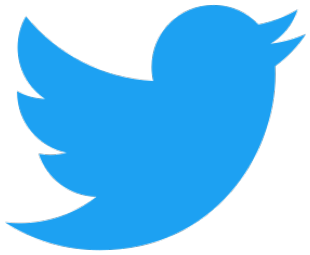


.: Rapport de projet T.E.R :.
.: Projet informatique - HLIN405 :.
.: **Humeur des tweets** :.

Notre équipe de programmation :
Canta Thomas / Reiter Maxime / Desgenetez Charles
Gracia-Moulis Kévin / Bancillon Adrien

Encadrant :
Pascal Poncelet

Licence 2 - Informatique
Faculté des sciences de Montpellier
Année universitaire 2018/2019

Résumé

Voilà presque 4 mois que l'on travaille en parallèle de nos autres cours sur ce projet. Il fut très éducatif de part la faible utilisation de certains langages jusqu'à aujourd'hui peu vus dans notre scolarité ainsi que part la découverte de la complexité associée à l'analyse de données. Il nous a donc fallu un certain temps d'adaptation avant d'arriver à tous bien nous coordonner sur le projet afin d'avancer le programme ensemble de manière constructive.

Nous tenions à remercier notre encadrant, Mr Pascal Poncelet, qui fut d'une grande aide lors de l'avancée du projet notamment part ses précieux conseils, sa bienveillance et son accompagnement sans quoi le projet ne ressemblerait pas à ce qu'il est aujourd'hui.

TABLE DES MATIÈRES

INTRODUCTION

Contexte : Depuis de nombreuses années la communauté des sciences des données s'est intéressée à détecter des opinions dans des documents, dans des réseaux sociaux, etc. C'est dans ce contexte que s'inscrit notre TER : détecter les opinions exprimées dans des tweets. Twitter met à disposition un certain nombre d'API pour récupérer les tweets. Lorsqu'il est interrogé le serveur retourne une structure JSON contenant non seulement le message du tweet mais également de très nombreuses informations dans les méta-tags associés : nom de l'utilisateur, localisation, tweet ou re-tweet, profil de l'utilisateur et bien d'autres. De plus il est possible de spécifier via les API le type de tweets que l'on cherche à obtenir : mots clés, régions personnes, etc ainsi que la méthode en flot continu (stream) ou sur une période passée (7 derniers jours).

Travail à réaliser : Dans un premier temps il faudra développer une application qui permettra de récupérer des tweets et de pouvoir les sauvegarder dans une base de données. Par la suite, il faudra analyser le contenu des tweets pour pouvoir leur affecter une opinion positive ou négative. Enfin il faudra que l'application puisse afficher les résultats agrégés.

CHAPITRE

1

CONCEPTION

1.1 Pistes de conception

La conception du projet n'a pas été évidente dès le départ. Le sujet étant relativement vaste, il nous a fallu dans un premier temps explorer Twitter, les API et les idées que nous avions. Dès la première réunion, notre encadrant nous a clairement dit que le projet n'avait pas un but précis mais la finalité était plutôt d'apprendre des concepts notamment sur l'analyse de texte. Il fallait suivre néanmoins une trame : récupérer des tweets, les analyser et afficher nos résultats d'une manière compréhensible et utile.

Notre premier réflexe a été de penser qu'il fallait réaliser une application en *python* seulement. Mais très vite est venu le problème de l'interface qui était pour le peu très rudimentaire. Nous avons donc essayé de chercher des librairies qui pourraient nous aider à obtenir une interface digne de ce nom. Pour ça, nous connaissions *Qt*, une libraire très complète disponible dans plusieurs langages, dont le *python*. Après plusieurs essais, nous avions envie d'essayer une interface Web. Beaucoup plus simple, très accessible et portable. C'est sur cette approche que nous sommes restés en développant une interface en HTML/CSS/JS.

Ainsi, nous avons nos deux éléments : l'interface web et le serveur en *python*. Enfin, nous avons voulu stocker les tweets afin de minimiser les requêtes que nous faisions à l'API *Twitter*. En effet, nous étions limités à 180 tweets toutes les 15 minutes. Enregistrer nos tweets dans une base de données permettait donc de pouvoir les analyser à répétition sans trop de contraintes. Pour le choix de la technique de sauvegarde, notre référent nous a très vite redirigé vers une méthode dite *sans relations* donc en excluant *MySQL* par exemple et en privilégiant *MongoDB*.

