

ENGAGEMENT AND INTEREST TWITTER TIMELINE IMPROVEMENT



Matthieu Nowicki
Master Thesis
School of Computer and Communications



Prof Denis Gillet,
Prof Tapan Parikh,
Lausanne, EPFL, 2014

Matthieu Nowicki
Avenue du 14 avril, 18
1020 Renens,
Switzerland

Nous nous découvrons vite des amis qui nous aident.
Nous méritons lentement ceux qui exigent d'être aidés.
— Antoine De Saint-Exupery

To my parents and my friends without who, nothing would have been possible.
Thanks to two universities which set me free and taught me the need to explore.
It was a pleasure to be encouraged by two Professors: Mr Gillet and Mr Parikh.
Thanks to many people who helped me during this semester: Thomas, Nick, Maité, Natalie,
James, Ally, William, Nima, Ingrid, Clement, Igor, Pierre-Stéphane, Alicia, Simon, Simone,
Julien, Gauthier ...
To INMTuning, for this 5 years of pure happiness

Do mojego dziadka

Abstract

Once a social networking service has as large a number of users as Facebook, Twitter or LinkedIn does, the new metric necessary to measure their progress is engagement. This value is not easy to measure, and does not have a unique formula; each service has its own based on the particular feature it has (retweet, favorite, etc.). Moreover, it is very difficult for companies to measure the time spent on individual posts like tweets. With this work, we would like to understand better the engagement with Twitter, and then try to increase the interest and the time spent on the platform.

To solve this problem, we created a new way to visualize tweets one-by-one. Using Twinder, the dedicated web application, it was possible to create an A/B test that recorded the time spent on tweets, and the decisions of users as to whether they liked each tweet. One unique calibration test A (a series of 50 tweets) allowed us to generate two tests B, which half of the candidates were doing. In test-like-B, the more a source of tweets was liked in test A, the more we displayed tweets from it. In test-time-B, the more time users spent on tweets coming from a source, the more we displayed tweets from it. We then used the time spent on tweets and the number of likes of the test A to generate tests B. After running this experiment and creating a survey, we realized that people in both configurations test-A/test-like-B and test-A/test-time-B are spending less time in the second series than the first one. However, in the first configuration we noticed a decrease of 24.20% of the time spent, compared to a decrease of only 2.51% in the second one. At the same, we measured the number of likes per series, which is a good indicator of interest. At the opposite, users who did test-like-B were way more interest in this series than users who did test-time-B. Indeed, the first configuration received 18.52% more likes than test A, and the second configuration received 9.14% less likes than test A. In parallel, the results of the survey allowed us to see that users who did test-like-B were more tired than the others, even if they seems more interested.

We did not succeed in increasing the engagement with a series of tweets, but we noticed that it is easy to increase the interest in a series. However, the interest cost a lot for the user, who is more tired, and spends less time in test-like-B. Finally, we found a way to keep the time engaged almost stable (-2.51%) despite the decrease of the interest (-9.14%).

Key words: Engagement, Interest, HCI, User Experience.

Contents

Abstract	i
List of figures	v
List of tables	vii
Introduction	1
Related works	3
0.1 EdgeRank	3
0.2 Twitter engagement	4
0.3 Twitter redesign	5
1 Motivations & Evolution	7
1.1 Motivations	7
1.2 Project evolution	8
2 Methodology	11
2.1 Big picture	11
2.2 Dataset	12
2.3 A/B Test	12
2.3.1 Test A : the calibration	12
2.3.2 Tests B	13
3 Experimental settings	17
3.1 The design	17
3.2 Data acquisition	18
4 Technical Implementation	21
4.1 Languages	21
4.1.1 Back-end	21
4.1.2 Front-end	22
4.1.3 Frameworks	22
4.2 Structure	23
4.2.1 General structure	23

Contents

4.2.2	Important files	24
4.2.3	Database	25
4.3	Parts of the code that is interesting	26
4.3.1	Authentication	26
4.3.2	Keyboard & AJAX call	27
4.3.3	Index.html	28
4.4	API	28
4.4.1	Wrapper	28
4.4.2	Request limit	29
4.5	Deployment	29
5	Results	31
5.1	Explanations	31
5.2	Twinder results	31
5.2.1	More likes and more time	31
5.2.2	Time and Like rate	32
5.2.3	Time spent on the 100 tweets	32
5.2.4	Time spent on tests B	33
5.2.5	Number of likes on 100 tweets	34
5.2.6	Number of likes in tests B	35
5.3	Survey	36
5.3.1	General questions	36
5.3.2	Variation between test A and B	37
6	Discussion	39
	Conclusion	43
	Annexe	45
	Bibliography	49

List of Figures

1	News Feed Optimization formula	4
2	Twitter engagement rate	4
3	Level of retweet	5
4	Twitter new design	6
1.1	Time spent of Social Media	7
1.2	Social Media engagement	8
2.1	A/B test design	15
3.1	Tinder design	18
4.1	MVC model	23
4.2	Connection with Twinder	24
4.3	MVC model	25
5.1	Average time on the 100 tweets	33
5.2	Time engagement in tests B	34
5.3	Average like on the 100 tweets	35
5.4	Number of likes in tests B	36
6.1	Project Diagram	48

List of Tables

1.1	Social Media comparison	9
5.1	More time & likes	32
5.2	rate time & likes	32
5.3	Survey - variation	37
5.4	Survey - number of tweets	37

Introduction

Since the sudden upsurge of Social Networks at the beginning of the 2000's, the time spent on social media has grown considerably. Today, they have a worldwide impact and influence many fields, both social and economic. However, social networks have to overcome the restrictions that come from their users, the web or social medias themselves. For example, Facebook admitted last year that they had some difficulty with engaging youth (ages 10-16) [1], which is directly linked to the large number of virtual communities. There are more than twenty social media platforms/social networking sites with a total of over 100 million users. As a result, there is a vast amount of information available to the public public that does not remain in the hands of one single network. These consequences have a direct impact on the confidence that investors have in companies like Twitter or Facebook. Despite its 500 million users, the microblogging service saw its stock price drop more than 23% [2] in February due to low user engagement. This new metric is not as easy to measure as the number of users and can have different aspects depending on the media [3].

Social media companies are now focusing on increasing user engagement and the time spent on their sites. In order to retain users, Facebook conducted research in the field of sociology linked to big data analysis. This research resulted in the EdgeRank algorithm being launched in 2010, whose main goal was to avoid overwhelming users with the News Feed. More recently, Twitter launched in its new design in April 2014. The design had good reception because it puts the user back in the center of the application. This strategy reassured many investors who saw this move as a desire to increase website engagement.

In this project, we simplified the engagement of the user by his or her engagement time. We created an application that shows tweets one by one and measures the time spent on each tweet, as well as the interest or not for it. In order to compare the two factors, time and interest, we performed two A/B test on different users. Our two independent variables are the number of tweets liked and the time spent on tweets during the test A. The first one is recorded by a binary classification of the tweets shown one by one. The second one is the time that the user take to make the previous decision. This design allows use to measure to dependent variables that are the interest and the time engagement.

Firstly, the paper discusses the studies related to this field and explains the relevance of this topic. Second, the paper describes the protocol that was followed and the technical implementation done in Python. Third, the method of data acquisition is described. Finally, the data is analysed and discussed, allowing us to arrive at our conclusion.

Related works

This project is at the intersection of Social media, overload of information and User Experience is relatively new and evolves quickly. As mentioned in the introduction, social networks appeared at the beginning of the 2000's and continue to be in perpetual movements. We can thus observe the transformation of the Facebook page since its creation or the one of the news feed. The criteria of immutability is not an benefit in this area and it would be interesting to see if the failure of MySpace for example is linked to it. We will give an overview of three related articles.

Firstly, we will talk about EdgeRank [4], the algorithm which sort the Facebook News Feed. Secondly, we will deep dive into the engagement on Twitter depending of the content of the tweets. Finally, we will recall the redesign of Twitter and its comparison with the Facebook page.

0.1 EdgeRank

In 2006, Facebook launched the News Feed, defined as: "News Feed is a constantly updating list of stories from people and Pages that you follow on Facebook. News Feed stories include status updates, photos, videos, links, app activity and likes". This new service is a success, although Facebook noticed that if it displayed all the news, the user has a good chance of being overwhelmed by the information. At the beginning, the News Feed ranking was determined by step-by-step fine-tuning or "turning knobs" [5], and it quickly evolved into a serious project called EdgeRank [4]. It was fired by Facebook in order to diminish the overload of information that users felt looking at their Timeline. Then, an algorithm in charge of the News Feed Optimisation (NFO) was invented. As you can see on Figure 1, the NFO takes three parameters into consideration: affinity, time and weight. With this method, an object has more importance and is more likely to appear in your News Feed if your friends have been interacting with it recently, such as "likes" or "comments".

Today, the company is using a more complex ranking algorithm based on machine learning. The News Feed is influenced by 100 000 individual weights that are customised by user fine-tuning. Then, when you flag a person as "close friend" or choose to "get notifications" from a particular person, the new algorithm adds this parameters to your profile.

After having researched what other companies are doing to struggle against the overload of

6. NFO: News Feed Optimization
EdgeRank

$$\sum_{\text{edges } e} u_e w_e d_e$$

u_e - affinity score between viewing user and edge creator
 w_e - weight for this edge type (create, comment, like, tag, etc.)
 d_e - time decay factor based on how long ago the edge was created

Figure 1: NFO was designed to reduce the overload of information in Facebook NewsFeed. The algorithm took three parameters: affinity, time and weight. More information are available at <http://www.whatisedgerank.com/>. Source: [6]

information, we decide to focus our interest on Twitter, especially what drives the engagement on this platform.

0.2 Twitter engagement

For a long time, Facebook has optimised its News Feed to spare its users from being overwhelmed by the vast amount of content that they receive. Research on Twitter is slightly different because the content is public and relatively controlled, seeing as a tweet is a message of less than 140 characters. The user can attach a photo, a video and hashtag. Thanks to this study [7], we have a better understanding of what facilitates engagement on twitter. The formula used to measure the rate is giving below Figure 2.

Engagement Rate = $\frac{\text{\# of Replies + Retweets}}{\text{\# of Followers}} \times 100$

Figure 2: The engagement rate is depending on the platform and the functionalities available. It is not an easy metrics to measure and has not a single formula. Find more information about it source [8]

This study shows that people do not engage equally with every Tweet. Indeed, adding extra content to the 140 limited characters increases considerably the engagement of the users. The Figure 3, illustrates the fact that photos, videos and links are influential factors in increasing the number of retweets.

After considering the content of the tweets and the fact that images are increasing the number of retweets, we discussed the design change that Twitter adopted in April 2014.

