed. This is represented with the following petri-net :

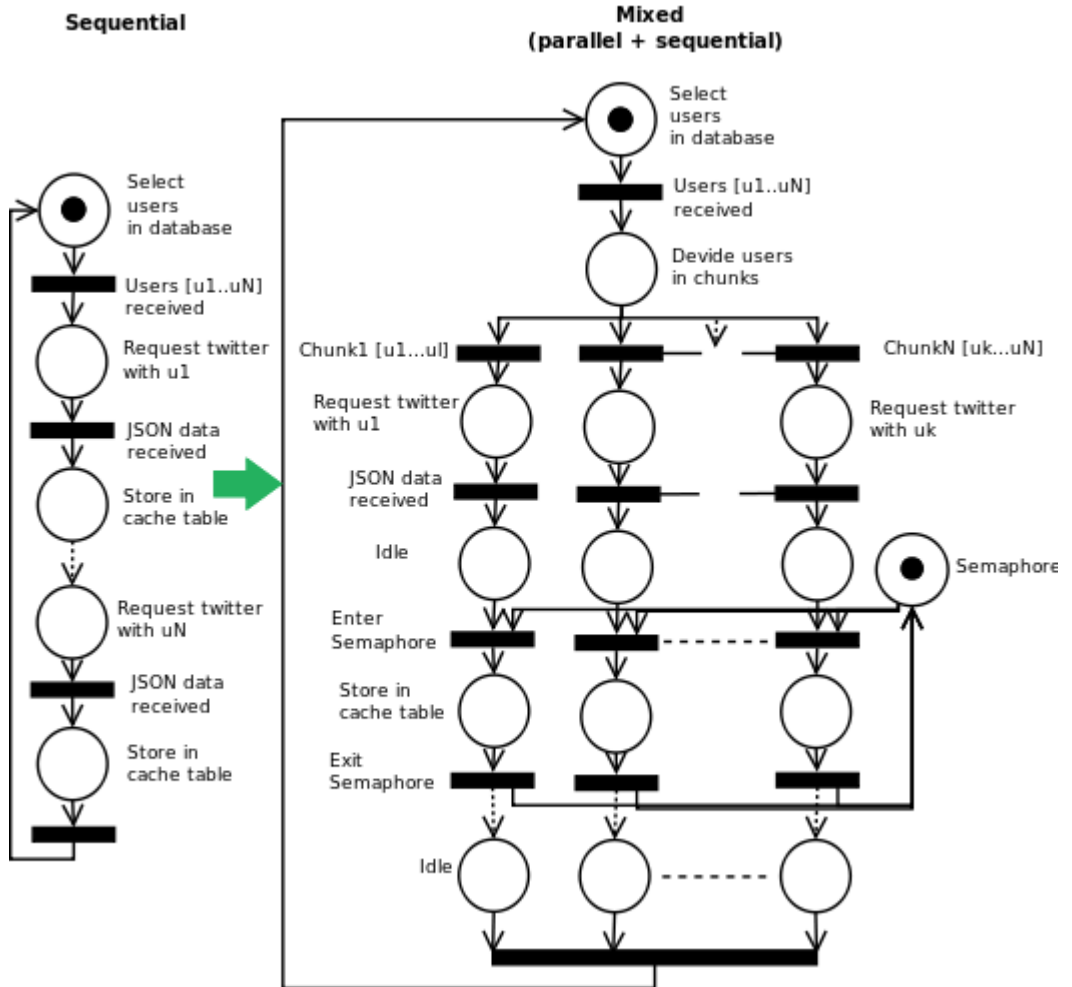


Figure 5.4: Petri net diagram

Instead of making complete sequential call to twitter as represented on the left side of the picture, it is transformed in a mixed architecture. The list of application users is divided in several chunks. Each chunk sequence is then executed in parallel. Moreover so as to limit the number of connection with the database the connection is open at the beginning and then shared between each process. This behavior is not handle by mysql so it is necessary to introduce an element to prevent errors that can occur when two process perform database operations at the same time. This element is a semaphore, it will prevent several processes to access database resource at the same time. It introduces a

delay when the process waits for the semaphore, however the database operation performed on cache table are minimal and really fast so it can be neglected compare to the time require to request twitter.

5.3 Data Processing

In the previous section the way twitter API is used to perform a bunch of cron tasks that recover critical information was detailed. This collect is done independently of the processing according to twitter recommendation. It enables to avoid bottleneck problem in case data processing is time consuming. This is precisely this data treatment which is discussed in this part.

5.3.1 Daemon

As described in the cron section the raw tweets are stored in mysql cache table by the different tasks. Then they are retrieve and treated before the insertion in the final database as represented in the following figure :

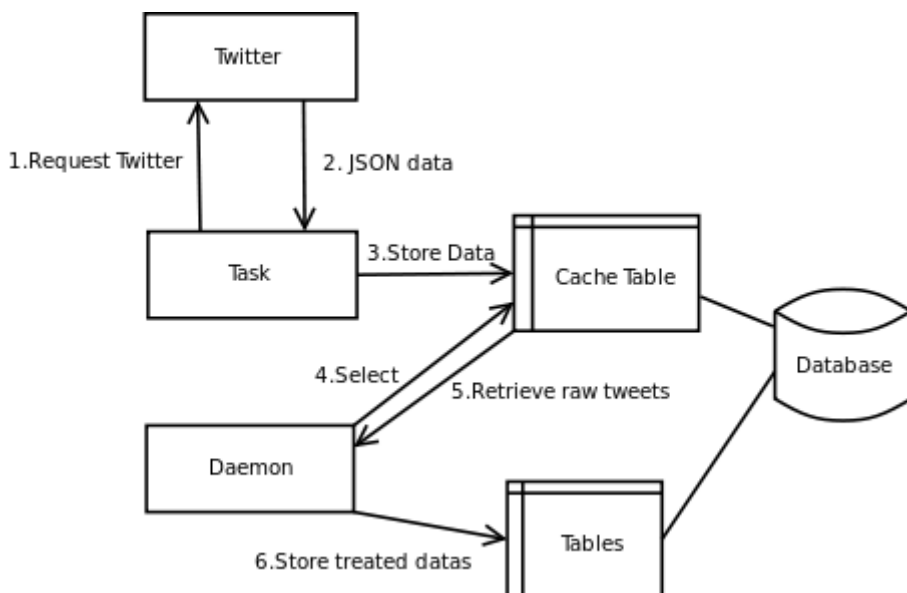


Figure 5.5: Data collect and process

A daemon is in charge of the data processing. This daemon is started by the cron and runs forever. This process retrieve raw tweets that have been collected in the tweet cache table. So these tweets are available for the extraction of interesting data which are then formatted accorded to database schema to be inserted in corresponding tables.

So daemon job consists of :

- Keeping only essentials information inside tweets (sender,content,date) and discard all other to reduced size to store.
- Analyze if a given tweet is a retweet. In such case the identity of the message sender and the one who retweeted it are retrieve and this information is used to feed the retweet table.
- Analyze the content if each tweets. If it contains mentions then this is register in the mention database. A combination of the hashtag #ff of #FF with mentions is also detected. In such case, based on the identity of sender and receiver a new row is added in the friday_follow table.

So as to clarify the way this is done lets take the example of a retweet. Assuming that one user A has retweeted a tweet send by one of his friends B. This retweet will be stored in the cache table by one of the cron tasks.