1.2 Cahier des charges complet

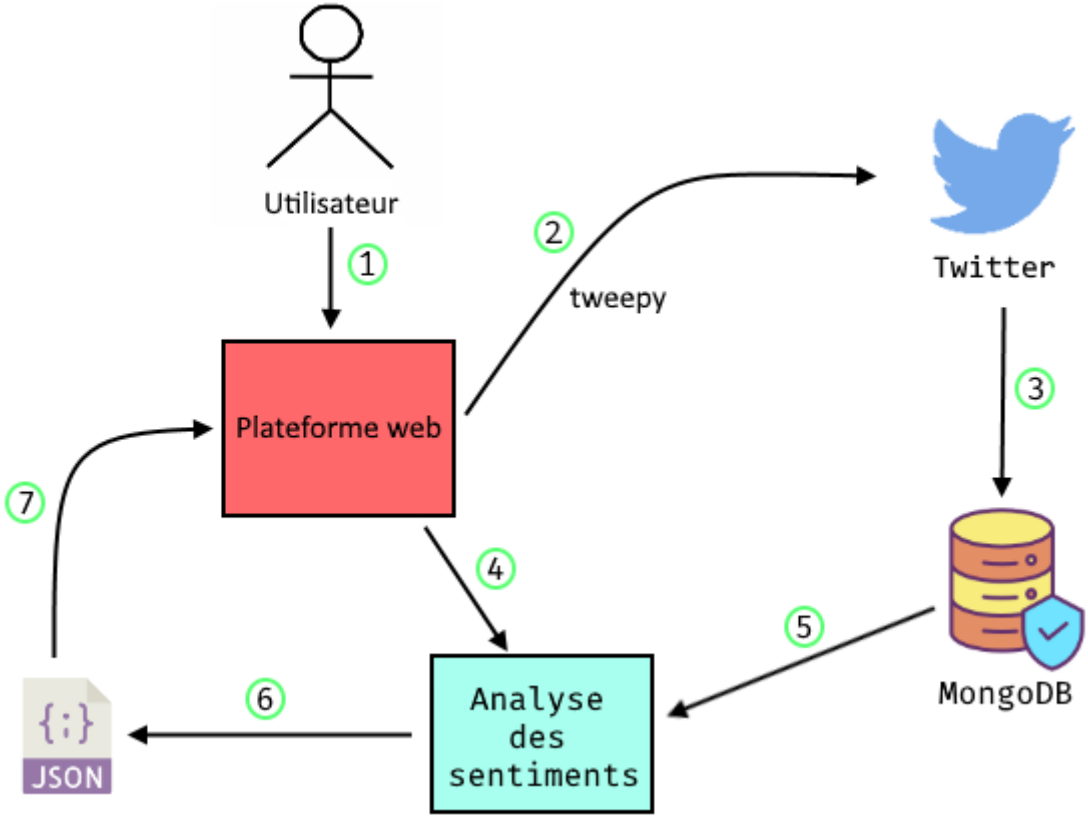


FIGURE 1.1 – Architecture générale

1. Dans un premier temps l'utilisateur interagit avec la plate-forme web et nous récupérons les données spécifiées par l'utilisateur dans les champs :
   - Recherche
   - Nombre de Tweets
   - Date de début
   - Date de fin
2. Lorsque les données de l'utilisateur sont récupérées elles sont envoyées à Twitter avec l'utilisation de Tweepy
3. Ensuite Tweepy nous renvoie les résultats dans notre base de données pour les stocker et avoir un accès plus simple aux données (ici MongoDB)
4. Lorsque c'est fini la plate-forme web lance l'analyse des sentiments de chaque tweet
5. Lors de l'analyse des sentiments le programme appel MongoDB pour récupérer les tweets
6. Une fois l'analyse finie les valeurs associées sont inscrites dans un JSON avec :
   - Nom de l'utilisateur
   - Le tweet lui-même
   - La polarité
   - Les autres données pour l'affichage web
7. Une fois le JSON terminé la plate-forme web actualise la page et affiche les données facilement lisibles pour l'utilisateur