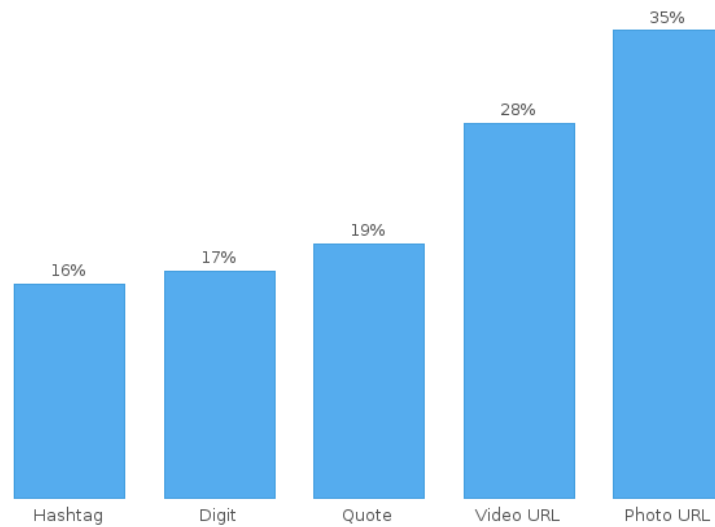


Figure 3: Engagement of people depending on the content added to the tweets, got from this source [7]

0.3 Twitter redesign

In April 2014, Twitter launched its new design, instilling confidence in their investors. This modification, as shown in figure 4, is modeled directly after the Facebook design. With this adjustment, Twitter aimed to cut off the overload of information [9], taking the user back to the center of the platform. As with Facebook, users can now enjoy a larger profile photo, customize their header, and show off his best Tweets.

Moreover, Twitter has developed some features to improve the content of the News Feed as it is mentioned on its blog [10]:

- **Best Tweets:** Tweets that have received more engagement will appear slightly larger, so your best content is easy to find.
- **Pinned Tweet:** Pin one of your Tweets to the top of your page, so it's easy for your followers to see what you're all about.
- **Filtered Tweets:** Now you can choose which timeline to view when checking out other profiles.

Engagement is a big issue for Social media companies, though its unclear definition makes it difficult to measure. We will try here to measure the difference between interest and engagement time and then find a way to increase the time spent on the Twitter news feed. In the next chapter, we will talk about the reasons why we chose this topic and how we reached it.

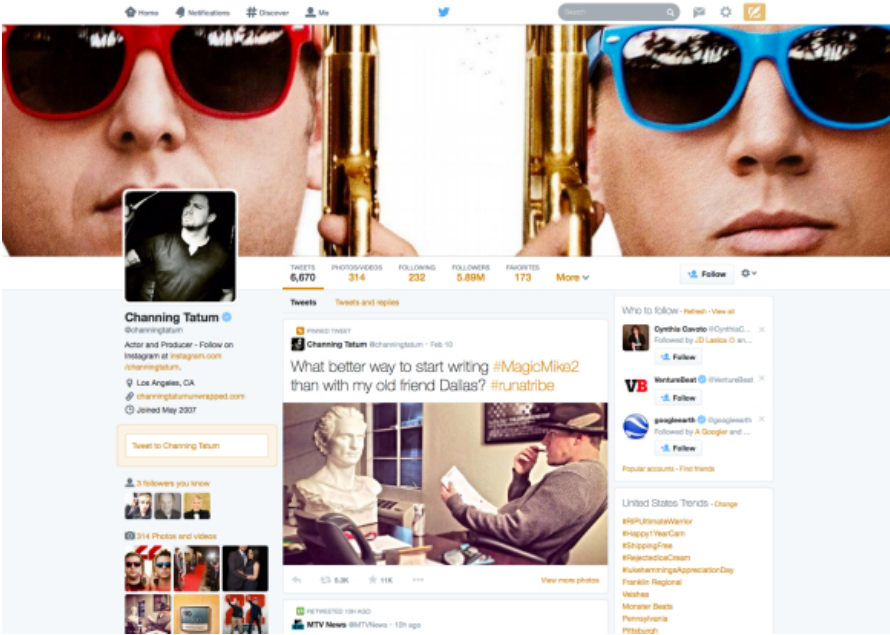


Figure 4: New design of a Twitter page. The profile picture and the Timeline remembers the Facebook design. The user is the central element of the page. Source: [11]

1 Motivations & Evolution

After talking about related works, we are going to list our motivations and then explain you the evolution of this project.

1.1 Motivations

Social media has an important role in the real world and has an impact that was probably under-estimated a couple of years ago. Historical events like the arab spring or the election in Cambodia [13] are two examples that show how social media can be a powerful force for change when compared to other media. These events lead to studies about human behavior and engagement on these platforms.

For users, one of the main goals is to increase the time efficiency on social media. Today, you can see on Figure 1.1 that 16 minutes per hour, or 27% of each hour, spent online is on social media. There are many methods used to try to make users' time on social media more efficient. As we saw with EdgeRank, the filtering of the information is one of the main ones.



Figure 1.1: This clock shows the distribution of an hour online spent by US people. 16 min are on social network and forum. Source: [14]

Recently, Twitter had to deal with an engagement problem. As we can see on Figure 1.2, despite its 550 million users, Twitter has an engagement lower than Facebook or Instagram: this weakness led to a drop in its stock price. However, "engagement" has no scientific definition and by consequence, there is no unique solution to this problem. By considering a new design which reduce the overload of information, it will be possible to analyse more details relating to the problem of engagement. By doing this it will be possible to improve user time efficiency and twitter engagement issues.

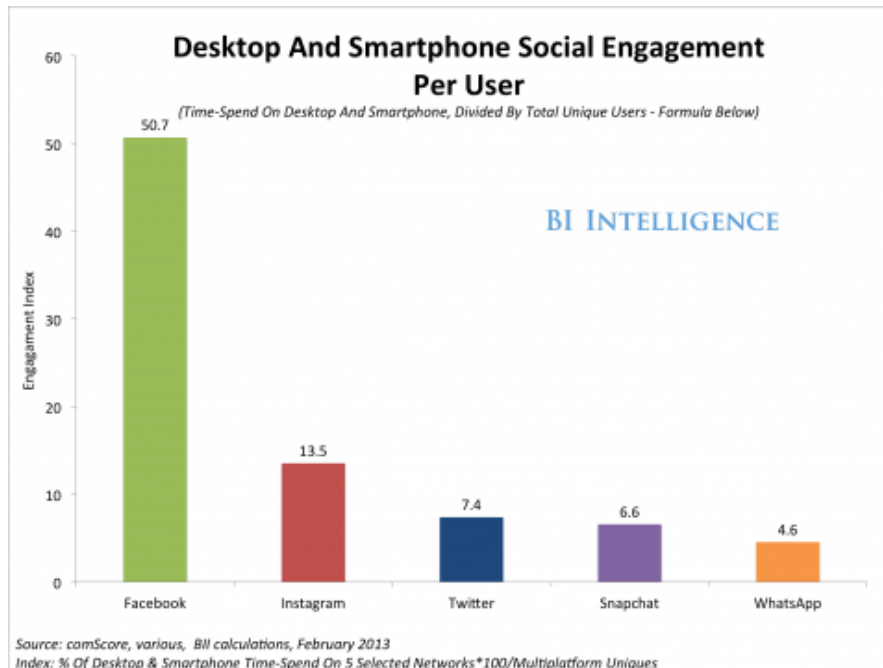


Figure 1.2: The graph gives us the social engagement per website. Facebook is from far the first one, it attracts roughly seven time more than Twitter. Source: [15]

1.2 Project evolution

Information overload is a large topic and many aspects could have been approached. By viewing Table 1.1 can understand better why it was decided to focus the research on Twitter. Our three main criteria are: privacy, content and technique. Firstly, with Gmail, privacy and content were being dealt with. Due to the email variate and the personal characters of email, it was difficult to establish a protocol in order to improve the filtering of the emails. Secondly, we started to develop, with a similar design, applications based on different social media like Reddit or Tumblr. Finally, Tinder and Twitter were found to be of particular interest, due to the design of the Tinder app and the 140 character limit of Twitter. For these two reasons and Twitter's simple API The idea of Twinder was arrived upon. You can find in annex the diagram 6.1 showing the evolution of the project through three aspects: technique, apps implementation and research.

1.2. Project evolution




















	Gmail	Feedly	Reddit	Tumblr	Twitter	Facebook
Privacy						
Content			 *	 *	  **	
API						

Table 1.1: In order to pick the best platform, we did a revue of 6 websites based on three criteria which are: privacy, content and API.

You can find the explanation below:

Red: problem for implementing the experiment.

Green: ok for implementing the experiment.

* : content is not constant.

** : 140 char with image or not.

2 Methodology

In this section, we describe our methodology for estimating behaviors of users in social media sites from observational data. According to the literature review, we noticed that interest and engagement are two crucial metrics for social media. These two values are dependent on many parameters, which are not easy to determine. The user and the platform (Facebook, Twitter...) are the two main sources of variation. From the user perspective, mood, stress and age [1] are factors modifying the two metrics. From the service side, the factors are well known from the companies, as many engineers focus on developing better solutions for the user. These solutions take into consideration the content, display, meta-data and are today more accurate due to Big Data.(ie: New EdgeRank version is taking in consideration more than 100.000 weight [5]).

Taking into consideration the three main reasons previously mentioned - privacy, content, and technique - we decided to base our experiment on Twitter. This way, the content is normalized and the user is familiar with it because we embed the tweet: the design of each tweet is similar to the one on Twitter. In order to add fun, we picked the Tinder concept of binary classifications of girls and applied it to the tweets. This approach helped us to design a protocol based on tweets classification, which differentiates the user interest and engagement. First of all, we give an overview of the methodology, explaining the test that we ran and the metrics used. Then, we introduce our Twitter dataset, composed of embedded tweets from user timelines. Finally, we delve into the process itself.

2.1 Big picture

In order to measure our users' behaviors, we decided to run A/B testing with two different B tests. This allows us to compare our dependent variables which are the number of likes and the time spent on a tweet. To be sure that we only keep in consideration the two dependent variables, a simple graphical design is used. This design is exactly like a PowerPoint document, where the content of each slide is an embedded tweet. The user has to decide for each slide, if the user likes or dislikes a tweet by pressing one of two different keys on the keyboard. Based on the design of Figure 2.1, we create an A/B test with one initial experiment called

"Calibration" (test A) and two others recording input the time (test-time-B) and the numbers of likes (test-like-B). Because of the unique design, we will explain how we select the tweets for the calibration part and the two other tests.

2.2 Dataset

As mentioned before, we decided to do a binary classification (like / dislike) of the tweets coming from the user's timeline. Because we use tweets from the timeline, it is assumed that users are interested by the content to a certain extent: they follow the writer of the tweet. In order to reduce external factors, we limit the timeline to a group of 10 people the person follows, called "friends". We select 10 friends randomly who published at least 25 tweets. Thanks to this data, we create a simple timeline that we suppose is similar to the real one. When a user logs in to Twinder, we retrieve in total 250 tweets: 25 tweets coming from 10 different friends. In order to keep the timeline chronological, we select the 25 most recent tweets from each user.

In the motivation chapter, we said in the table 1.1 that the content of twitter is standardized with a maximum of 140 characters. However, the content can take several forms, like photos, videos, text and it starts to become more difficult to differentiate each case. We finally consider them similar to a pure text because the majority of the tweets with images contain text. After having considered the content of the tweets, the next problem that we faced was the layout of each tweet. Indeed, after displaying only the text of the tweets, we realized that the embed layout was something important for the user. It allow the person to contextualize the information and link it to twitter. (img difference of embed not).