Once this is done, the daemon process will select this tweet from the cache table and analyze its data contents (JSON format returned by twitter). Those data contain a field *'retweeted_status'* which specify that this tweet is a retweet and another field *'created_at'* contains the datatime of this retweet creation . Apart from that the two other fields *'retweeted_status'* and *'user'* identify users B and A respectively. Knowing those information the daemon will recover the friendship link between A and B stored in the database and associate a new retweet to it.

5.4 Conclusion

The back-end implementation ensure the collect of data using twitter API. This collect is separate from the processing to avoid bottleneck problems and get more flexibility. On one side the retrieving is done by associating a cron table with a tasks queue. Moreover most critical tasks are optimized by using forks to address simultaneous requests to twitter. On the other side collected tweets are analysed by a daemon. It extracts and formats relevant information following database schema before they insertion in appropriate tables. Now the model is

ready and available for integration in a structure ensuring the interaction with each user. It is precisely the subject of the next chapter.

CHAPTER 6

Front-end

Thanks to the back-end the application gets the necessary data model which constitute its foundation. Now the implementation of the rest of the application is still lacking. This is the part that must interact with users and thus is called *Front-end*.

6.1 Symfony2

The prototype is implemented using the **Model View Controller (MVC)** Pattern . This is a oriented object pattern used to dissociate the representation of information from the interactions users have with it. It is composed of three distinct parts :

- A **model** contains the business logic. It provides an interface to manipulate and retrieve its state and it can send notifications of state changes.
- A **view** is a visualization of the state of the model. It is responsible only for rendering the UI elements
- A **controller** is responsible for interacting between the view and the model. It takes the user input to change the state of the model.

In order to build this pattern symfony2 framework is used. This powerful tool get the following architecture for each project :

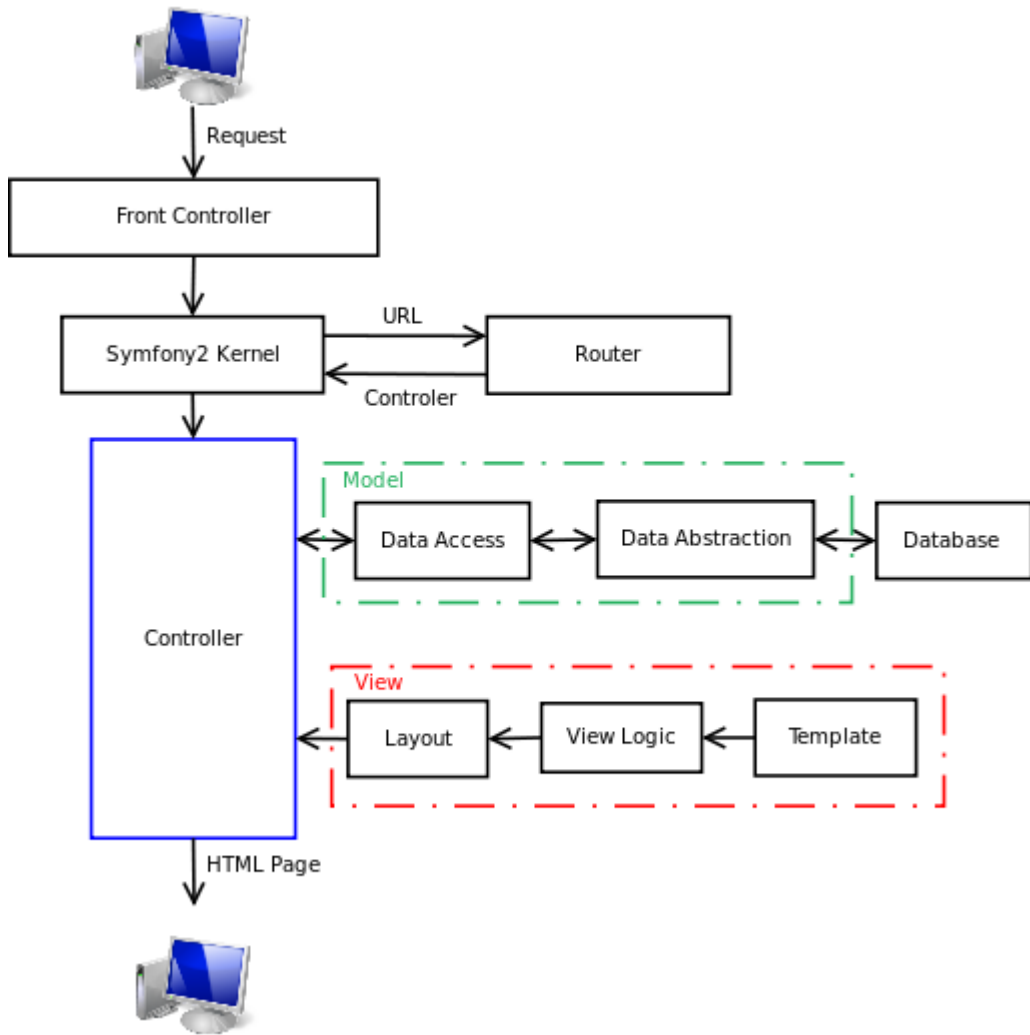


Figure 6.1: Symfony2 structure

The represented flow is the following :

- 1 - A visitor ask for a page
- 2 - The front controller catch the request, load the kernel and transmit the url.
- 3 - The kernel asks the router for the controller to execute the action corre-

sponding to the given URL.

4 - The given controller action is executed. The controller can interact with the model through the data access layer to retrieve

some data. Then those data are used by the view so as to build an HTML page.

5 - The controller return the entire html page to the visitor.

6.2 Controllers Structure

The different controllers integrated in the symfony2 architecture as presented in the previous chapter are grouped by bundle. A symfony2 bundle is a whole of files and directories implementing one or several functionalities. For the application the functions have been grouped in three bundles as represented below :

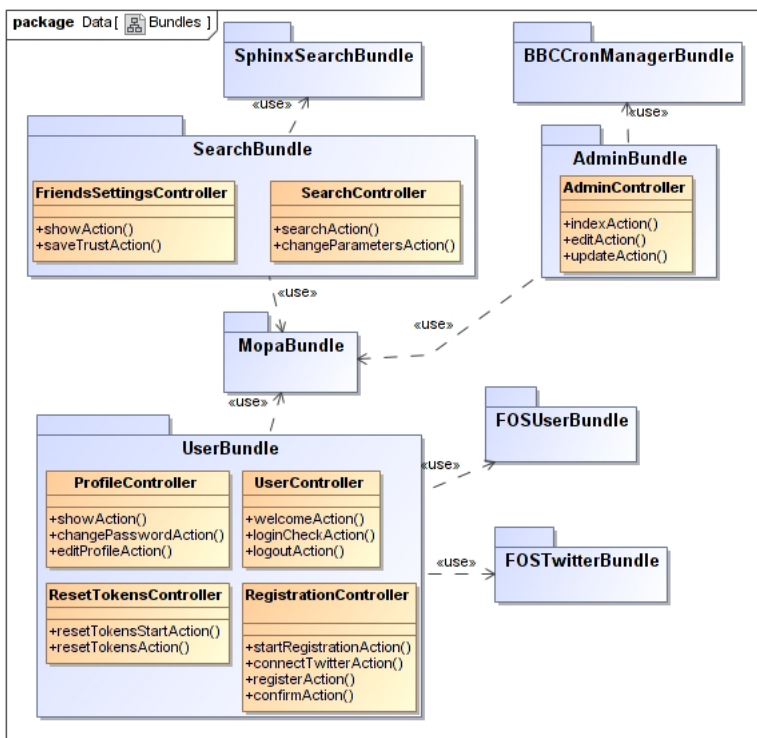


Figure 6.2: Application Bundles

The *AdminBundle* is in charge of actions performed by the administrator (see overview requirements).