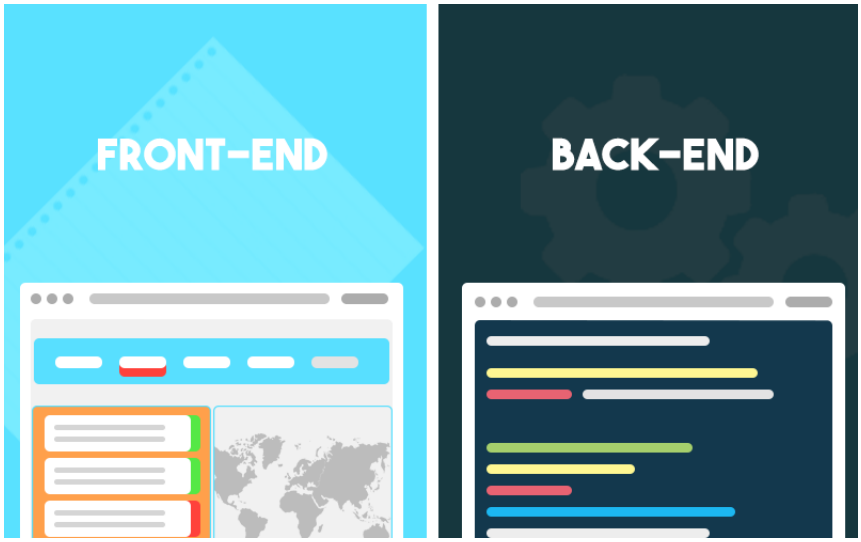
CHAPITRE

2

ORGANISATION

2.1 Division du travail

Afin de réaliser le projet efficacement, nous avons créé deux équipes dès les premiers instants après la conception terminée. Nous n'avons pas hésité bien longtemps, il nous fallait une équipe de développement de l'interface web (*front-end*) et une équipe de développement en python (*back-end*).



2.2 Moyens de communications

2.2.1 Salle Informatique

Nous passions **la plupart de notre temps** en salle informatique, à la Faculté des Sciences, si nous n'étions pas en cours. Nous nous regroupions afin de **discuter** des tâches que l'on avait chacun à accomplir, **partager** nos idées et nos envies sur notre programme.

2.2.2 Trello

Trello est un **outil de gestion de projet** complet, Il est basé sur une organisation des projets en planches listant des cartes, chacune représentant des tâches. Les cartes sont assignables à des utilisateurs et sont mobiles d'une planche à l'autre, traduisant leur avancement. La gestion de l'avancement, la jonction des idées et l'affichage des bugs se faisait donc sur cette application web. Ce site nous a permis d'**établir un vrai plan de travail** avec répartition des tâches et imposition d'une date limite à la réalisation d'une fonctionnalité. (cf : Aperçu de notre Trello, figure 5 page 16).

2.2.3 Discord

Discord est un logiciel gratuit de VoIP *(Voice over IP)* conçu à la base pour les communautés de joueurs. Tel que Skype, il permet de **communiquer** oralement ou à l'écrit. Il nous permettait, le soir ou le week-end de nous entretenir autour des améliorations et des nouveautés ajoutées au programme.

2.2.4 GitLab

Grâce à la faculté des sciences nous possédons un compte GitLab. Ce logiciel offre un emplacement pour **le stockage de code** en ligne et le **développement collaboratif** de projets. Le dépôt comprend un **contrôle de version** pour permettre l'hébergement de différentes branches et versions de développement, permettant aux utilisateurs d'inspecter le code précédent et de le restaurer en cas de problèmes imprévus. Vous pouver cliquer ici pour accéder à notre dépôt.

CHAPITRE

3

RÉALISATION

3.1 Récupération des tweets

La première étape est de récupérer les tweets recherchés par l'utilisateur dans Twitter. Pour faire cela nous utilisons le module Python Tweepy qui nous permet de faire une recherche soit en stream (flot continu de données) ou sur une période passée, ici dans ce projet nous ne regardons pas les données en flot continu mais sur une période passée (7 jours maximum). La plate-forme web nous renvoie 4 données différentes qui sont utilisées pour lancer la recherche :

- La recherche
- Le nombre de tweets recherchés
- La date de début
- La date de fin

```
0  import tweepy
1  api = tweepy.API(auth)
2  for tweet in tweepy.Cursor(api.search,
3                             q=recherche,     # Le mot/phrase rechercher
4                             since=debut,     # La date de début
5                             until=fin,       # La date de fin
6                             tweet_mode='extended', # Cette ligne permet de
                                 récuperer les tweets entier
7                             lang='fr'                # On cherche
                                 principalement des tweets français
8                             ).items(max_tweets):    # Délimite le nombre de
                                 tweets a récuperer
9          if 'retweeted_status' in dir(tweet):
10             continue # Lorsqu'un retweet est trouvé on l'ignore (normalement il
                  n'y en a pas)
11         else:
12             process_or_store(tweet._json,macollection) # On lance une
                  fonction secondaire qui stocke l'information dans MongoDB
```