The methodology of the experiment is based on the dataset assumption that we enumerated previously. The A/B test will be explained, it is composed of the test A (calibration) and the test-time-B or test-like-B.

2.3 A/B Test

In this section, we explain the methodology of the experiment which is designed like an A/B test. We describe the two steps of the test and try to show why we chose this method to evaluate the two different behaviours of the user, which are interest and engagement.

2.3.1 Test A : the calibration

This step is common to every candidate, it is the base of our data collection and it gives us the input for the next step of the experiment. From the 250 tweets coming from 10 differents friends, we create a subset of 50 tweets. To compose this subset, we take 5 tweets randomly from each friend. By doing it this way, for the calibration test we have: 5 tweets times 10 friends which is a total of 50 tweets. Thanks to this design, the candidate is going to classify 50 tweets and then allow us to rank the interest and the engagement of the user for each one of these 10

friends.

Twinder, the slides application, saves each decision of the user in an object. These logs are used afterwards to define the next distribution of tweets. Mainly, the object stores: a timestamp, the decision (like / dislike), the tweet_id and the friend_id. This structure give us two important pieces of data: the time and the decision. After the test A, we can already display two graphs. The first one is the sum of the time spent on tweets of a particular friend. Despite a tweet having a maximum length of 140 characters, we decided to normalized the time spent on a tweet per its number of characters.

The second one is the number of likes that a friend received. As we display 5 tweets per friends, the number of likes is between 0 and 5.

Thanks to these two different parameters, we are going to generate a series of 50 other tweets with two different distributions. In order to measure the user's behaviors, the distribution of the 50 tweets is going to depend on the time for test test-time-B and the number of likes for test test-like-B.

2.3.2 Tests B

As we mentioned previously, test B depends on two different parameters: the time spent on the tweets and the number of likes. In this way, the same number of users is going to participate in test A / test-timeB and A / test-like-B. We are going to describe the two implementations and explain how they are measuring the engagement and the interest.

2.3.2.1 Test-time-B

After the calibration series is completed by the user, we can obtain from the data: the number of likes per friend and the time that a user spent on a specific friend. For the test test-time-B we chose to use the time and then measure the engagement. As we saw in the reviewed literature, it is impossible to give a unique definition of engagement. It is driven by the action that the user can do in the platform: click, scroll, like, retweet. As we cannot identify precisely what the impacts of each action are, we decided to only take engaged time into consideration. In this way, the more time a user is spending on a friend, the more engaged he is engaged by him no, matter what he decided. The interest of the user does not have any importance.

As was explained previously, we are going to generate the second series of 50 tweets based on the engaged time per friend. We are going to rank the friends as a function of the engaged time, and attribute more importance to the longest time. Indeed, our goal is to increase the engaged time.

To generate this vector of 50 tweets, first , we retrieve from the database, the 25 tweets per friend and remove the 5 tweets used in the test A. Then we use a linear function which takes the time as input, and an integer between 1 and 20 is the output. This integer is the number of tweets that we have to take for a given friend. In this way, we are going to take 20 tweets from

the friend with the maximum time spent and only 1 from the friend with the minimum time. The tweets are always selected randomly.

Once we have taken the tweets proportionally, we obtain a subset of more than 50 tweets coming from the 10 friends. Randomly, we will take 50 tweets from this subset, this method leaves the possibility of every friend appearing in the next series. After this, we repeat the classification experiment like in test A and take the logs of the decision.

This method is designed to increase the engaged time, we only take into consideration the time spent on friends in series A when creating the next subset of 50 tweets. Doing this, we suppose that the time spent on the second series is going to increase. While this method is supposed to increase the engaged time, the second method considers the interest of the users.

2.3.2.2 Test-Like-B

While test test-time-B focuses on the engagement, test test-like-B only considers the interest of the user. The interest of users is like the engagement, something not easy to measure, especially if we consider the two main ways which we can show it: retweet and mark as favorite. Thanks to the simple design of our application, we simplify the concept and allow the user to only choose if he likes or dislikes it. Of course, it is simplistic to measure the interest in a binary way but if we do the parallel with Tinder, it creates a good approximation as the user has to make a decision to pass to the next tweet.

Exactly like the previous test test-time-B, we are going to select a subset of 50 tweets based on the number of likes received per user. Then, we use the same linear function except the which take in input the number of tweets liked per friend an output a number between 1 and 20. If a user likes the 5 tweets of a given friend in test A, then we take 20 tweets from that friend in subset. Every time, we randomly select the tweet for a given friend. After applying the same process for each friend, we take 50 random tweets from the subset, which become our input for test test-like-B. The user has to pass the classification test again, and we record the logs of the new series.

The test-time-B and test-like-B tests were designed to isolate two different metrics: time engaged and interest. We submitted test A to every candidate and had half of the candidates do test test-time-B and the other half, test test-like-B. In order to better describe the experiment, the next chapter explains the experimental settings.

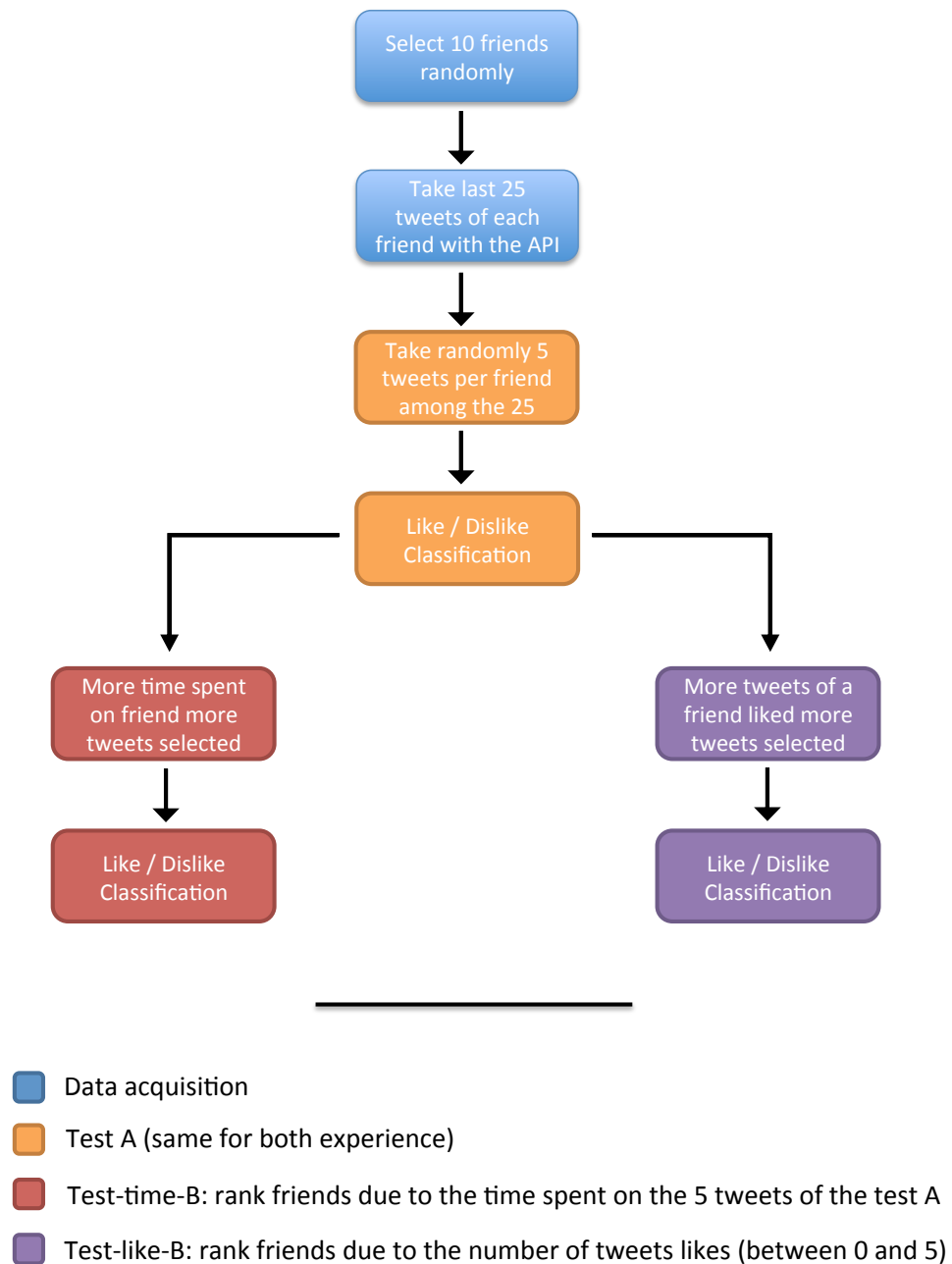


Figure 2.1: The diagram shows the design of the A/B test. First, each user took the unique test A. After a short break of one minuit, half of the users did test-time-B. The rest of them did test-like-B.

3 Experimental settings

The different research that we analyzed and the purpose of our research drive us to focus our interest on the design of the application. As we mentioned earlier, Twitter changed its design in April in order to increase the engagement of its users. We noticed that the design was inspired by the Facebook one. We chose to simplify the design of the application and then reduce the overload of information as much as possible in order to avoid background noise, which would be difficult to measure. For this, we modeled after Tinder, an application where the user classifies females and males according to their physical appearance. We kept this classification workflow and improved the look and feel in order to easily identify our two metrics: engaged time and interest.

This section will be separated into two parts, where we will present and explain our design in one section and discuss the data acquisition in the other.

3.1 The design

Our protocol described last chapter did not include the design of the application. It gave us the metrics of engaged time and interest. The general workflow of the application is dictated by the A/B test, the design of the apps is the only remaining task that we need to develop. In order to collect data and measure user behaviours without noise, we start thinking as simple as possible. The two aspects of Twinder are to visualise and classify tweets exactly like Tinder. As you can see on picture 3.1, the priority of the mobile application is the picture display. Two buttons are at the bottom of the page in order to classify the individual. When the user clicks on one of the buttons, the next profile appears and so on. Tinder is composed of three items: two buttons and a stack of pictures.

Twinder is based on the same logic as Tinder, we keep the same three items: two buttons, but we change the stack of pictures into a stack of tweets. As we developed our apps for a laptop, we used the keyboard to take the input of the users. There was no need to display the two like / dislike buttons on the screen. To resume, we have a simple screen where the user visualizes

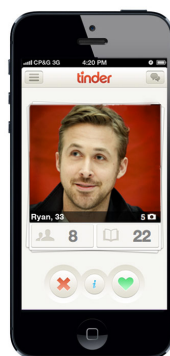


Figure 3.1: Tinder concept is not really new. However, the developer time create an easy application with a perfect user experience [16]. The addiction to the platform is total. [17].

tweets one after another by pressing two buttons. These buttons are two keyboard keys: L for like and D for dislike.

As you can see in the picture, we succeeded in retrieving the embedded tweets. We mentioned in the methodology section that the users need a context to more clearly understand these short messages of 140 charts. The embed format allows us to display the same content as one sees on Twitter, especially the images and other content, which is not only text.