The *SearchBundle* contains functions related to the search engine (search and modify searching parameters).

The *UserBundle* is in charge of all that is directly related to user such as registration process, login, modify profile...

Those three bundles need to use external bundles that have been developed by the symfony2 community. They are represented in the figure under names *SphinxSearchBundle*, *BBCronManagerBundle*, *MopaBundle*, *FOSUserBundle* and *FOSTwitterBundle*.

6.3 User Section

After the focus on the symfony2 structure the different functions and the graphical user interface are presented in the next sections. It was divided into two parts according to the requirements. The functionalities designed for the user and the ones for the administrator. This part focus on what is offered to the users.

6.3.1 User Registration

So as to use the application each user must first register using their twitter account and then fill a form to complete the operation. Those steps are managed by one Controller called *RegistrationController*. This controller will make the necessary verifications and implement the registration flow represented below :

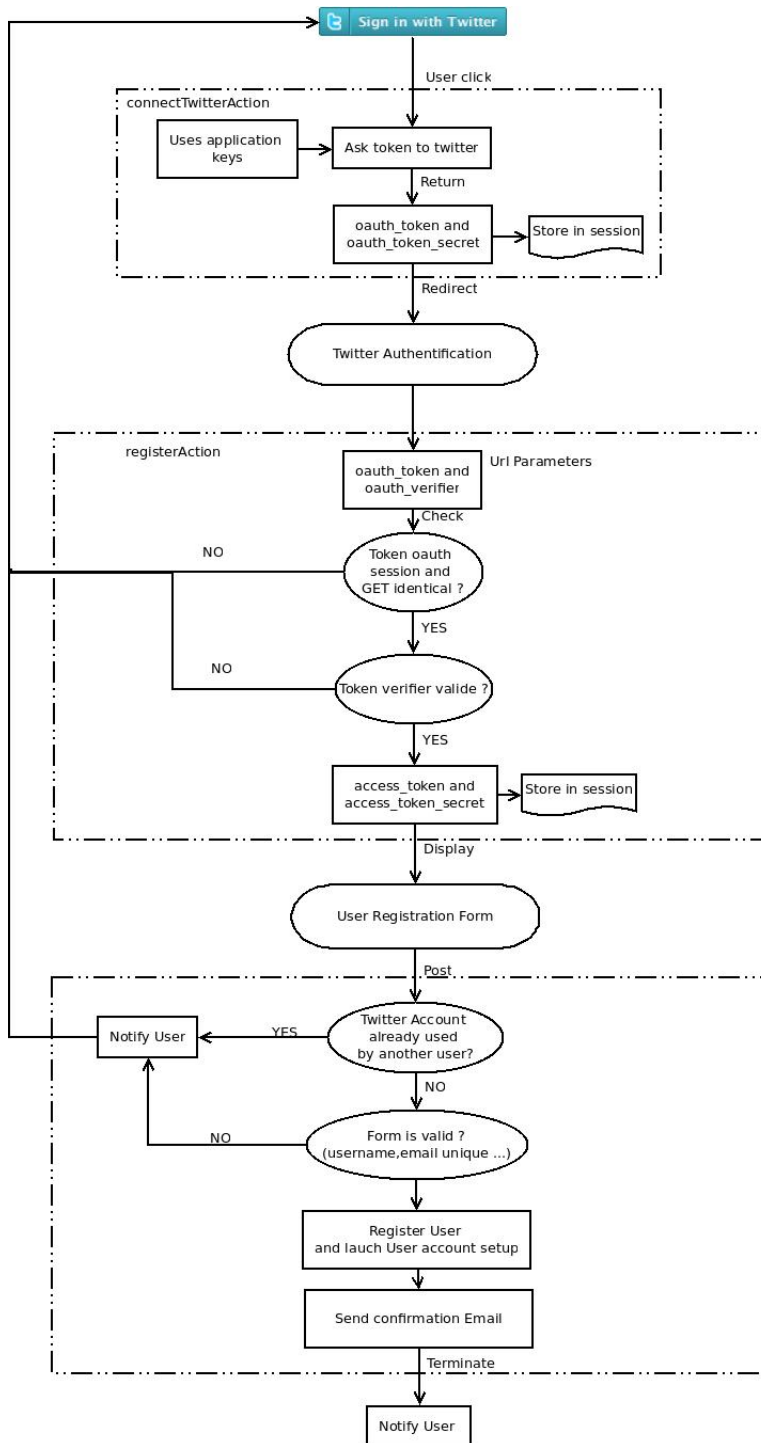


Figure 6.3: Authentication process

The represented flow is the following :

1. First the user visualize a page where he/she is asked to authenticate using his twitter account by clicking on the provided link.



Figure 6.4: User Registration start

2. On user click the controller then use the application keys stored in the *paramaters.ini* to query oauth tokens from twitter. Once those tokens are received, they are stored in session variables and used to build a redirection link. Therefore the user is redirected to authenticate with his twitter account.

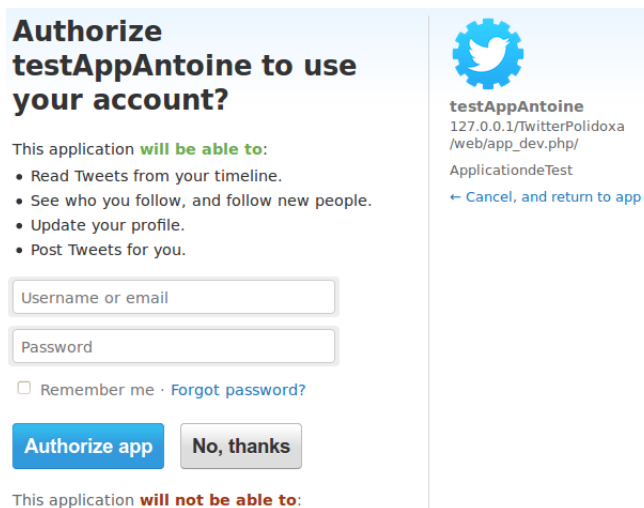
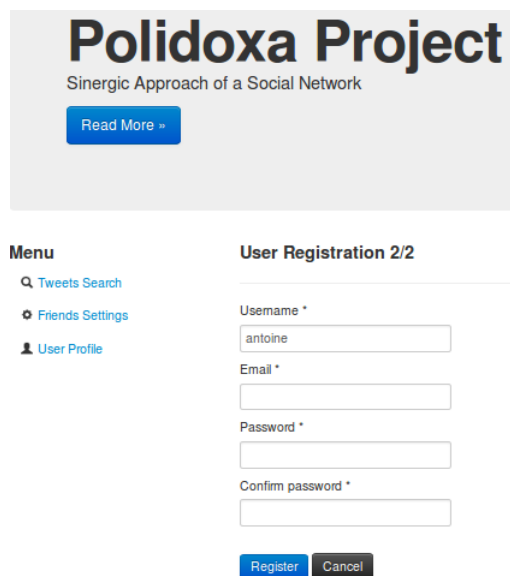


Figure 6.5: Twitter Authentication

3. After successful twitter authentication the oauth token contained in the

callback URL is compared to the one in session. This is done to check if the `oauth_token` in session is an old one. If it is not the case the second token contained in the URL (`oauth_verifier`) is used to get the two final access tokens. Those tokens are specific to each couple application-user and don't change within time so they are store in the database as user attributes.