Listing 3.1 – inserintomongo.py

3.2 Stockage des informations

Le stockage d'informations se fait avec l'outil de base de données MongoDB. Il est appelé lors de la récupération des tweets (vu au-dessus) pour les stocker et ensuite lors de l'analyse des tweets pour récupérer les données stockées. On utilise le module python Pymongo qui permet l'utilisation de MongoDB avec Python

Le stockage des données est simple il suffit de transformer les informations transmises par Tweepy en json et de les envoyer sur la base de données :

```
0  import pymongo
1  import json
2  from bson import json_util
3  #Setup de mongodb
4  monclient = pymongo.MongoClient("mongodb://localhost:27017/")
5  madatabase = monclient["madatabase"]
6  macollection = madatabase["tweets"]
7  ...
8  def process_or_store (tweet, macollection) :
9      data = json_util.loads(json.dumps(tweet)) #transforme en donnée
           utilisable par pymongo le json
10     macollection.insert_one(data) #rajoute le json dans la collection tweets
```

Listing 3.2 – inserintomongo.py

3.3 Analyse des tweets

Une fois les données stockées il faut les analyser. Pour faire cela c'est très simple il suffit de parcourir MongoDB et de lancer l'analyse. Comme vous pouvez le voir on utilise différentes méthodes de traitement :

```
0  for tweetjson in macollection.find({}):#Récupère chaque "tweet" dans toute la
       base de données
1          polarite = tt.process(tweetjson["full_text"]) # Traite avec
               TreeTagger + Sentic.net
2          textBlobFR = tb.processblobfr(tweetjson["full_text"]) # Traite
               avec TextBlob FR
3          textBlobEN = tb.processbloben(tweetjson["full_text"]) # Traite
               avec TextBlob EN
4          classy = cl.classy(tweetjson["full_text"]) # Traite avec le
               classifieur
```

Listing 3.3 – server.py

3.3.1 Dictionnaire

La première méthode que l'on a utilisée est celle du Dictionnaire. Cette méthode consiste à analyser chaque mot et le comparer à un dictionnaire qui a une valeur associée au mot, si le mot ne se trouve pas dans le dictionnaire alors rien n'est renvoyé.

Dans un premier temps on sépare la phrase en tokens, i.e on sépare les mots 1 par 1

```
0  tbo = preprocess(tweetjson["full_text"]) #Met en tokens le texte
1  # Exemple : "Salut ça va :)" -> [ 'Salut', 'ça', 'va', ':)']
```

Listing 3.4 – dictionnaire.py

Ensuite on regarde si le mot est dans le dictionnaire et s'il est dedans on rajoute +1 au compteur de mot positif sinon on rajoute +1 au compteur de mot négatif

```
pos, neg = 0, 0
for word in tbo: #Pour chaque mot dans tbo regarde si il est dans le dico et sont classement
    if isDansDico(word):
        if isPositif(word):
            pos += 1
        else:
            neg += 1
```

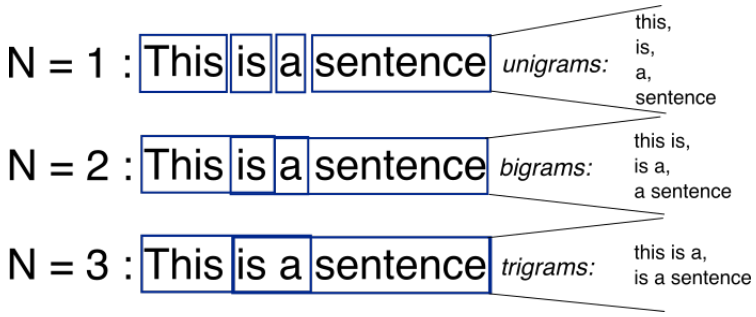
Listing 3.5 – dictionnaire.py

Une fois le texte entièrement traité on analyse les deux compteurs et on attribue au tweet l'une des valeurs suivantes :

- **1** si le compteur positif est supérieur au compteur négatif,
- **−1** si le compteur négatif est supérieur au compteur positif,
- **0** si le compteur positif est égal au compteur négatif.