We created a simple design for a simple action: classifying tweets. We developed an easy-to-use platform, which leads to a few random parameters that are difficult to evaluate. In order to control in a better way these few factors, we will describe the data acquisition process.

3.2 Data acquisition

The data acquisition was done in two steps. First, we did a beta test with 3 people and then performed the test on 15 people.

The beta test provided global feedback of the experiment and that allowed us to improve the application and survey. A sociology student advised us on how formulate our users' instructions without a clear bias or influence.

After doing this beta test, we ran a 15 experiences a day. Fifteen users took the same test A. Eight of them did test-time-B while the rest did test-like-B. All candidates shared similar profiles. They were all entrepreneurs working at 500 Startups, one of the biggest incubators in San Francisco. Each test subject had comprehensive understanding of Twitter and the popular new media. An unexpected result of this project showed that people from across the world do not use Twitter in the same way.

The test was administered in a quiet conference room at the 500 startup office.

The candidates took the test individually and received test A simple instructions at the start of the experiment:

- **Press L** if you find the content of the tweet interesting.
- **Press D** if you do not find the content of the tweet interesting.
- Take your time to do the test.

Once the user finished test A, we talked to the user in to try and change their mind. After our conversation, each user would then start test B and finish the survey. Moving forward from the explanation of the experimental setting, we will now define the technical implementation of Twinder.

4 Technical Implementation

In the previous chapter, we explained the methods and experimental settings of the project. We explained that we want to measure two different behaviours of the twitter users: engagement related to time and interest measured by the number of likes. We designed an A/B test with two B tests, which depend on a classification that had previously been done on test A. To analyse the users' behaviours, we designed a simple application, similar to Tinder where the users see their tweets one after another. To move on to the next one, the user must either press the L key to like the tweet or the D key to dislike it. We will discuss the different problems that we faced and the technical solutions we implemented to resolve these issues.

First, we present the programming languages and frameworks that were used. We then clarify the general structure of the application and explain the relevant parts of the code. Finally, we consider the Twitter API and the tools adopted in order to commence our project.

4.1 Languages

When creating a web application, many possibilities exist. We decided to choose Python and Javascript for the Back & Front end, respectively. We built our application on top of Django, which is one of the biggest Python Frameworks. We now will explain our decisions at the technical level.

4.1.1 Back-end

As you can see in diagram 6.1, we tested two different back-end languages: Javascript (Node.JS) and Python. After developing a bite with Javascript, we noticed that Python was more compatible with our protocol and easier to program.

As stated in Wikipedia: *"Python is a widely used general-purpose, high-level programming language."* To develop a web-app, these two criteria are crucial and Python offers more advantage. Indeed, it is a language often used in research due to its general-purpose characteristics and its clear readability as compared to Javascript. Moreover, Python uses modular programming,

whose design separates the functionality of a program into independent, interchangeable modules. This architecture makes the addition, alteration, or deletion of functions much easier. In addition, this language offers the developer a rich amount of data structures, much like dictionary whose lists explain the correct the way to code. In contrast, having many libraries and frameworks can be advantageous, but also other functionalities that we do not need to implement. Finally, the python community and documentation is extensive. Python thus quickly became our Back-end language mainly due to its simplicity, while Javascript became our clear front-end choice, the reasons for which are as follows.

4.1.2 Front-end

In contrast to the back-end model, where we tried two different possibilities, we directly opted for Javascript combined with HTML & CSS in the front-end choice. This trio is the base of front-end web development, seeing as it provides the majority of the tools needed to create a dynamic and well-designed application. As described in our methods and experiment setting chapter, we need an interface where the user interacts directly with their tweets and can like or dislike each one. This language makes it possible to create requests directly to the back-end and then add data to the database. To add some features to our javascript, we installed jQuery, a powerful library, which simplifies actions and AJAX calls requests to the back-end. The two other languages, HTML & CSS, are independently used as the standard markup language for creating web pages and the style sheet language used to describe the aesthetics and formatting of the page, respectively.

The last advantage of our configuration is the Python / Javascript frameworks, which provide useful functions. We will now explain the required frameworks for Twinder.

4.1.3 Frameworks

To complete our setup, we added two importants frameworks: Django [18] and Reveal [19]. The first one is open source and follows the model-view-controller architectural pattern. The MVC pattern facilitates the internal logic of the code and forces the developer to split the functions in three parts, as shown in figure 4.1.

The django framework also provides a robust templating solution which allows us to render html pages with back-end parameters. As you can see in Figure 4.1, the backend function returns an html template with a context. This context is accessible in the html page, which makes it possible to call "tweet" with this syntax: `{{tweet}}`.

Finally, Django comes with a lot of functions that are easy to integrate. In our application, Twinder, we are making use of the authentication and database parts in particular.

The second framework that we used provides us with a simple design and structure to create dynamic slides. Many parameters are available, such as introducing a time for each slide or displaying the progression of the slides. However, we reduce Reveal to its minimum possible level in order to determine the influence to which the user is subject.

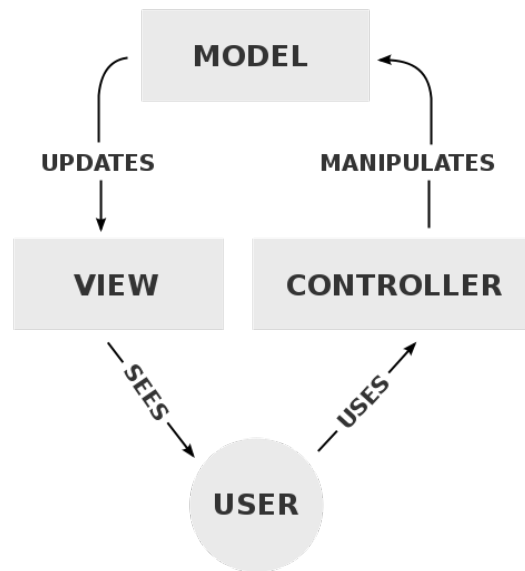


Figure 4.1: Django appears to be an Model-View-Controller framework [20]. It helped us to design properly Twinder.

The two frameworks are available on GitHub:

- **Django:** <https://github.com/django/django>
- **Reveal:** <https://github.com/hakimel/reveal.js/>

The languages used have now been explained, allowing us to now focus on the general structure of the application.

4.2 Structure

We described previously the programming languages and the framework that we use to develop Twinder. In order to meet the expectations of the method, which is visualizing and giving their opinion on Tweets, we choose the duo Python / Javascript to build the apps. The framework Django gave us a structure for the entire architecture of the application and the database.

First we will explain to you the general structure of the application. Then, we will follow by describing the design of the database in detail.

4.2.1 General structure

The architecture of Twinder has evolved since the beginning of the project due to some constraints mainly coming from the API. Twitter limited the number of requests to the API,

that is why, we chose the design that you can see above in figure 4.2.

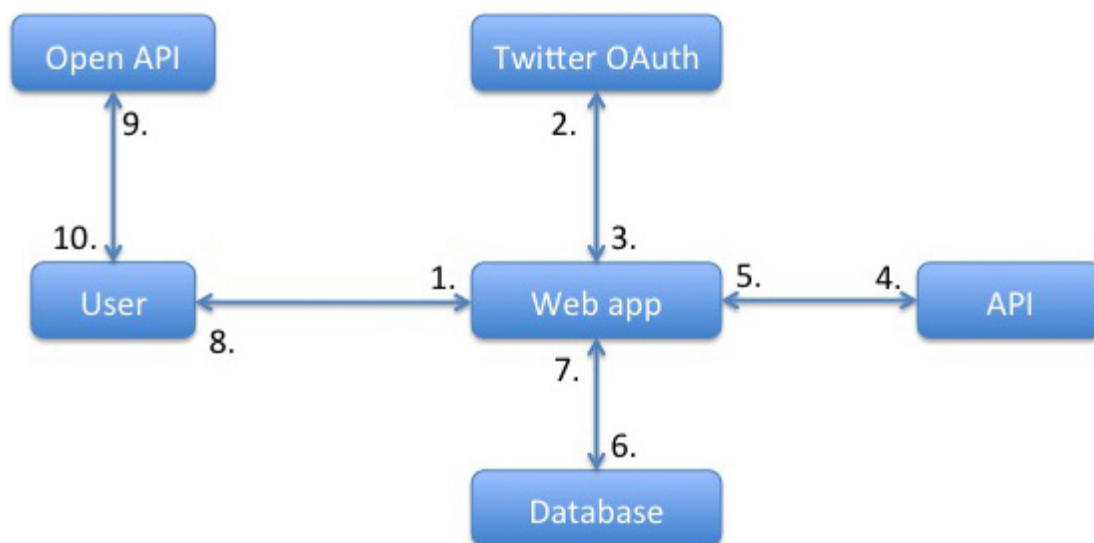


Figure 4.2: The graph shows us the interactions with the API. 1. Connection to Twinder ; 2. Redirect to OAuth serveur ; 3. Token is issued ; 4. Request API + Token ; 5. Answer ; 6. Insert to db ; 7. Retrieve from db ; 8. Answer with tweets id ; 9. Ajax call to Open API to get embed tweet ; 10. Embed tweets.

Thanks to the diagram, you can see the essential steps required for the application to retrieve tweets from the API. In order to be more complete, we should add strides, necessary for the continuity of the process. Indeed, we registered Twinder to Twitter as an official app which allow us to get tokens for the authentication of our users. In figure 4.2, you can see the run of the application which first starts with an Oauth delivered by a dedicate server, a token is issued. With this token, Twinder can request tweets to the API and then stores them in the database. As we explained previously, we now have 25 tweets times 10 friends in the database. After selecting 50 tweets to display, we send the list of ID's to the clients. Our method mentioned the fact that we should display embedded tweets in order to keep the context of the data. However, due to the API limits that we are going to explain after, we were not able to store the embed tweets directly in the database: we had to retrieve them from the client side. This step is represented by arrow 9 and 10, which show the AJAX call to the Twitter unauthenticated OEmbed endpoint. Indeed, as tweets are essentially public, it is possible to retrieve with their id, the embedded form without the constraint of limited requests [24].

4.2.2 Important files

Now that you understand the functioning of the application and the dependency linked to the API, we are going to describe the structure of the files dictated by Django. Figure 4.3

gives you the tree view of the application. We will briefly talk about three important files: *urls.py*, *views.py* and *index.html*, the template. First, *urls.py* is a URL dispatcher which helps the developers keep a clean, elegant URL scheme. You can see above a sample of the code. We attribute to the address ***base_url/tweets***, the function ***index2*** which is in the file *views.py*.

```
1 url(r'^tweets/$', views.index2, name='index2')
```

The next step is handled by *views.py* which contains all the backend functions. Functions are the logic of the application, if they are not internal, they are rendering templates with a context. This context is generated by the function itself and gives the content to the template *index.html*. As we saw in the previous chapter, balises enable the use of the backend data in HTML pages. After this overview of the Django structure, we are conducting a deep dive into the structure of the database.