4. The second part of the registration starts at this point when the user is redirected to the registration form. It is composed of four field, one for the username, one for the user email and the last two for the password. Once filled it is posted to the controller.



The screenshot shows the Polidoxa Project user registration form. At the top, there is a header for "Polidoxa Project" with the tagline "Sinergic Approach of a Social Network" and a "Read More" button. Below this is a "Menu" section with links for "Tweets Search", "Friends Settings", and "User Profile". The main content area is titled "User Registration 2/2" and contains four input fields: "Username *" (with the value "antoine"), "Email *", "Password *", and "Confirm password *". At the bottom of the form are two buttons: "Register" and "Cancel".

Figure 6.6: User Registration Form

5. It checks that the provided twitter account have not been used by another user and the correctness of the provided information (email,username and password). In case of success a validation email is send to the user with a link to validate is account.

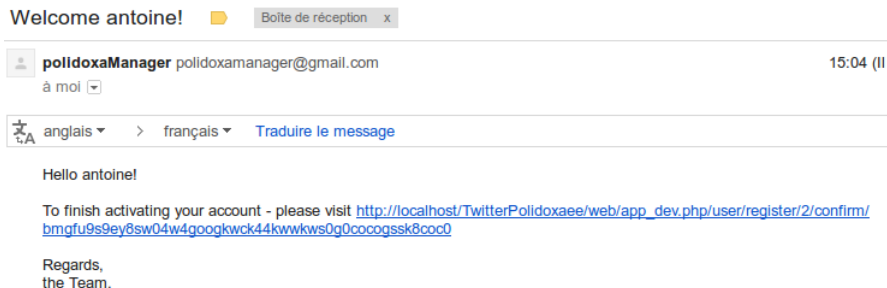
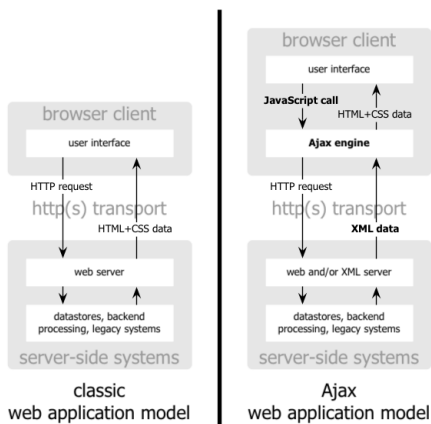


Figure 6.7: Confirmation Email

6. After validation the registration is completed and the user can log-in.

6.3.2 Friend Static Trust

One of the requirement is for each user to have a static parameter associated to each friend that can be modified and represents the trust granted by the user to this source. This is implemented by the *FriendSettingsController*. This is done using AJAX (Asynchronous JavaScript and XML) technology.



Unlike classic web model, with Ajax it is possible to execute some JavaScript that send a request to the server. The server compute it and return the result to the client. There is no need to reload the page to display the changes. This is what is used in the Friend Settings section.

As shown in the print screen of the Friend Settings section below each user has access to the list of his friends. For each of them a static trust is displayed, this trust is a percentage with a default value of 50 %. Using a slider user can easily enter a new value and save it. This value is in the scope 0-100%.

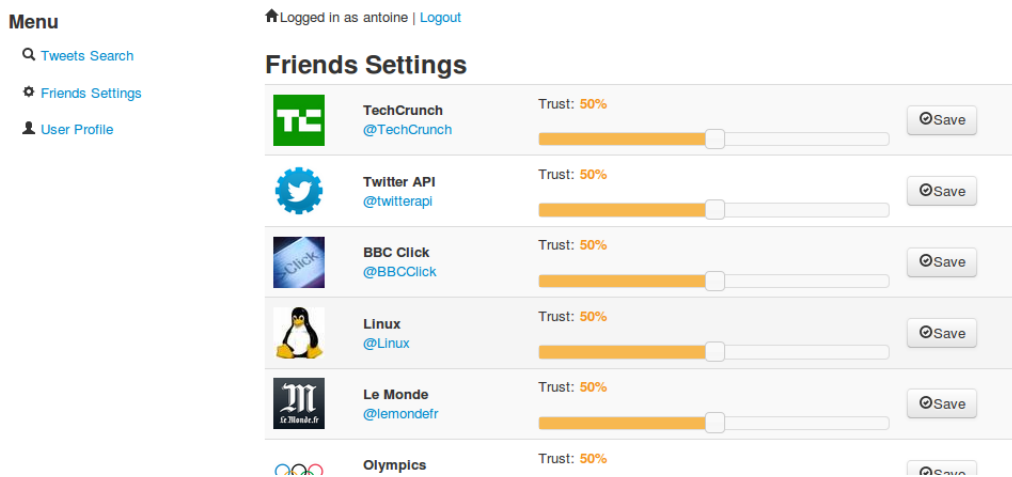


Figure 6.8: Static Trust Modification

6.3.3 User Profile

The next functionality offered to users is the possibility to consult his/her profile. The user profile section of the web application is displayed as following :



Figure 6.9: User Profile

This profile contains a bunch of useful information. Amongst them the user-name, email and password can be changed. It uses the same principle as for the static trust change (i.e AJAX requests). The network loading point gives information on which number of tweets are available for the user search. It matches the number of friend tweets stored in the database. The associated percentage is obtained by divided this value per the max number of tweets that can be stored in the database (the number of tweets to store per user can be modify by the administrator).

6.3.4 Tweets Search

In this section is described all that is related to the search section of the application.

6.3.4.1 Sphinx Data Indexing

According to the requirements the user can search into its friends tweets. As described in back-end section, the tweets for each friends are stored in the database.

However the databases are of type InnoDB which doesn't currently provide an efficient mean to search through text.

Thus to overcome this issue a search engine is used. Many exists but the two most popular are Sphinx and Lucene. Those two engine are quite similar in terms of performance but Sphinx unlike Lucene natively supports direct imports from MySQL. So the choice has been made in favor of Sphinx.

The running process is then the following : sphinx index the database table on which the search must be done and then a client can be used to get indexes corresponding to a search query. The indexed database table here is *tweet* which contains all users' tweets. So as to keep up to date a reindexing need to be done frequently. When indexing small data-sets, a full reindex can be used. But as size grows, so does the index, and with it the time it takes to index.

To work around this problem the delta indexing method is used. It consists in fact to introduce two indexes. One *main* index that is design to index all the tweets in database and a second index called *delta* containing indexes for only the tweets that changed since the last *main* index run. So a full indexing is done on the *main* index (containing most of tweets) only one time per day during the night. Beside that the *delta* index is rebuild frequently to keep synchronized with the database. Once that tweets are indexed and sphinx is correctly parameterized then the service is ready for use. The consuming process is the following :

1. The search controller sends query to the sphinx service using a php client.
2. The tweet ids corresponding to the result are returned.
3. The tweets are retrieve from the database using given ids.

This process is usually very fast (less than 1 second) which is much better than any query done on full text . The third point is necessary since sphinx doesn't store the text content so result return is only composed of tweet ids whereas the application need the whole content of tweets.