3.3.2 Sentic.net et n-grams

Après avoir utilisé la première méthode nous avons remarqué qu'il existe beaucoup d'expressions qui sont composées de nombreux mots donc nous avons cherché une solution et nous sommes tombé sur la méthode des n-grams qui nous permet de regarder les groupes de mots.



De plus nous avons changé notre manière d'analyser les mots et nous utilisons une API nommée Sentic.net qui nous renvoie des valeurs réelles entre -1 et 1 et non pas des valeurs binaires.

3.3.3 Treetagger et Sentic.net

Nous avons très vite remarqué que la dernière méthode avait beaucoup de mots parasites, par exemple le déterminant "de" à une polarité de 0.1 ce qui fausse grandement les résultats. Aussi il n'existe pas beaucoup d'expression de plusieurs mots sur Sentic.net.

Donc nous nous sommes focalisé sur les **adjectifs** et les **adverbes** car c'est eux qui permettent le plus souvent d'exprimer un sentiment. Pour cela nous avons besoin d'un TreeTagger, un TreeTagger c'est un outil qui permet d'analyser les mots et de donner leur étiquette grammaticale (leur nature) pour ensuite récupérer seulement les adverbes/adjectifs.

Nous utilisons le TreeTagger de *Helmut Schmid* (lien) et le module Python associé : treetaggerwrapper.

Dans un premier temps il faut tagger le texte, i.e donner leur étiquette grammaticale :

```
import treetaggerwrapper
...
def process(texte):
    ...
    tags = treetagger.tag_text(texte)
    tags = treetaggerwrapper.make_tags(tags)
    ...
```

Listing 3.6 – treetag.py

Lorsque l'on tag un mot on récupère 3 données, par exemple pour le mot beau on retrouve :

```
>>> import treetaggerwrapper
>>> treetagger = treetaggerwrapper.TreeTagger(TAGLANG='fr')
>>> tags = treetagger.tag_text('beau')
>>> tags
['beau\tADJ\tbeau']
>>> tags = treetaggerwrapper.make_tags(tags)
>>> tags
[Tag(word='beau', pos='ADJ', lemma='beau')]
```

Listing 3.7 – python3

Lorsqu'on trouve un ou plusieurs adverbes ils amplifient le prochain adjectif donc on a instauré un multiplicateur :

```
...
if( y[0:3] == "ADV"):
    ...
    multiplicateur += 0.2
...
```

Listing 3.8 – treetag.py

Enfin lorsqu'on trouve un adjectif on appel SenticNet et on ajoute le multiplicateur qu'on remet à 1

```
...
polarite += multiplicateur * sn.polarity\_value(precedant) #
   sn.polarity_value(precedant) = La polarité du mot
multiplicateur = 1
...
```

Listing 3.9 – treetag.py

3.3.4 TextBlob Français et Anglais

TextBlob est une librairie python qui permet de faire du traitement de langage il utilise une méthode similaire à notre méthode de dictionnaire et celui d'adverbe et d'adjectif, en effet si on va dans le code source de textblob on trouve un fichier .xml qui contient beaucoup de définitions de mots.

✎ exemple : pour le mot "common" on trouve deux entrées sous ce format :

```
0 <word form="common" cornetto_synset_id="n_a-510254"
    wordnet_id="a-00492677" pos="JJ" sense="belonging to or
    participated in by a community as a whole" polarity="0.0"
    subjectivity="0.1" intensity="1.0" confidence="0.8" />
1 <word form="common" cornetto_synset_id="n_a-522137"
    wordnet_id="a-01593079" pos="JJ" sense="of or associated with the
    great masses of people" polarity="-0.6" subjectivity="0.9"
    intensity="1.0" confidence="0.8" />
```

Listing 3.10 – en-sentiment.xml

Si on enlève ce qui nous intéresse pas on se retrouve avec ceci :

```
0 <word form="common" pos="JJ" polarity="0.0" subjectivity="0.1"
    intensity="1.0"/>
1 <word form="common" pos="JJ" polarity="-0.6" subjectivity="0.9"
    intensity="1.0"/>
```

Listing 3.11 – en-sentiment.xml

TextBlob utilise plusieurs définitions du mot et ensuite en fait la moyenne. Il utilise principalement ces 3 valeurs :

- **polarity** : plus ou moins positif/négatif,
- **subjectivity** : si le mot est subjectif ou pas,
- **intensity** : si le mot a une influence sur le prochain (exemple : 1.0 aucun influence , 2.0 double le prochain mot)

Dans notre projet on utilise TextBlob à deux reprises :

- **textblob-fr** : on utilise textblob directement sur du texte français,
- **textblob-en** : on traduit d'abord le texte français en anglais puis on analyse la version anglaise

3.3.5 Classifieur

Le dernier point d'analyse que nous avons traité est sur le Machine Learning, en effet même si nous n'avons pas pu créer notre propre classifieur dû au manque de temps et surtout d'expérience, notre encadrant nous à fourni un classifieur prêt à utilisation.