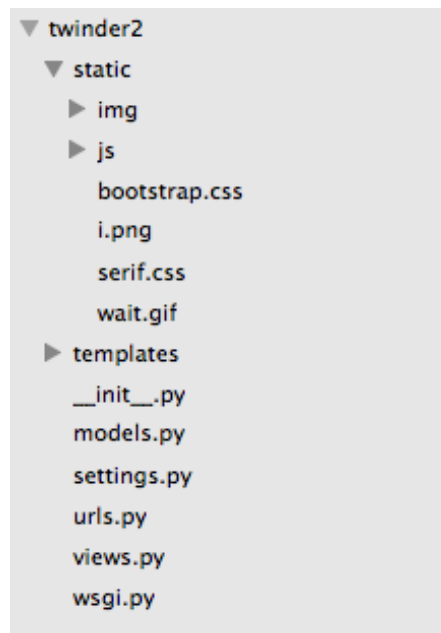


Figure 4.3: Django comes with its own architecture. Statics files are the Javascript, CSS and the images. The templates are rendered by the data coming from *views.py*. *Models.py* generate the database structure and *urls.py* links the urls to function in *views.py*.

4.2.3 Database

We mentioned in the previous paragraph, three important files but we forgot a last one, *models.py*. This file contains the model of our database and is defined by three classes as you can see with this code:

```
1 class UnUser(models.Model):
2     first_name = models.CharField(max_length=30)
```

Chapter 4. Technical Implementation

```
3     last_name = models.CharField(max_length=30)
4     user_name = models.CharField(max_length=30)
5
6     class UneSerie(models.Model):
7         left = models.BooleanField(default=False)
8         right = models.BooleanField(default=False)
9         created_at = models.DateTimeField(auto_now_add=True, editable=True, blank=
10             True, null=True)
11         txt_length = models.CharField(max_length=200)
12         friend_id = models.CharField(max_length=200)
13         tweet_id = models.CharField(max_length=100)
14         user = models.ForeignKey(UnUser)
15
16     class LesTweets(models.Model):
17         tweets = models.TextField()
18         random_ids = models.TextField()
19         friend_id = models.CharField(max_length=100)
20         user = models.ForeignKey(UnUser)
```

The three classes: UnUser, UneSerie and LesTweets are composed by several fields and are linked together by Foreign Keys.

Now that you have a better understanding of the structure of the application and the database, we are going to scrutinise some interesting parts of the code.

4.3 Parts of the code that is interesting

In order to deep dive into the application, we are going to show you three central parts of the code which will give you a better understanding of Twinder. First of all, we describe the authentication process which allow the application to connect to the Twitter API. Next, we explain the javascript code which retrieves the opinion of the user for their tweet. Finally, we introduce to you the main function index which generates the vector of 50 tweets and render the html template.

4.3.1 Authentication

In order to retrieve tweets from a user thanks to the API, we need to first authenticate the candidate. As you can see above in figure 4.2, the application is redirected to Twitter which will provide the login page. Line 4 of the code generates this redirection, and adds to this redirection the KEY and SECRET of the application which is registered on Twitter App [21]. After that the user entered its credentials on Twitter, the application receives through a GET method, the token: `get('oauth_token')` and the secret: `get('oauth_token_secret')` related to the user.

The final line returns an object which contains the credentials of the user and allow the program to query the API. As you probably noticed, we use the wrapper tweepy (ie. line 4 & 6) in order to connect easily to the Twitter API with Python. We will discuss this more in detail in the section concerning the API.

After having provided the details of this authentication function, we will focus on the javascript function which retrieves and sends to the backend the opinion of the user for a given tweet.

```
1 #authentication procedure
2 def authentication(user):
3     instance = UserSocialAuth.objects.filter(user=user).get()
4     auth = tweepy.OAuthHandler(SOCIAL_AUTH_TWITTER_KEY, SOCIAL_AUTH_TWITTER_
5         SECRET)
6     auth.set_access_token((instance.tokens).get('oauth_token'), (instance.tokens)
7         .get('oauth_token_secret'))
8     return tweepy.API(auth)
```

4.3.2 Keyboard & AJAX call

The previous Python function which is in charge of the authentication is related to the backend. We will now jump to the front-end and explain the function which takes the decision of the user (like/dislike) and writes it on the database. This function written in Javascript can be split in two parts: the acquisition of the decision and the transmission of the decision. The acquisition of the information is handled by a library called KeyboardJS. Thanks to this library, the application is listening to the input coming from the two keys: F and L [Lines 25 and 26 of the code in annexe]. When the user presses one of the two keys, it calls the function mark with different parameters depending on the choice of the user. "Mark" has to do the connection between the front-end and the back-end. Thanks to jQuery, the JavaScript library, it is possible to easily create an Ajax query. This query is composed by 3 attributes: the url, the type and the data. We are going to describe briefly these attributes:

- **url:** thanks to the mapping function done by urls.py, the url is linked to the backend function which is ready to handle the data.
- **type:** the POST method allows to transmit data to the backend without passing it through the url in plain text.
- **data:** it is the content transmitted by the function to the backend function. In this case, we are sending a JSON with four components: tweet_id, friend_id, txt_lenght(number of characters in the tweet) and direction (left/right which is a mapping for like/dislike)

Once the data is transmitted, if there is no error between the two sides, "mark" receives an answer from the backend. If data is false, it means that an error occurs in the backend. If this is not the case, the backend handles the JSON and inserts it into the database, creating a new object UneSerie. After having explained the communication between front-end and backend with the function mark, we will deep dive into the main function "index".

4.3.3 Index.html

The index function is responsible of the rendering of the template. Before it calls the html page, we need to authenticate the user thanks to the authentication function line 8 that we describe previously. Then, we add the user to the database if it is not already in it. The next function called, is in charge of retrieving the tweets from the API and put them in the database. Some parameters were introduced in the function "retrieve_tweets_api" in order to change easily the number of friends or tweets to retrieve. Once the tweets are in the database, "retrieve_from_db" display the tweets to the user one-by-one. For more informations, the code is available in annexe or Github [22] .

4.4 API

In the previous chapters, we talk about the general structure of the application, the programming languages that we used and details some important parts of the code. However, we did not explain the API side which allows us to retrieve tweets. Firstly, we are going to talk about the wrapper that we used: Tweepy. Secondly, we will detail the request limitation of the API. Finally, we will point out the method that we found in order to foil this limits of the API.

4.4.1 Wrapper

As we previously explained it, we use Python as backend language. In order to easily connect our application to Twitter API, it was necessary to use a wrapper library. As it is explains on Wikipedia: *"a Wrapper libraries (or library wrappers) consist of a thin layer of code which translates a library's existing interface into a compatible interface. This is done for several reasons:*

- *To refine a poorly designed or complicated interface.*
- *Allow code to work together which otherwise cannot (e.g. Incompatible data formats).*
- *Enable cross language and/or runtime interoperability."*

Tweepy [23], our wrapper library, allows us to easily connect Twinder to the API. Moreover, it is possible to get any object and use any method that the official Twitter API offers. The short sample of code gives you a good idea of how simple it is to retrieve information after having been authenticate.

```
1 user = api.me()
2 print('Name: ' + user.name)
3 print('Location: ' + user.location)
4 print('Friends: ' + str(user.friends_count))
```

4.4.2 Request limit

Once we connected our application to the Twitter API thanks to the wrapper Tweepy, we had to figure out how to deal with the request limit fixed by the company. Indeed, the twitter resources are not unlimited and computational power start to be consequent for platforms as big as Twitter.

A list is available on Twitter developer page [26] and it gives the limit per service. Twinder is limited at several point has some limitations but the bottleneck is the Oembed request which is limited to 180 requests per user every 15min. As we load 250 tweets per person it was a critical issues to solve this problem. We finally found an interesting discussion [25]:

You should find that you're able to fetch as many embed codes as you need from the unauthenticated OEmbed endpoint. Rate limits are not applied to this endpoint in the same way as for API v1.1.

We bypassed the limitation by using a public OEmbed endpoint which is reach by a GET ajax request:

```

1 function streaming(id,friend,txt_length){
2     $.ajax({
3         dataType: "jsonp",
4         url: "https://api.twitter.com/1/statuses/oembed.json?id="+id+'&
           align=center',
5         type: 'GET',
6         success: function(data,args) {
7             if (data.html){
8                 $(".slides").append('<section id="'+id+'" value="'+
                           friend+' name="'+txt_length+'">'+data.html+'</
                           section>');
9             }
10        },
11        error: function(Xhr, textStatus, errorThrown) {
12            alert(Xhr);
13        },
14    });
15 }
```

After having explained the limit issues coming from the API and our solution. We will talk about the versioning tools that we employ: Git and Heroku.

4.5 Deployment

In order to manage this project, we use two services essentials in web programming. On the one side, Github is a Git repository web-based hosting service useful to track the bug in our case. On the other side, Heroku is a cloud platform as a service and allows us to deploy directly from Github our project. You can find the code open source of Twinder on Github [22] and the application on Heroku [27].

5 Results

In the previous chapter, we delved deeply into the technical implementation of the application which has two goals: to individually display tweets and collect information on the series regarding the interest for each tweet and time deciding the level of interest of each tweet. Using the results from fifteen test users and an analysis of the subsequent data, we will display and explain six graphics. Then, we will interpret the results of the survey. For consistent clarity, we will provide the notation that we are going to use at the beginning of each chapter.

5.1 Explanations

As we explain in chapter 3 (method), in order to measure the level of interest and the time of engagement, we created an application, which allows us to do two A/B tests. These two A/B tests, respectively test-time and test-like, are composed by two series each. We have named them test-time-A, test-time-B and test-like-A, test-like-B. To clarify, test-time-A and test-like-A are the calibration tests and that share many similarities. We will present the results acquired by the application, which are based on the number of tweets liked in a series, and the time spend on tweets. The time spent on a tweet is normalized by the number of characters of this tweet. All of the data acquired during the experiment are available on Github [22].

5.2 Twinder results

5.2.1 More likes and more time

Table 5.1 gives us two sets of data. Column 1 shows the percentage of people who took more time to do test B than test A. The second column, column 2, shows the percentage of people who liked more tweets in the test B than the test A.

When we select the tweets for test-time-B based on the time spent on tweets in test-time-A, we obtain better results when testing for time than when using test-like-B. This demonstrates that 37.5% of the users spend more time on the series test-time-B than the first test-time-

A. However, if we consider the number of tweets liked, we obtain clear results if we take tweets considering the number of tweets liked in the series test-like-A. Our information also demonstrates that 71.4% of users doing test-like-B liked more tweets than in the series test-like-A.

test	time	likes
test-like-B	28.6%	71.4%
test-time-B	37.5%	25.0%

Table 5.1: This table shows the percentage of people who take more time and like more tweets in the second series that the first one.

5.2.2 Time and Like rate

The table 5.2 gives us the time and 'like' rates that were computed with this formula:

$$rate = (x_{past} - x_{present}) / x_{past}$$

Because the test-like-A and test-time-A are the same, it allows us to perform a calibration to see the rates between tests A and tests B.