6.3.4.2 Search Principles

Thanks to sphinx it is possible to perform a keyword search function. All the tweets containing keywords entered by the user are returned. However according to the overall requirement they need to be ordered to be displayed to the user. It is here that the embedded trust is taken into account. Indeed so as to order the tweets two options are available :

- **Ordered By Static Trust :** As specified previously in the report each user assigned a static trust to his friends in the scope 0-100. When this option is selected the tweets are ordered according to this value first. The sub-ordering is done using sphinx ranking mode SPH_RANK_PROXIMITY_BM25 which combine proximity and BM25 ranking. This point is discussed longer below. In case those two values (trust and SPH_RANK_PROXIMITY_BM25 rank) are equal newest tweets are displayed at the top.
- **Ordered By Dynamic Trust :** A dynamic trust value is calculated following the formula explained in the next section. This value correspond to the static trust value corrected using user activities on the network. More clearly some user activities on the network such as retweet, favorite give boost to the static trust because they are indications on how close two user are. The sub-ordering modes are the same as for the static trust.

6.3.4.3 Dynamic Trust :

The dynamic trust is used to order result using information on the user network activities. These network activity information have been collected and stored in the database by the *back-end*. It corresponds in the database model to the tables retweets, mentions, favorites and fridayfollow.

The dynamic trust is calculated for each user's friend using the formula:

$$Dynamic_Trust = Static_Trust + \alpha_F * Nbr_favorites + \alpha_R * Nbr_retweets + \alpha_M * Nbr_mentions + \alpha_{FF} * Nbr_FridayFollows + \alpha_C * Results_count$$

The formula contains the following terms :

- **Static_Trust :** Value between 0 and 100 freely chosen by each user representing the trust granted to each friend.
- **Nbr_favorites :** Number of tweets sent by the friend and favored by the user. This number is multiply by a coefficient α_F chosen by the administrator.
- **Nbr_retweets :** Number of tweets sent by the friend and retweeted by the user. This number is multiply by a coefficient α_R chosen by the administrator.

- **Nbr_mentions** : Number of tweets sent by the user containing mentions referring this friend. This number is multiply by a coefficient α_M chosen by the administrator.
- **Nbr_FridayFollows** : Number of tweets sent by the user containing friday follows referring this friend. This number is multiply by a coefficient α_{FF} chosen by the administrator.
- **Results_count** : Number of tweets belonging to the friend matching the given search. This number is multiply by a coefficient α_C chosen by the administrator.

Amongst the previous variable four (*Nbr_favorites*, *Nbr_retweets*, *Nbr_mentions* and *Nbr_FridayFollows*) are related to the activity between the user performing the search and each of his/her friends. The aim is to give more importance to the people with whom user have more interaction and thus he/she is closer to. The underlying principle is considering that the closer you are to someone the more reliable source he/she is.

The last parameter (*Results_count*) is not specific to a 'person' but to a research. It correspond to the number of matching documents when a query is performed.

For instance assuming that user A have a friend B that is a fan of bikes and has sent many tweets containing #BMW and another friend C that has only few tweets containing this keyword. It will result on an extra trust granted to friend B proportional to the number of tweets containing #BMW. This information is not free and requires to perform two search execution. The first one is devoted to gives the *Results_count* for each friends by grouping results by sender.

Once this value has been collected the dynamic trust is calculated for the search request. If a trust value go beyond 100, values are rescaled to keep in a scope between 0 and 100.

So as to decrease the dynamic trust value, the network activity of each user are not taking into account if they are older than one year. So if you have less interaction with some of your friends they will lose trust in favor of others. In other word if you stop to interact with one of your friends he/she will not be regarded as a reliable source.

The different coefficients that appears in the formula can be freely modified by the administrator. The choice has been made to restrict this possibility to only this super user to avoid confusing user. Indeed it would be quite difficult to understand and adjust coefficient for lambda person.

Beside this dynamic trust there is two sub-ordering modes. They play a role in case two tweets have the same trust.

The first sub-ordering is made using Sphinx SPH_RANK_PROXIMITY_BM25 ranking mode. This mode is a combination of phrase proximity and BM_25 calculated as : $weight = doc_phrase_weight * 1000 + integer(doc_bm25 * 999)$. The first factor `doc_phrase_weight` is a number of keywords that occurred in the document in exactly the same order as they did in the query. Here is the example from the documentation :

- query = one two three, field = one and two three `field_phrase_weight = 2` (because 2-keyword long "two three" sub-phrase matched)
- query = one two three, field = one and two and three `field_phrase_weight = 1` (because single keywords matched but no sub-phrase did)
- query = one two three, field = nothing matches at all `field_phrase_weight = 0`

The second factor `doc_bm25` depends on frequencies of the matched keywords. Altogether it gives a quite precise indication on the tweet relevance according to the query performed. So it is used to sort tweet with the same trust. The second sub-ordering is by date. If some tweet result have both same trust and relevance then the most recent are displayed before.

The result of a search is displayed as below to the user :

The screenshot shows a search results page. At the top left is a 'Menu' with links for 'Tweets Search', 'Friends Settings', and 'User Profile'. At the top right, it says 'Logged in as antoine | Logout'. The main heading is 'Search Tweets' with a search bar containing 'Lille'. Below the search bar, it says '177 Tweet Found'. Three tweets are listed:




Profile	Text	Trust	Date
 Le Monde @lemondefr	Ligue 1 : Lille se contente d'un nul pour inaugurer son stade http://t.co/1QhyeEoX	Trust : 52.52%	17/08/2012 21:02:03
 Le Monde @lemondefr	Ligue 1 : Lille inaugure son grand stade face à Nancy http://t.co/zkzo1BOz	Trust : 52.52%	17/08/2012 15:33:33
 Centrale Lille @Centralelille	RT @DavidSimplotRyl: ICSCS'2012: 1st Int. Conf. Systems and Computer Science, #Lille, Aug 29-31, http://t.co/PP8SqaBt cc @CentraleLille ...	Trust : 51.73%	30/07/2012 18:21:08

Figure 6.10: Search Result

The number of results is given at the top just before the list of tweets. This list is composed of ordered tweets. For each of them the trust is given at the top right corner. So as to facilitate the loading, a JQuery infinite scroll mechanism is used. Only the first 50 results are loaded. If the user scroll down then the next 50 are added and son on until no more results are available.

Moreover search can be parametrized using the parameter search menu :

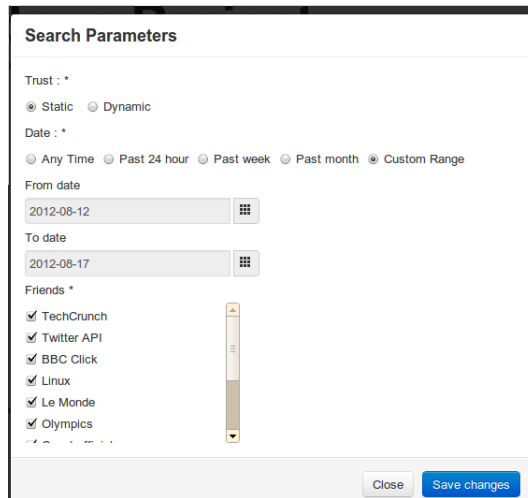


Figure 6.11: Search parameters

This menu addresses the requirement which imposes that the application users must be able to change some search parameters as they desire. Indeed one field allow to switch between static and dynamic trust to order tweets. The second option specify on which time range perform the search.

The last option gives the possibility to restrict the search to only some specific friends.