CHAPITRE

4

FRONT-END DE L'APPLICATION

4.1 Cahier des charges

Nous voulions créer une interface simple et épurée permettant à l'utilisateur d'en comprendre très facilement le fonctionnement. Nous avons donc débattu sur la forme de notre site et nous nous sommes mit d'accord sur une première version très simple composée en deux parties, l'une est une barre horizontale permettant de fournir les informations nécessaires à la recherche des tweets et l'autre l'affichage de chacun d'eux.

Une fois implémentée nous avons amélioré notre site afin qu'il soit plus complet et plus intéressant pour l'utilisateur. Nous savions qu'il était possible lorsque l'on fait un tweet de donner sa position ou le pays ou on est. Par nos recherches on a découvert que Tweepy pouvait nous fournir cette information donc on a décidé de créer une map monde interactive permettant de visualiser mondialement les sentiments de notre recherche.

Pendant la création du site, l'analyse des tweets évoluait elle aussi, elle devenait plus complète de part les différents traitements utilisé, on a donc trouvé très intéressant de créer une nouvelle page fournissant un aspect plus détaillé des différents résultats de chaque traitement.

4.2 Création d'un serveur local

Le but était de créer une barre de recherche permettant de donner 4 informations à notre script python, le mot à rechercher, le nombre de tweet souhaité, la date de début et de fin de recherche. A cela il a fallu gérer les contraintes liées à l'accès de l'API de Twitter, à savoir 180 tweets maximum par recherche et la possibilité de prendre des tweets datant minimum d'une semaine à partir d'aujourd'hui. Enfin, le bouton d'envoie nous permet de faire le lien entre le site web et le serveur python en envoyant les données. Mais pour l'instant il ne sert à rien car sans avoir créé de serveur elles ne peuvent être communiquées au script python, il nous a donc fallu en créer un.

Il existe un module python, Flask, permettant la création d'un serveur en localhost afin que le script javascript et le script python puissent communiquer. Celui-ci permet lors de la soumission (submit) des informations données par l'utilisateur qu'elles soit récupérées par le script python.

✎ exemple : L'utilisateur à recherché **3** tweets comportant le mot **Macron** entre le **5 Avril 2019** et le **11 Avril 2019**. Nous n'allons pas expliquer comment cela fonctionne, il suffit juste de comprendre que les éléments, présent dans le nouvel URL du navigateur ci-dessous sont récupérés par le script python qui va donc s'en servir comme paramètre dans la fonction test.

FIGURE 4.1 – l'URL suite à la recherche

```
0 from flask import Flask, render_template
1 ...
2 #Permet de renvoyer les données de l'utilisateur au script
3 #query = macron, number = 3, dateDebut = 2019-04-05, dateFin = 2019-04-11
4 @app.route('/<query>:<number>:<dateDebut>:<dateFin>')
5 def test(query, number, dateDebut, dateFin) :
6 ...
```

Listing 4.1 – server.py

Il a donc était nécessaire de gérer les erreurs de l'utilisateur, si celui-ci oublie de remplir des champs ou rentre des mauvaises informations, afin d'éviter que le programme ne plante, en voici un exemple :

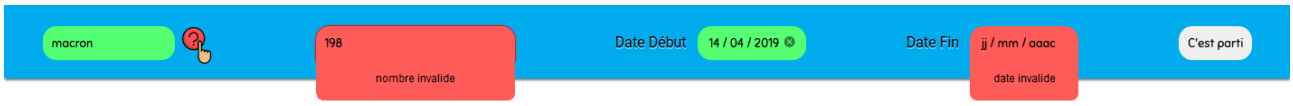


FIGURE 4.2 – Exemple d'erreur lors d'une recherche

4.3 Résultat final

Pour avoir un aperçu global de notre site nous vous invitons à regarder la liste des liens ci-dessous, disponible en annexe, montrant ses différents aspects :