As demonstrated in column one, for both B tests, people spend on average less time on the second series. However, our information suggests that test-time-B has better results than the other because the difference is 2.51% when compared to 24.20% on the other test.

Column two is way better for test-like-B which shows that an increase of 18.52% in the number of tweets like in average compare to test-like-A. As we did not take in consideration the number of 'likes' in test-time-B, we see this number decrease by 9.14%.

test	RateTime	RateLike
test-like-B	-24.20%	18.52%
test-time-B	-2.51%	-9.14%

Table 5.2: This table shows the time rate and like rate between the first and second test.

5.2.3 Time spent on the 100 tweets

The blue dotted curve in Figure 5.1 represents the average time spent by all the 15 users on each tweets. The normalized time of the tests A (respectively test-like-A and test-time-A) are given by the value between 1 and 50. From 51 until the end, we have the normalized time spent on tweets of the tests B. In order to make it more clear, we draw in red the simple moving average of the blue curve.

The red curve is obtain with this formula:

$$MobValue(n) = (x_{n-1} + x_n + x_{n+1})/3$$

We clearly notice that users take more time at the beginning to take a decision. From tweets 1 to 5 people are in average taking double of the time to decide. We do not see the same augmentation at the beginning of tests B with tweets 51-56.

In general, the signal is pretty random with a slight decreasing tendency at the end of each tests, form tweets 30 to 50 and 80 to 100.

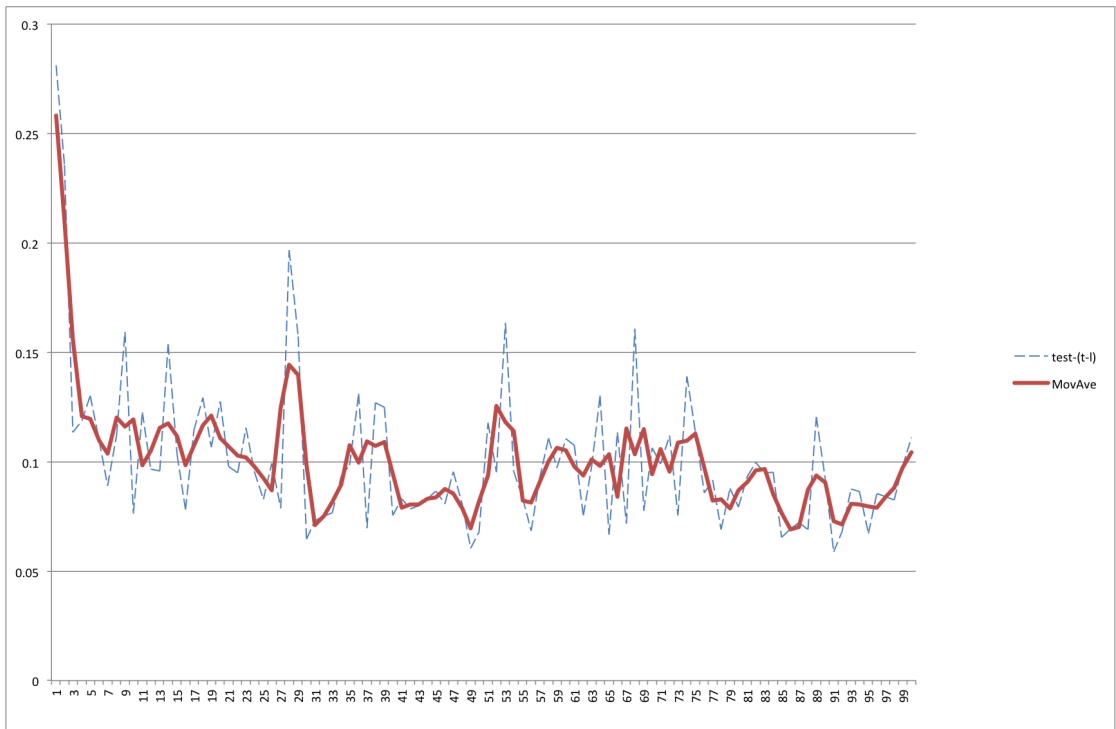


Figure 5.1: The graph shows the time spent in average on the tweets of each series A and B. Test A are tweets from 1 to 50. Test B are tweets from 51 to 100.

5.2.4 Time spent on tests B

Figure 5.2 represents as previously, the moving average of the average time spent on tweets in test-like-B (red curve) and tweets in test-time-B (blue curve). We observe that people spend more time in test-time-B than test-like-B. Moreover, after analyzing the data, we can see that the increase of the red curve close to tweet 39 is due to a value 20 time superior at the average of the other value. Moreover, we can see that the blue signal is more random than the red one. Indeed, in series test-like-B, the variation of the signal is equal to 0.4 compare to the 0.9 for the other curve.

Chapter 5. Results

The decision time is more constant in test-like-B, we can conclude that the decisions are more mechanical than on test-time-B. So, people are less involve in test-like-B and their engagement time is less important. At the opposite, test-time-B has important variations between tweets decision time, it shows that the engagement is not the same for each tweets but it is in general higher than test-like-B.

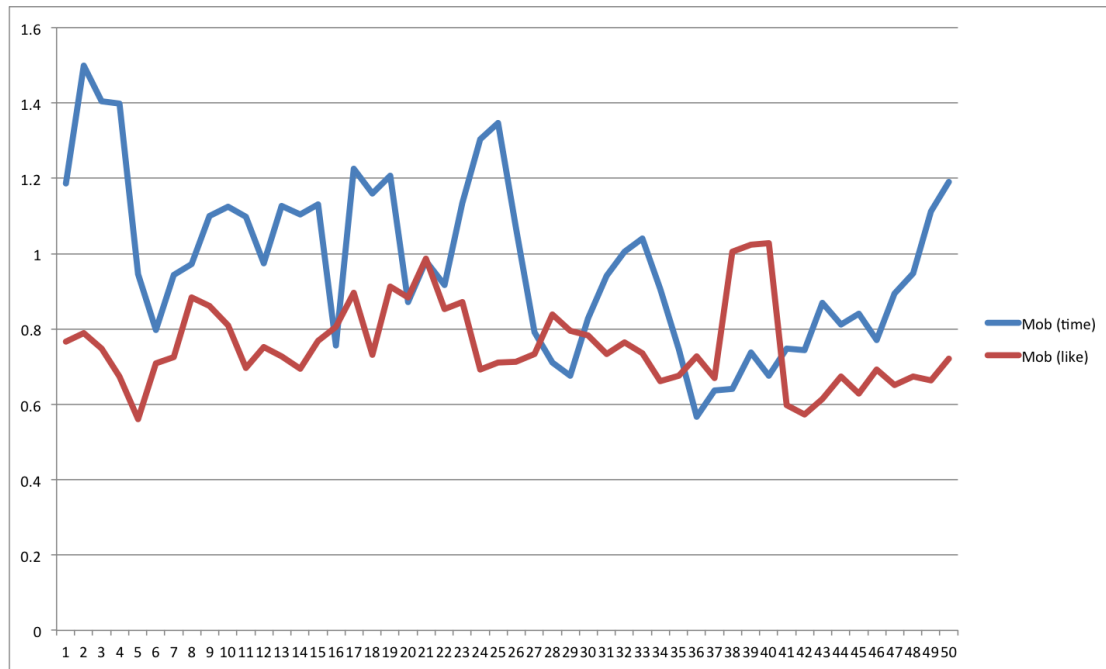


Figure 5.2: The graph shows the time spent in average on tweets in both test-like-B (red) and test-time-B (blue). We notice that the blue line is in average higher than the red one. The time engagement is better in test-time-B.

5.2.5 Number of likes on 100 tweets

Figure 5.3, show the moving average of the average of likes that receives tweets during an experience. As previously, tweets 1 to 50 are the test A and tweets 51 to 100 are test-like-B and test-time-B all together. We can see that the number of likes is almost random during the test. We notice a pick of likes at the beginning of the test B which results from test-like-B and is augmentation of tweets like of 18.52% 5.2. We can conclude that people are classifying tweets until the end of both test. We could have imagine people pressing only L (like) or D (dislike) in order to finish faster.

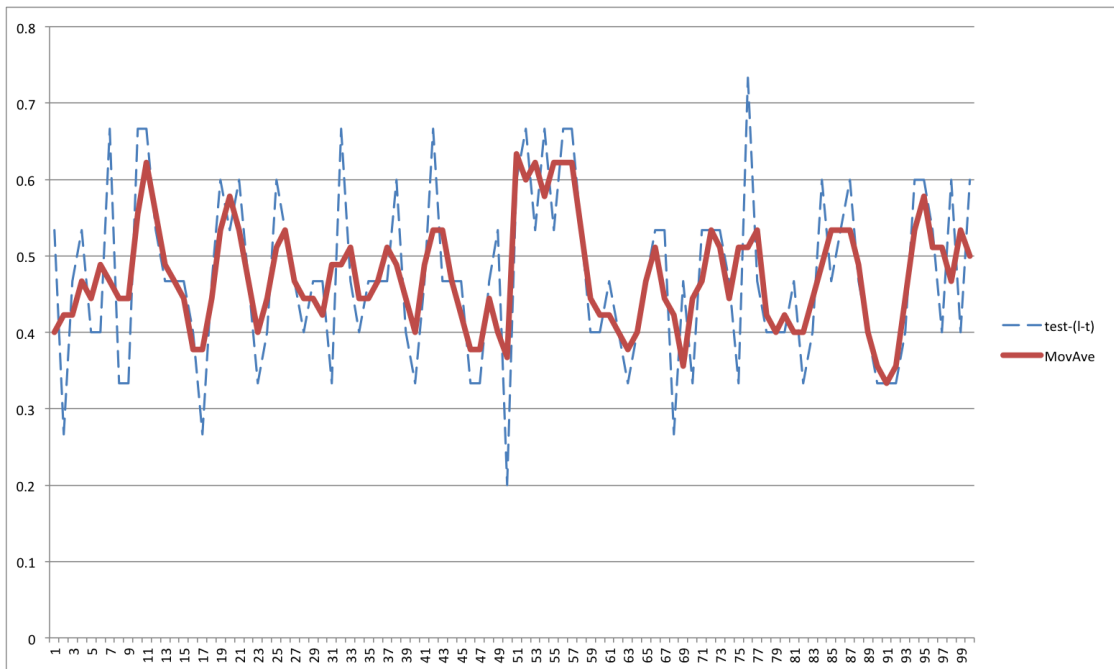


Figure 5.3: The graph shows the average number of likes on each series A and B. Test A are tweets from 1 to 50. Test B are tweets from 51 to 100.

5.2.6 Number of likes in tests B

Figure 5.4, gives us the same information that Figure 5.2 but this time we measure the average of likes and not the time. The curve red shows us the moving average of the normalized number of likes in the test-like-B. Respectively, the curve bleu represents the same information in the test-time-B. We notice that when we select tweets based on the number of likes in test A, we have a higher results than based on the time spent on tweets. This graphic illustrates the results of the table 5.2 which reveal a positive rateLike (18.52%) for test-like-B and negative (-9.14%) for test-time-B.