6.4 Administrator section

The application provides an administrative section that can be access by login as administrator using correct credentials. It is composed of two parts the first one lets the possibility for administrator to modify some application parameters :

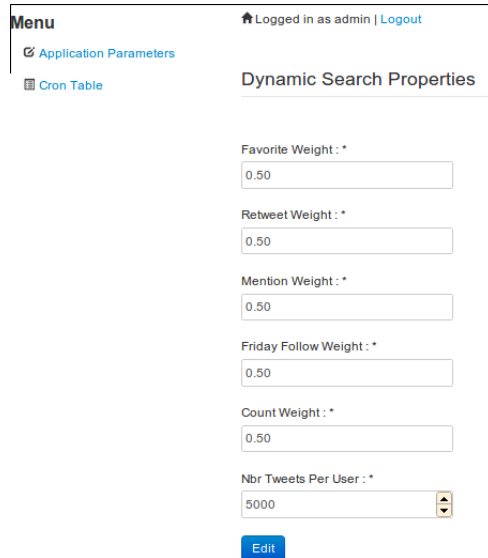


Figure 6.12: Application Parameters

The available parameters are the five coefficients used to calculate the dynamic trust. Moreover the number of tweets stored per user can be changed. The greater this number is the heavier is the database but the longer back in time a search will give results.

The other section displayed to the administrator is the cron manager. This manager is based on an external Bundle *BBCronManager*. It represents cron table as follow :

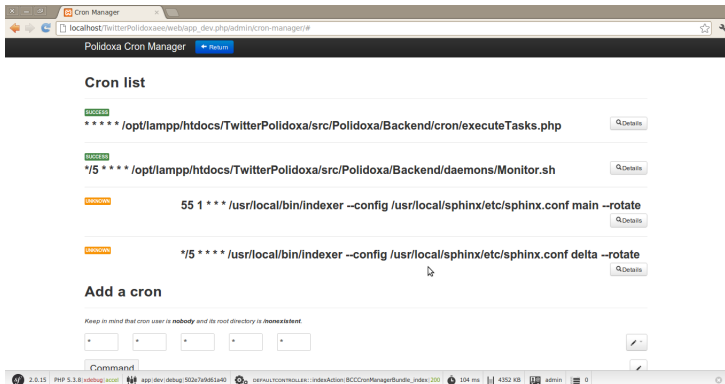


Figure 6.13: Cron Table UI

For each programmed task it is possible for the user to see task status. In case of error logs files can be read.

Cron list

SUCCESS
***** /opt/lampp/htdocs/TwitterPolidoxa/src/Polidoxa/Backend/cron/executeTasks.php [Details](#)

🕒 Last Run August 17, 2012 17:08

📝 **Comment** task queue execution

📄 **Log File** /home/antoine/logs/test.log [Log File](#)

📄 **Error File** /home/antoine/logs/error.log

🔍 **Raw cron**
***** /opt/lampp/htdocs/TwitterPolidoxa/src/Polidoxa/Backend/cron/executeTasks.php > /home/antoine/logs/test.log 2> /home/antoine/logs/error.log #task queue execution

[Edit](#) [Remove](#)

Figure 6.14: Cron Task details

6.5 Testing

6.5.1 Deployment

The application developed during this master project is a web application. Thus a php web server is required together with a mysql database. Moreover Symfony2 framework requires some specific characteristics to run :

- PHP version $\geq 5.3.2$
- Sqlite3, JSON, ctype extension must be activate
- The date.timezone parameter must be specified in php.ini

The application has been deployed on a virtual private server (vps). This has been chosen because the deployment on mutualized hosting is difficult due to the necessity to setup sphinx daemon which is most of the time not allowed on this kind of host. The characteristics are :

- Ubuntu Server 10.04 LTS "Lucid Lynx" - 64 bits
- Hard drive 50 Go
- CPU 1.5 Ghz

Symfony2 is delivered with a script to check the server compatibility (see Symfony2 documentation). After having install all the necessary extensions the database has to be generated. The generation process is automated thanks to the symfony2 console. By running the two commands *doctrine:database:create* and *doctrine:schema:update --force* the database and tables are created. This operation assume of course that suitable parameters for the given database have been previously filled in the symfony2 parameter file.

The next step is to install sphinx daemon. So the sources are download and compiled following instructions given on the website. Then the configuration file has to be replaced with the one associated to this project. Before it is done the sphinx daemon can be started. Regarding the cron setup the following operations are done :

Some task that should run periodically and forever are added in the queue : *DatabaseCheckOutTask* with 24 hours period, *RetrieveFVTask* with 2 hours period, *RetrieveRTTask* with 2 hour period, *RetrieveNTTask* with 5 hours period and *RetrieveHTTask* with 5 minutes period. The given periods have been chosen according to the expected data flow paying attention to keep below the limit

of 350 call per hour.

Then the cron table is created using the admin graphical interface. The table is composed of three tasks :

- one task which is executed each minute and is used to launch queue tasks.
- one task executed each 5 minutes is responsible to start and monitor (restart if crashed) the daemon.
- one task to index the delta sphinx index each 5 minutes
- one task to index the main sphinx index during the night (at 1.55 am)

The last step is to register a new application on twitter developer website and add generated tokens into symfony parameter file. Now that it is done the web application is ready. The administrator should periodically look at the log files so as to ensure that everything run properly.

6.5.2 Functional Testing

So as to test the application a twitter account was created :

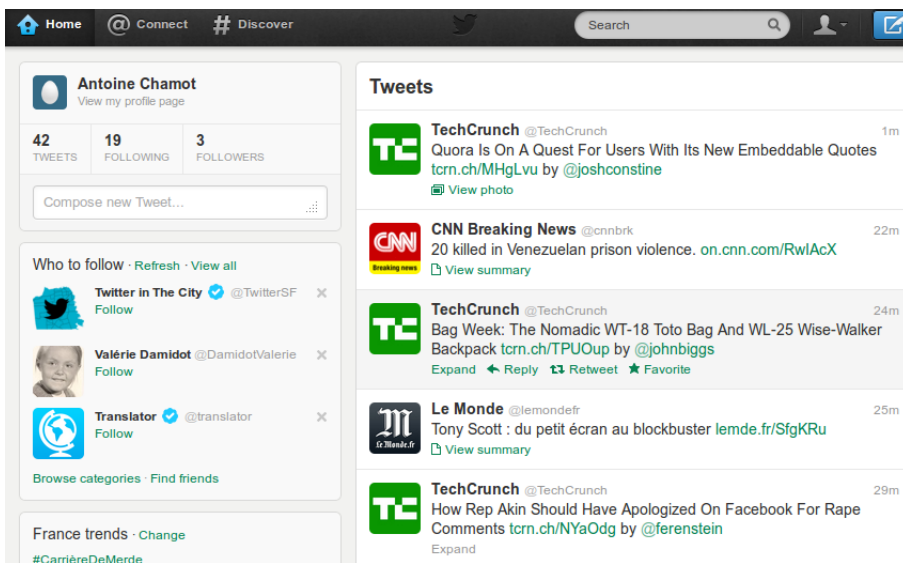


Figure 6.15: Twitter Account

This account follow 19 other accounts which are regarded as friends. Using this

account the user follow the registration procedure. According to the requirements the user his friends information must be register in the database. So as to check that it has been correctly done the different databases content are verify using php myadmin. The user database table contains indeed correct data :

	id	username	username_canonical	email	email_canonical	enabled	salt	password	last_login
<input type="checkbox"/>	36	antoine	antoine	chamot.antoine@gmail.com	chamot.antoine@gmail.com	1	m6ju9eq4a8owkg8kcs44xc08cgsc	AHs2pmCuQE /Htpt+IU+ozXUGQqqwH5SoX5+3O1 /2w6t6cp4Vv.	2012-08-19 14:50:00
<input type="checkbox"/>	37	NULL	NULL	NULL	NULL	0	NULL	NULL	NULL
<input type="checkbox"/>	38	NULL	NULL	NULL	NULL	0	NULL	NULL	NULL
<input type="checkbox"/>	39	NULL	NULL	NULL	NULL	0	NULL	NULL	NULL
<input type="checkbox"/>	40	NULL	NULL	NULL	NULL	0	NULL	NULL	NULL
<input type="checkbox"/>	41	NULL	NULL	NULL	NULL	0	NULL	NULL	NULL
<input type="checkbox"/>	42	NULL	NULL	NULL	NULL	0	NULL	NULL	NULL
<input type="checkbox"/>	43	NULL	NULL	NULL	NULL	0	NULL	NULL	NULL