- ✎ Figure 5 page 16 : Aperçu de la page
- ✎ Figure 5 page 17 : Aperçu d'une tooltip de la map monde
- ✎ Figure 5 page 17 : Aperçu de la page offrant plus de détails

CHAPITRE

5

CONCLUSION ET PERSPECTIVES

Connaissances et apprentissages

De part l'aboutissement de notre projet nous avons acquis de nombreuses connaissances dans des langages que nous avions très peu abordés, voire pas du tout, pendant notre parcours universitaire. On a pu aussi découvrir encore de nouvelles techniques pour faciliter le travail de groupe permettant ainsi d'apporter une meilleure qualité de travail.

Le langage Python avait été abordé en système d'exploitation (HLIN303) mais aussi en ISN (Informatique et Science du Numérique) pour certains en terminal. Toutes ces connaissances nous ont permis d'avoir une base plutôt solide pour la conception de notre projet.

HTML et CSS eux ont étaient vus en binaire au web (HLIN102) et, en plus pour certains d'entre nous, en culture général dans la branche concepts et outils de base en informatique (HLSE305).

Les cours de techniques de communication, de conduite de projets (HLIN408) nous à fait découvrir la gestion de projet et nous à permis d'apprendre à utiliser de nombreux outils utiles tels que LaTeX, GitLab et Trello.

Ce projet nous a également permis de développer de nouvelles compétences, en particulier sur la partie web, avec le javascript, et sur la base de donnée. De plus nous avons découvert l'importance des API qui permettent de faciliter grandement le travail des développeurs.

Perspective

L'analyse des tweets étant déjà assez performante, elle aurait pu être amélioré. Avec une base de données conséquente nous aurions pu utiliser le Machine Learning - *technologie d'intelligence artificielle permettant aux ordinateurs d'apprendre sans avoir été programmés explicitement à cet effet* - qui aurait permis de discerner l'ironie qui est l'un de nos plus gros facteurs d'erreurs actuellement. Pour l'interface web le seul bémol est que la recherche avancée de Twitter n'est pas totalement fonctionnelle, une fois réparée elle permettrait une recherche plus sophistiquée pour l'utilisateur.