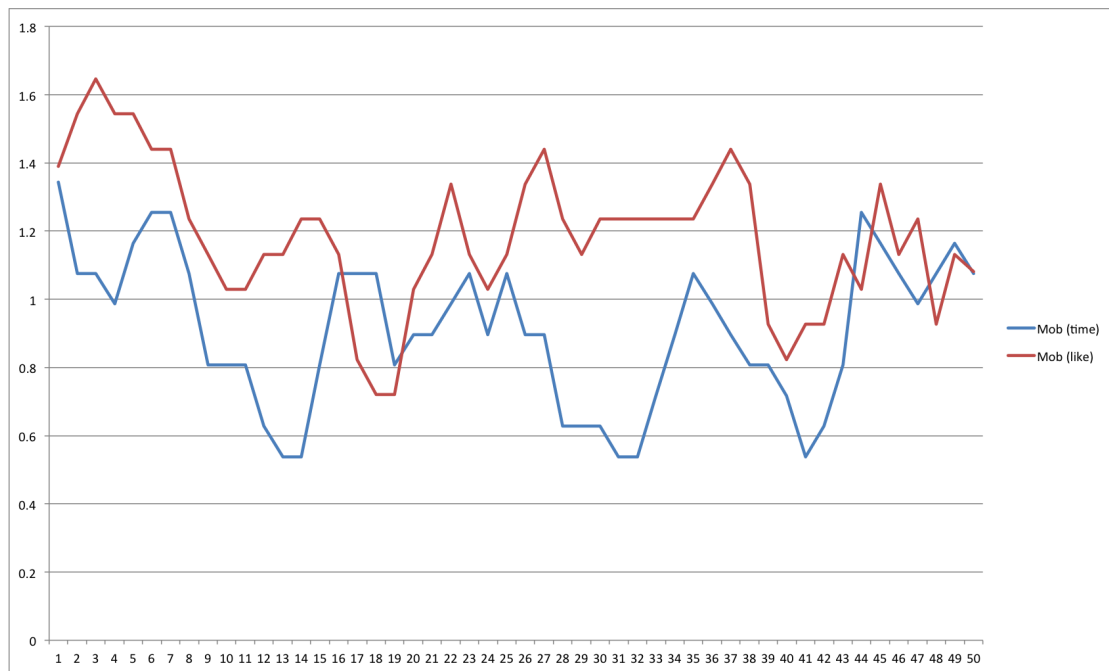


Figure 5.4: The graph shows the average number of likes in both test-like-B (red) and test-time-B (blue). We notice that the red line is in average higher than the blue one. The interest is better in test-like-B.

5.3 Survey

After analyzing data from the experiment, we are going to give you the results of the survey that candidate had to fill out at the end of the test. We will give general results about how the 15 people use Twitter. Next, we will analyze the variation of the results between the test A and tests B. The results of the survey are also available on the Github repository [22].

5.3.1 General questions

In order to have a better understanding of the results of the experiment, it is interesting to have an average profile of our candidates. To do this profile, we asked three general questions about Twitter, then we can interpreted the average score:

- *How often do you use Twitter ? (1: less than once a week / 5:more than once a day):* The average is 3.7, we conclude that our candidates are daily users. They have a good understanding of the service.
- *Do you usually find the content interesting on twitter ? (1:no /5:yes):*We obtain a result of 3.4 over 5, which is just above the average. Users choose the content that they want on their timeline because they select people that they are following.

- *Do you usually feel engaged when you are looking at your timeline on Twitter ? (1:no,5:yes):*
The result is 3 over 5. On one side, we can conclude that people are not so much engaged on Twitter. On the other side, we can say that people cannot easily measure their engagement on Twitter [3].

In conclusion, our average candidate is a daily twitter user who find the content of his timeline enough interesting to look at it every day. However, his engagement is something unclear and the user spend probably more time on Facebook than Twitter. We will now analyse the results between the test A and test B.

5.3.2 Variation between test A and B

The table 5.3 below gives us the variation of engagement, interest and tiredness between test-like-B and test-like-A respectively test-time-B and test-time-A. We see that variation of the interest is in agreement with the previous results 5.4. Indeed, the interest decreases of 3.8% in test-time-B and increase of 26% in test-like-B. The engagement is almost two time higher in test-like-B and we can put in parallel this engagement and the tiredness of the user after the test-like-B. Then, users are more engagement but the price is significantly high, 38% more tired. As for test-time-B, the engagement increases of 10% and candidates are 8.7% less tired than in test-time-A.

Test	Engagement	Interest	Tiredness
test-like-B	20.00%	26.32%	-37.50%
test-time-B	8.33%	-3.85%	-8.70%

Table 5.3: This table shows the variation of three parameters: engagement, interest and tiredness between test A and tests B. These values are coming from the survey [28].

We finally asked to the candidates: *How many tweets are in the series ?*

Test	More in 1 st series	More in 2 nd series	Equal
test-like-B	29.0%	29.0%	43.0%
test-time-B	37.5%	12.5%	50.0%

Table 5.4: This table shows the percentage of people who thought that the number of tweets between the series was different or equal.

In general, no matter if it is test-like-B or test-time-B, approximatively 50% of the candidate answer the correct one: Equal. Now if we consider the statement "*More in the first one*", we notice that the result is higher for test-time-B. We can link this result to the fact that user are less tired after this test than the user.

6 Discussion

We previously presented and analysed the data recorded by the web application and the survey. It gave us a better understanding of the candidates' behaviors and allowed us to see how our A/B test affected length of engagement and interest. In this section, we will summarize the results and share other details learned about our users' perspective. Next, we will discuss some limitations of our design. Finally, we will provide recommendations for more accurate readings in the future.

As a result of our previous test, we had a good understanding of how the application affects the actions of users. To measure the performance of Twinder we recorded the following values: number of tweets liked per series and time spent on the series. Engagement time did not increase between test A and test B but saw a boost in interest of the series test-like-B. Then we see that it is easier to increase the number of likes per series than the time spend on it. However, the augmentation is not without some sacrifices, indeed candidates who took test-like-B were more tired and spent less time on the series than users on test-time-B. So, test-time-B is interesting in the way that users spent almost the same time as test-time-A even if the series is less interesting to them. We will propose after some updates of the experiment in order to improve the time and the interest.

Apart from the two metrics studied, it was interesting to run the experiment and see how users actually utilize Twitter's service. We differentiated the users into two categories:

- **Superficiales (users 4 & 5):** They usually do not read tweets and check only the name of the writers. On average, they spend one second on each tweet and pay more attention to tweets with pictures, videos or hashtags. Their engagement time is hard to increase if we consider only tweets with text.
- **Careful (users 6 & 12):** They check twitter less often than the superficiales but do read all the tweets more carefully. Their engagement is high and it is possible to augment it if the content is well selected. (ie: User 6 was so engaged that they started to interact with

me and told me to read it). After interpreting the results, we are going to talk about the different problems that we had with the design and the application. [3].

Even though the application brought explanations of how people look at tweets we noticed some limits and problem links to the experimental design. Indeed, even if it is not too significative on the results, classifying 100 tweets is a big ask in terms of concentration and can bore people. The problem comes from the fact that our sample set included 10 friends and so the content of tweets were limited especially during the second part of the experiment (test-like-B and test-time-B). One of the users told us at the end of the test:

"I know exactly what people are posting on twitter (e.g. VC's, Business writers, etc). The first series was enjoyable but the second one was repetitive in content and similar to the first one without variation on content. It bothered me."

After these general comments on our method, we can say with our data that people are not fully concentrated at the beginning of the test A. This problem arises especially with test-time-B and the ratio between the time spend on the first test and the second one. The last limit to the experiment is the parameters that we choose to determine the interest and the engagement of the users. This approach is binary and prevents us to get more information from users, especially if they do not have an opinion on the subject. These limits impacted the project and we will try in the next paragraph to bring about improvements to the design of the application.

We will provide some improvements to the process as discussed previously. First, we should not take into consideration the first 5 tweets of test A. They would be a pre-test in order to let users adapt to Twinder and the concept of classification. One of the most important modifications would be to filter out tweets that include anything but text, i.e. no images, videos, etc. The related work on Twitter that we found was showing that the engagement on images is better than on text. For your model, it was not possible to take it in consideration. It would be interesting to develop a mobile app based on Twinder and measure the engagement on it. Users would utilize it to visualized tweets one-by-one and then be able to sort their Twitter timeline at the same time. Then, it would be possible to measure time of engagement during different periods of the day (e.g. public transport, work, home, etc).

In order to improve at the same time interest and time engaged we could improve the formula which ranks the 10 friends in test-like-B and test-time-B. A combinaison of time spent and number of likes would be easy to develop and test in the same way that we did previously. By fine tuning, it would be possible to find a good compromise and then increase the interest and the engagement of the users.

Another possibility would be to design a new application which compares the time spent on a list of tweets as Twitter displays it, then do the calibration with Twinder and finally show the

user a new sorted list. This tryptic would be a concrete case and would show if Twinder really has an impact on the list.

Previously we noticed that the metrics used are not completely reliable. It would be interesting to measure the focus of users with a better approach and then link it to the engagement of the candidates. Indeed, it would be interesting to measure the users' brainwaves during the tests. Many applications could result from these new data sets and they would be a better predictor for the decisions of a user.

Conclusion

This multidisciplinary thesis gave us a better understanding of user behavior on Twitter. The application incorporated Human Computer Interaction, User Experience and sociological methods. We first wanted to create an application which increased the time engaged on series of Tweets seen one-by-one. The two A/B tests made it possible to check the number of tweets 'liked' or the time spent on tweets during the first series and if they had an important influence on the interest and the time engaged on the second series. With Twinder, the web application developed for the project, it was possible to display tweets one-by-one while recording the time spent on them and the decision made to 'like' or 'dislike' each tweet. While many others studies have been done on this subject, their focus has centered on levels of engagement with the content of the tweets. With our design, it was possible to precisely measure the time of engagement for each tweet. After running the experiment on 15 users, we analyzed the results of the application and the survey. Despite the few numbers of candidates, we could see clear differences between the test-like-B and test-time-B. While it is easier to increase the interest of the users, as demonstrated by test-like-B, this interest strongly affects engagement. The users spend much less time on the series and feel more tired at the end of this test. The opposite is seen, however, when people spend almost the same time on test-time-A and test-time-B despite the decrease in interest for the second series.

This project also demonstrated the different ways people are using Twitter. We identified two categories: the superficial user and the careful user. These behavioral differences are essential because it shows the limits of our experience.

The levels user engagement seem to be the new metric for emerging, massive social networks. However, it is an extremely difficult constant to measure. Depending on social networks, we can link the engagement to the engaged time, the number of 'likes', and numbers of retweets. The irregularity of the engagement makes it difficult to produce strong results. However, as we mentioned in our discussion, it would be interesting to measure this user focus with an EEG. These brainwaves could directly give the users' state after each tweet, and they would act as a better metric for engagement. A redesign of the methods based on the brainwaves and the solutions given in the discussion would be a good starting point to increase user engagement on Twitter.