Figure 6.16: User table

As it can be seen on the screen-shot all the users have the same fields but user and his friends are differentiate by their content. Indeed the friends have most of their field null except those related to the twitter accounts which are the twitter Ids, twitter usernames, twitter names and twitter profile image URLs. The authenticated users which are those who subscribed for the application (in this case Antoine Chamot) have extra information relative to the application account itself (username,email,password...). Altogether with those users the corresponding relationships have been created and store in the friendship table :

	id	user	friend	trust
<input type="checkbox"/>	42	36	41	50
<input type="checkbox"/>	43	36	44	50
<input type="checkbox"/>	44	36	37	50
<input type="checkbox"/>	45	36	43	50
<input type="checkbox"/>	46	36	46	50
<input type="checkbox"/>	47	36	40	50
<input type="checkbox"/>	48	36	38	50
<input type="checkbox"/>	49	36	39	50
<input type="checkbox"/>	50	36	45	50
<input type="checkbox"/>	51	36	42	50
<input type="checkbox"/>	52	36	47	50

Figure 6.17: FriendShip table

Those are the one sided relation between the user and its friends. Those relationships ids are the foreign keys for data in retweet, mentions, favorites and

friday follow tables as it can be seen here :

<input type="checkbox"/>					42	2012-04-30 14:04:58
<input type="checkbox"/>					42	2012-05-02 12:09:10
<input type="checkbox"/>					42	2012-05-02 20:25:15
<input type="checkbox"/>					42	2012-05-18 16:19:05
<input type="checkbox"/>					42	2012-07-13 13:20:39
<input type="checkbox"/>					42	2012-07-15 14:28:54
<input type="checkbox"/>					44	2012-05-02 10:31:18
<input type="checkbox"/>					46	2012-07-04 16:32:22

Figure 6.18: Retweet table

It has been verify that twitter data corresponding to retweets, favorites, mentions and friday follow have been correctly stored. Then together with that, first home timeline statuses are stored. After letting the application running for some hours all the friends has been pre-populated with their last 3200 tweets (API limit) and the database contains several tens of thousands tweets corresponding to all friends and stored in the table :

<input type="checkbox"/>					id	sender	tweet_id	tweet_content	created_at
<input type="checkbox"/>					19	41	234665742564466688	Outside Lands Wins The Internet -- And Mobile, Too...	2012-08-12 15:00:47
<input type="checkbox"/>					20	41	234680895741784064	Now in 20 Cities, Startup Grind Aims To Inspire Th...	2012-08-12 16:01:00
<input type="checkbox"/>					21	41	234696391962787840	Iterations: A New Era in Transportation Systems ht...	2012-08-12 17:02:34
<input type="checkbox"/>					22	41	234696639984590848	Gillmor Gang: Remote Wipe http://t.co/EbjFWDie by ...	2012-08-12 17:03:34
<input type="checkbox"/>					23	41	234705396739235840	Barnes And Noble Cuts Nook Tablet Prices As New Ki...	2012-08-12 17:38:21
<input type="checkbox"/>					24	41	234711240969158656	VP Pick, Paul Ryan, Has (Mostly) Been A Friend To ...	2012-08-12 18:01:35
<input type="checkbox"/>					25	41	234716866067697664	http://t.co/3ZtMEwqv Reaches Its \$500k Funding Goa...	2012-08-12 18:23:56
<input type="checkbox"/>					26	41	234726186826227713	Report: Google Could Face UK Tax Inquiry After Jus...	2012-08-12 19:00:58
<input type="checkbox"/>					27	41	234741210173034496	YC-Backed HiMom Helps Your Parents Keep Up With Yo...	2012-08-12 20:00:40
<input type="checkbox"/>					28	41	234749017236586496	How Google+ Punk'd The Oatmeal http://t.co/3CNVpj...	2012-08-12 20:31:41
<input type="checkbox"/>					29	41	234786484652212224	Why The Search For The Mystical Data Scientist Sho...	2012-08-12 23:00:34
<input type="checkbox"/>					30	41	234831783022182400	Answer Underground Aims To Be A Mobile-Focused Quo...	2012-08-13 02:00:34

Figure 6.19: Tweet table

Now that the account his setup correctly. The user change trust value granted to his friends.

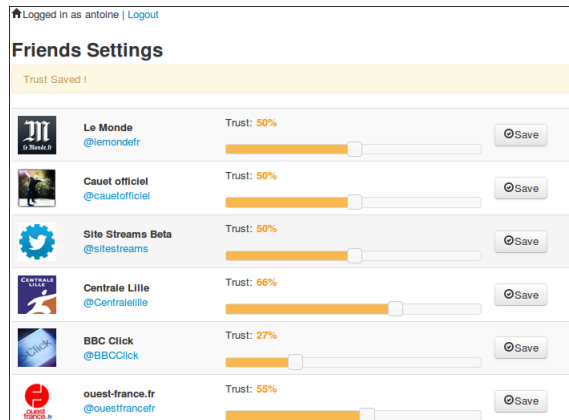


Figure 6.20: Static trust setup

Once this is done the application is ready for searching. Let's search for example new on 'apple' keyword. The following result appear :

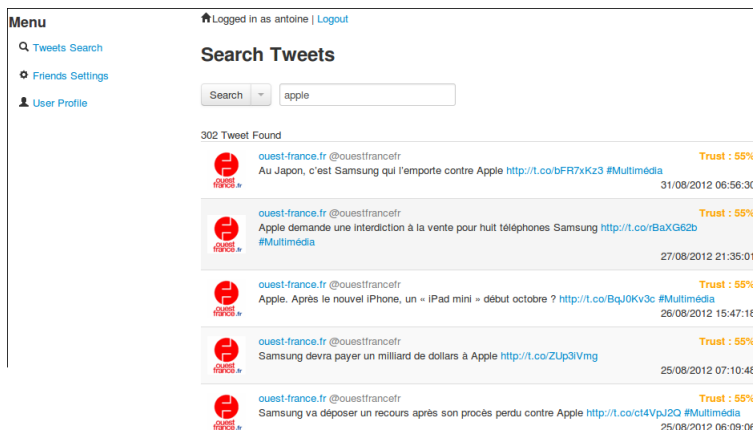


Figure 6.21: Result with static trust

As it can be seen on the screen-shot the results coming from the newspaper twitter account 'ouest-france' are the first displayed because the source is considered are the most reliable by the user itself (55% of trust). Indeed the default parameter for search is static so the result are ordered according to the static trust previously modified. But if the user switch to dynamic trust in the search parameters and make the same search again then the result is different :

The screenshot shows a Twitter search interface. On the left is a 'Menu' with links for 'Tweets Search', 'Friends Settings', and 'User Profile'. The main area is titled 'Search Tweets' and shows a search bar with the word 'apple'. Below the search bar, it indicates '302 Tweet Found'. The results are a list of tweets from the account 'TechCrunch @TechCrunch'. Each tweet entry includes the TechCrunch logo, the account name, the tweet text, a link to the original tweet, the user who retweeted it, and a 'Trust' score of 56.02%. The tweets are:

- Tweet 1: "Apple's Patent Win Is Bad For Us All <http://t.co/KJaiCylv> by @rezendi" (01/09/2012 13:02:40)
- Tweet 2: "The State Of Linux — How Even Apple Is Going Open Source <http://t.co/aDCvH4Oy> by @alexwilliams" (29/08/2012 21:01:27)
- Tweet 3: "Another Sign Of A New iPhone? Apple Will Pay You To Recycle Your 4S <http://t.co/a5N1ZYrk> by @ingridunden" (29/08/2012 09:36:00)
- Tweet 4: "NPD: Apple Customers Love The Genius Bar, Mostly Because Service Is Free <http://t.co/gEZErfSi> by @jordancrook" (28/08/2012 13:14:28)
- Tweet 5: (partially visible) "TechCrunch @TechCrunch" with a trust score of 56.02%

Figure 6.22: Result with dynamic trust

Now the first results that appear are the one belonging to 'TechCrunch' with a trust of 56.02 %. This is due to the fact that the twitter account associated to the user had more interaction with this account and thus has gained extra trust. So despite the fact that the user gets the impression that 'ouest-france' is more reliable than 'TechCrunch' its activities on the network show the contrary. This is all the interest of such parameter : correct user biased judgement.

Future Work

This section lay out possible directions of future work. It proposes improvements and recalls of issues left open.

During the implementation stage the choice has been made to use twitter rest API. This was not a deliberate choice but rather a solution in absentia of alternative. Indeed such a choice imposes limitations that has been workaround as better as possible but still exists. The alternative would be to use the streaming API. Unfortunately this streaming API is still at his beta version. This implies that is access is restricted and must be granted by twitter. The application for access has been made during this master, however regarding the time constraint it was not possible to get positive answer. Thus it is expected that sooner or later the user site stream API will be release. At that time the cron tasks would be advantageously replaced by permanent open stream. This has several major advantages, the application becomes real-time and users doesn't suffer from delays due to cron periods. Moreover high traffic can be handle easier since there is no more need to make parallel calls to twitter. By getting rid of limitations data collection could be extended to include other clues on the proximity between two users. For instance the direct messages exchanged by the user could to be included in the dynamic trust formula.

The application has been intentionally restricted in term of functionalities according to time given for this thesis. However one can imagine further devel-

opment so as to extend the prototype step by step. For instance the network analysis is now limited to the closest friends (i.e those with one degree separation). Taking into account impact on the database load the prototype could be extended to one or more degree further. For instance the friends of user friends can become also source of information with a trust degree reduced. The way the separation degree between users influence the decrease of the trust have to be investigate. A second step could be to integrate the possibility to access information on the whole web. For instance users would be allow to mark a news on the web that they consider as interesting and thus making it available to their relationships.

Another line of future work may be oriented to improve the formula used to calculate the dynamic trust. Indeed the formula was created in a test purpose to include various parameters. However it is not optimize to truly reflect the correlation between the user activity and the expected gain in term of trust. Determine the optimal coefficient values together with the appropriate formula would require a study on numerous twitter user account to show the correlation between their activity for instance favorite a tweets and the quality of information given by those sources. The developed application can play a role in this process. Indeed it is easy to modify the formula used by the application so as to verify hypothesis.

Conclusion

Information access has known a revolution with the emergence of the Internet. It offers an alternative to traditional media for accessing news. User can freely choose their agenda by searching subject they are interested in and select sources. Thus it guarantees diversity of information since Internet with numerous sources is less susceptible to suffer from control than traditional media. However this abundance requires to get rid of garbage and keep only relevant news. This process is done by search engine like Google by filtering step by step data. This guaranty good quality but is not sufficient to fully trust the receive documents. This lack of embedded trust is the main issue that Polidoxa intends to solve.

So as to visualize and test the idea of embedded trust an application prototype has been conceived implemented and deployed during this master. This application is based on one of the main tool that people favor when they search for news on Internet i.e. a social network. The one selected is twitter which has the advantage to be more than a simple social network but a efficient tool to share news. Following Polidoxa idea the prototype includes trust notion divided into two parts static and dynamic. The static trust is a value chosen by the user and can be modified at any time. It provides a way for user to influence the information retrieving. It is also a mean to give a feedback by adjusting value in case result is not considered satisfactory.

The dynamic trust is a parameter recalculate for each search action. It takes advantage of the activities that user share with his/her closest contact. Indeed favor or retweet a friend tweets is regarded as a sign of interest not only in the document itself but also in its source and thus gives it a trust boost. This mechanism corrects the user judgment that can sometimes be biased by making the trust evolved according to facts (retweets, mentions ets) and not only impression.

This prototype enable to puts forward news via tweets that are regarded as trustworthy. However this constitute only a first step to show the interest of embedded trust for news searching. Further investigations are required to find optimal way for trust to be calculated.

APPENDIX A

Source Code of the Prototype

The current appendix contains the source code of the prototype implemented.

Since it is very large to include in the appendices, the code of the implementation of the prototype can be found in the CD attached to the report.

It contains an compressed archive composed of two elements. They are the symfony2 project folder named 'TwitterPolidoxa' together with the configuration file for sphinx. This needs to be adapted according to the configuration of the web server on which the project is deployed.

Moreover sphinx sources can be download at the address <http://sphinxsearch.com/downloads/>.

Bibliography

- [1] Doctrine 2 documentation. <http://docs.doctrine-project.org/en/latest/index.html>. 2012.
- [2] Mysql reference manual. <http://dev.mysql.com/doc/>. 2012.
- [3] Symfony documentation book. http://symfony.com/pdf/Symfony_book_2.0.pdf?v=2. 2012.
- [4] Twitter api documentation. <https://dev.twitter.com/docs>. 2012.
- [5] Twitter bootstrap documentation. <http://twitter.github.com/bootstrap/>. 2012.
- [6] Alex Cheng and Mark Evans. An in-depth look inside the twitter world. <http://www.sysomos.com/insidetwitter/>. 2009.
- [7] M. Mazzara A. Marraffa L. Biselli L. Chiarabini. Approach of a social network and a search engine to offer trustworthy news. 2011.
- [8] jQuery Community Experts. *Query Cookbook: Solutions and Examples for jQuery Developers*. O'Reilly Media, December 2009.
- [9] Kevin Makice. *Twitter API: Up and Running*. O'Reilly Media, March 2009.
- [10] M. McCombs. Setting the agenda: the mass media and public opinion. 2004.
- [11] M. Mazzara A. Marraffa L. Biselli S. De Nicola. Social networks and collective intelligence for trustworthy news...and how polidoxa fixes the problem. 30 November 2011.

- [12] Haewoon Kwak Changhyun Lee Hosung Park and Sue Moon. What is twitter, a social network or a news media? pages 591–600, 2010.