Annexe

Javascript file:

```
1 // Full list of configuration options available here:
2 // https://github.com/hakimel/reveal.js#configuration
3 $(document).ready(function(){
4     Reveal.initialize({
5         controls: false,
6         progress: true,
7         history: false,
8         center: true,
9         //autoSlide: 5000,
10        keyboard: true,
11        //loop: true,
12        progress: false,
13
14        theme: Reveal.getQueryHash().theme || 'serif', // available themes are in /
15              css/theme
16        transition: Reveal.getQueryHash().transition || 'linear', // default/cube/
17                  page/concave/zoom/linear/fade/none
18
19        //keyboard modifications
20        keyboard:{
21            '76':'next',
22            '70':'next',
23        },
24    });
25    KeyboardJS.on('l', function() { mark(Reveal.getCurrentSlide(),'left');add_
26    last() }, null)
27    KeyboardJS.on('f', function() { mark(Reveal.getCurrentSlide(),'right');add_
28    last() }, null)
29
30    function mark(slide,direction) {
31
32        if ($(slide).attr('id')){
33            $.ajax({
34                url: "/mark/",
35                type: 'POST',
36                data: {'tweet_id': $(slide).attr('id'), 'friend_id':$(slide).attr('
37                value'), 'txt_length':$(slide).attr('name'), 'direction':direction
38            },
39            success: function(data) {
40                if (data === false){alert('error')};
41            }
42        });
43    }
```

```
39         }
40     })
41 }
42 }
```

Index.html code:

```
1  {% extends 'base.html' %}
2  {% load staticfiles %}
3  {% block main %}
4      {%if error %}
5      problem
6      {% else %}
7          {% if user and not user.is_anonymous %}
8              <div class = "reveal">
9                  <div class = "slides">
10                     <section id="1" name="0" value="0">
11                         Ready!</br>
12                         Press L to start.
13                     </section>
14                 </div>
15             </div>
16
17             {% else %}
18                 <div class = "reveal">
19                     <div class = "slides">
20                         <section>
21                             <h3>w e l c o m e</h3>
22                             click <a href="{% url 'social:begin' 'twitter' %}?next={{
23                                 request.path }}">Here</a> to login to Twitter
24                         </section>
25                     </div>
26                 </div>
27             {% endif %}
28
29             {%if le_json %}
30             <script type="text/javascript">
31                 var tweets_f = $.parseJSON("{{le_json|escapejs}}");
32             </script>
33             {% else %}
34             <script type="text/javascript">
35                 var tweets_f = '';
36             </script>
37             {% endif %}
38
39
40
41 <script type="text/javascript">
42
43     display();
44
45     //display tweet after streaming them
46     function display(){
47
48         for (i=0;i<tweets_f.length;i++){
49             var tweet_id=tweets_f[i].tweet_id;
50             var friend_id=tweets_f[i].friend_id;
```

```

51         var txt_length=tweets_f[i].tweet_txt;
52         streaming(tweet_id,friend_id,txt_length.length);
53     }
54 }
55
56 //Stream tweets tks to ajax call
57 function streaming(id,friend,txt_length){
58     $.ajax({
59         dataType: "jsonp",
60         url: "https://api.twitter.com/1/statuses/oembed.json?id="+id
61             +'&align=center',
62         type: 'GET',
63         success: function(data,args) {
64             if (data.html){
65                 $(".slides").append('<section id="'+id+'" value
66                                    ="'+friend+'" name="'+txt_length+'">'+data.html
67                                     +'</section>');
68             }
69         },
70         error: function(Xhr, textStatus, errorThrown) {
71             alert(Xhr);
72         },
73     });
74 }
75
76 //add last page
77 function add_last(){
78     if ($('#section').length=== 51){
79         $(".slides").append('<section><h3> Thanks {{ user.get_full_name|
80                             default:user.username }}!</h3>This is the end</section>');
81     }
82 }
83
84 </script>
85
86 {% endblock %}

```

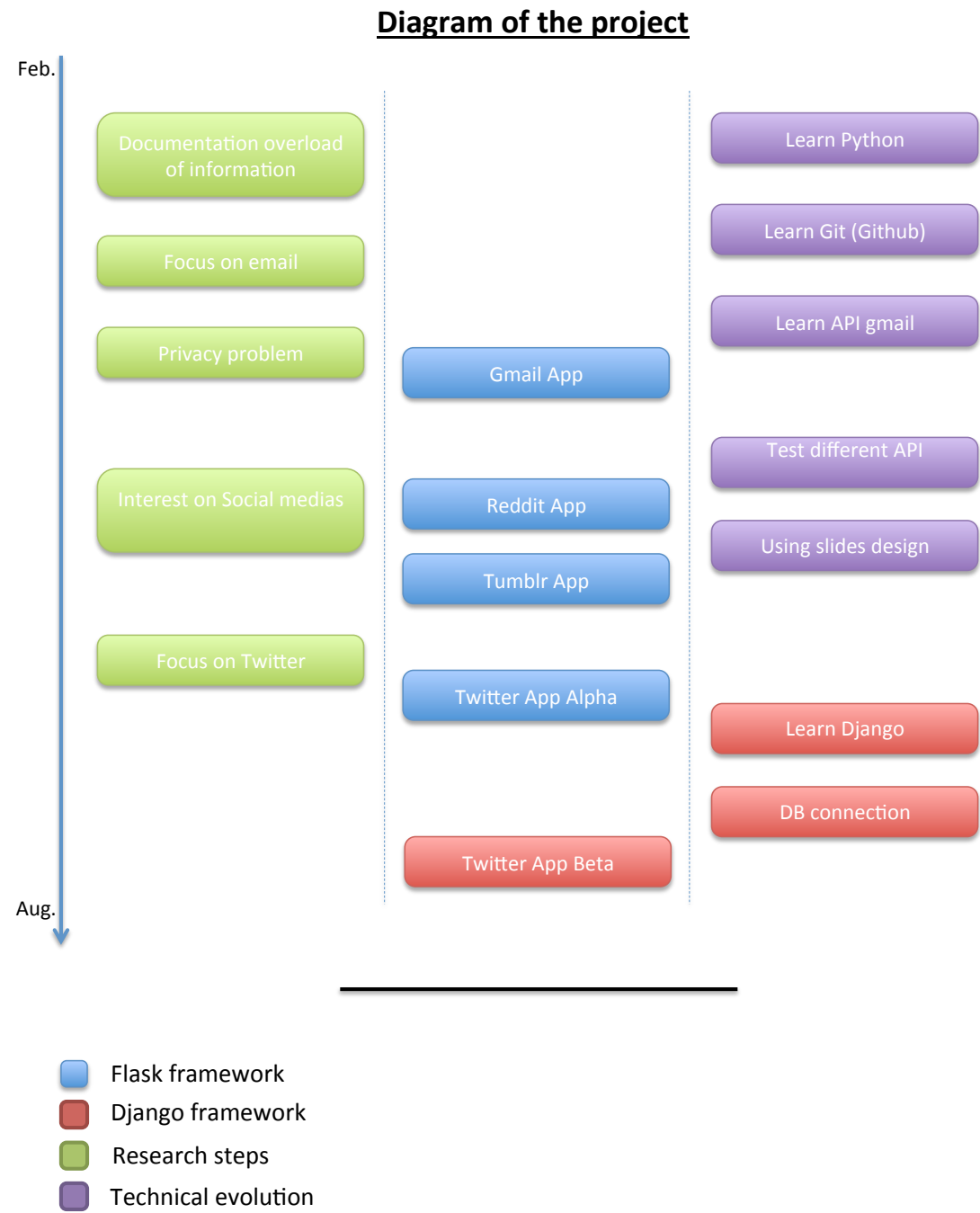


Figure 6.1: This diagram shows the evolution of the project through three aspects: research, apps implementation and technique.

Bibliography

- [1] <http://www.marketingpilgrim.com/2013/03/facebook-admits-it-has-a-young-person-problem.html>
- [2] <http://mashable.com/2014/02/06/twitter-stock-drop-earnings/>
- [3] <http://gigaom.com/2012/10/12/dark-social-why-measuring-user-engagement-is-even-harder-than-you->
- [4] <http://www.whatisedgerank.com/>
- [5] <http://marketingland.com/edgerank-is-dead-facebooks-news-feed-algorithm-now-has-close-to-100k-w>
- [6] <http://techcrunch.com/2010/04/22/facebook-edgerank/>
- [7] <https://blog.twitter.com/2014/what-fuels-a-tweets-engagement>
- [8] <http://www.socialbakers.com/blog/467-formulas-revealed-the-facebook-and-twitter-engagement-rate>
- [9] <http://www.independent.co.uk/life-style/gadgets-and-tech/news/twitter-launches-custom-timelines-to-cut-down-on-information-overload-8936768.html>
- [10] <https://blog.twitter.com/2014/coming-soon-a-whole-new-you-in-your-twitter-profile>
- [11] <http://www.onlinesocialmedia.net/20140409/new-twitter-design-rollout-vs-old-design/>
- [12] <http://www.marketingdive.com/news/one-small-design-change-boosted-twitters-engagement-617/292965/>
- [13] <http://asiancorrespondent.com/111980/cambodia-election-proves-facebook-more-than-just-a-social-n>
- [14] <http://press.experian.com/united-states/press-release/experian-marketing-services-reveals-27-percent.aspx>
- [15] <http://www.businessinsider.com/social-media-engagement-statistics-2013-12>
- [16] <http://www.itsmakeable.com/unconventional-wisdom/good-user-experience-design-ux-can-do-what-r>
- [17] <http://www.businessinsider.com/why-tinder-is-so-addictive-2014-7>

Bibliography

- [18] <https://www.djangoproject.com/>
- [19] <http://lab.hakim.se/reveal-js/>
- [20] <https://docs.djangoproject.com/en/dev/faq/>
- [21] <https://dev.twitter.com/>
- [22] <https://github.com/mnowik/twinder2/>
- [23] <http://www.tweepy.org/>
- [24] <https://blog.twitter.com/2013/api-v1-retirement-update>
- [25] <https://dev.twitter.com/discussions/19022>
- [26] <https://dev.twitter.com/docs/rate-limiting/1>
- [27] <http://twinder2.herokuapp.com/>
- [28] https://docs.google.com/forms/d/1_oLdfTxdSBP8BXIu3J-Xu4OliZsSaVkFpptje1S3-AE/viewform?usp=send_form
- [29] <http://www.pearanalytics.com/blog/2009/twitter-study-reveals-interesting-results-40-percent-pointless-babble>
- [30] <http://techcrunch.com/2014/08/01/facebook-is-down-for-many-2/>
- [31] <http://www.forbes.com/sites/markfidelman/2013/06/05/3-twitter-engagement-tricks-you-should-do-every-day>
- [32] <http://www.stanforddaily.com/2013/11/15/information-overload-and-social-media/>
- [33] <http://arxiv.org/pdf/1403.6838.pdf>
- [34] <http://mashable.com/2012/11/28/social-media-time/>
- [35] <http://blog.shift.com/blog/2014/1/17/twitter-engagement-study-photo-vs-text-tweets.html>
- [36] <http://blog.chartbeat.com/2013/03/18/using-engaged-time-to-understand-your-audience/>