Annexes

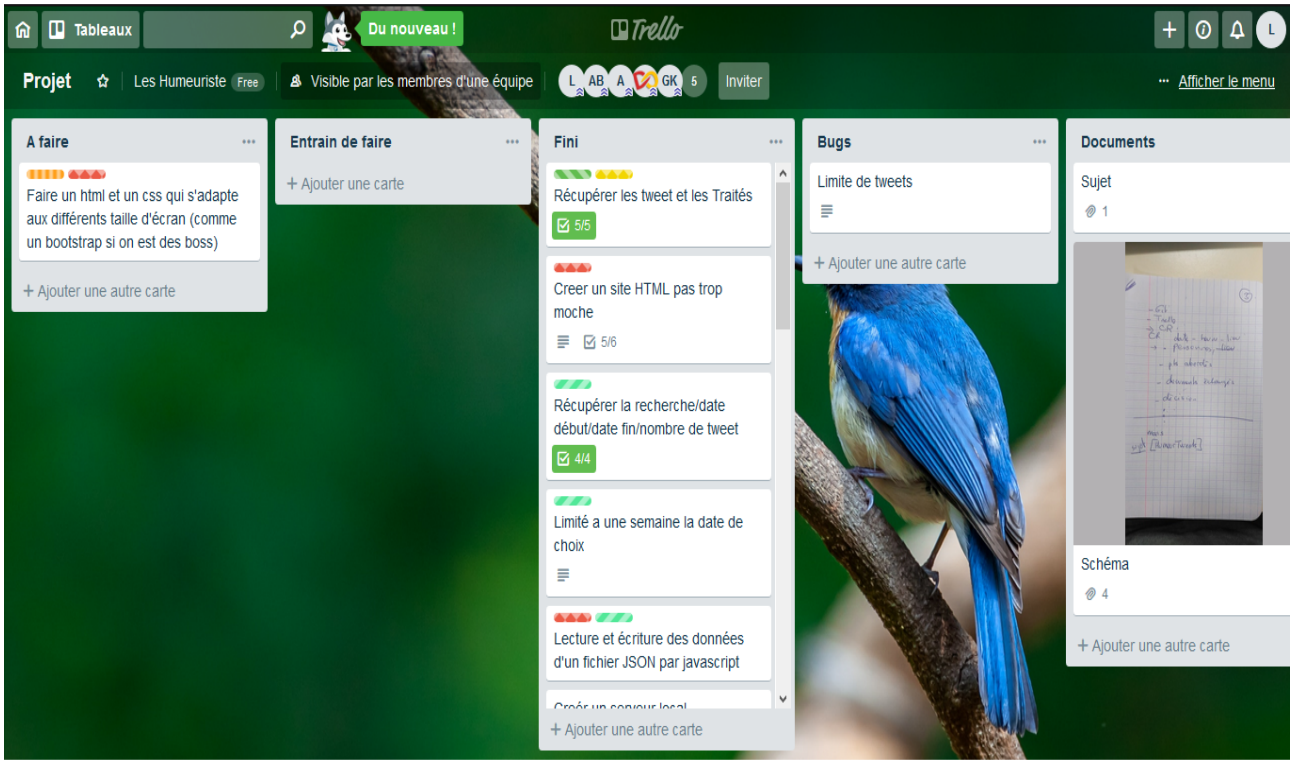


FIGURE 5.1 – Aperçu de notre trello revenir au texte

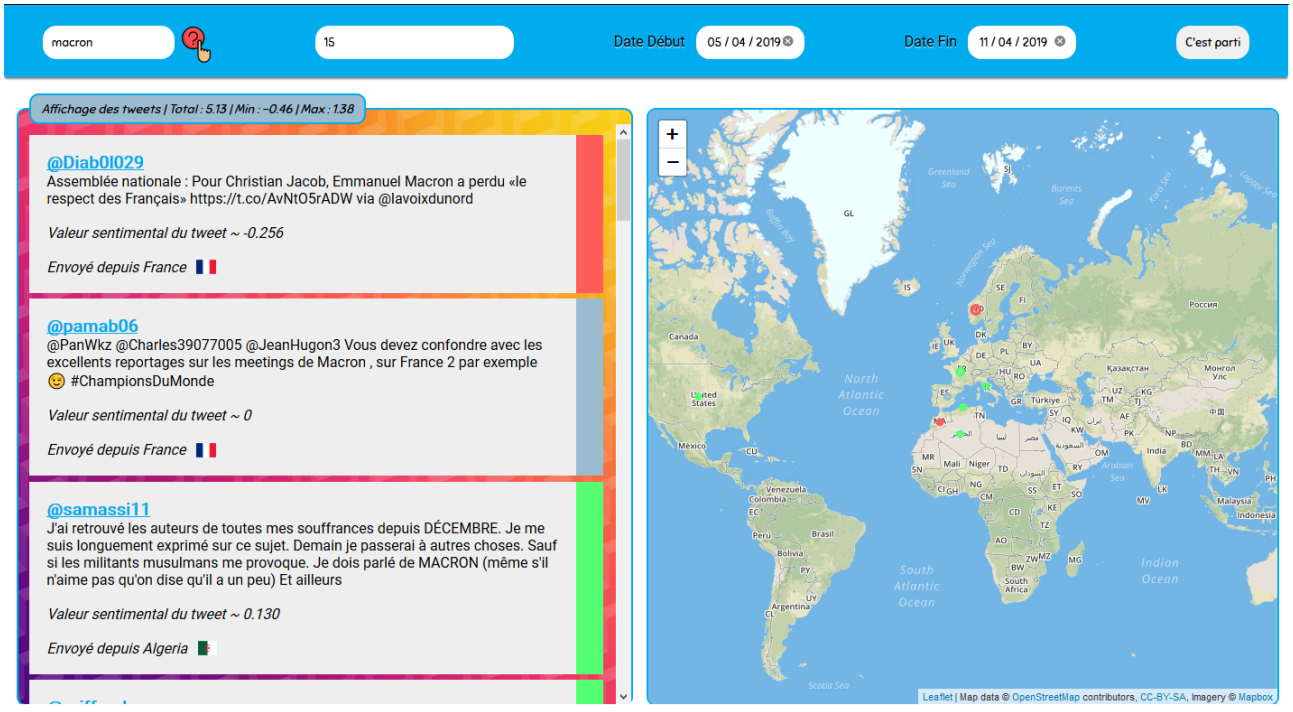


FIGURE 5.2 – Aperçu de la page revenir au texte

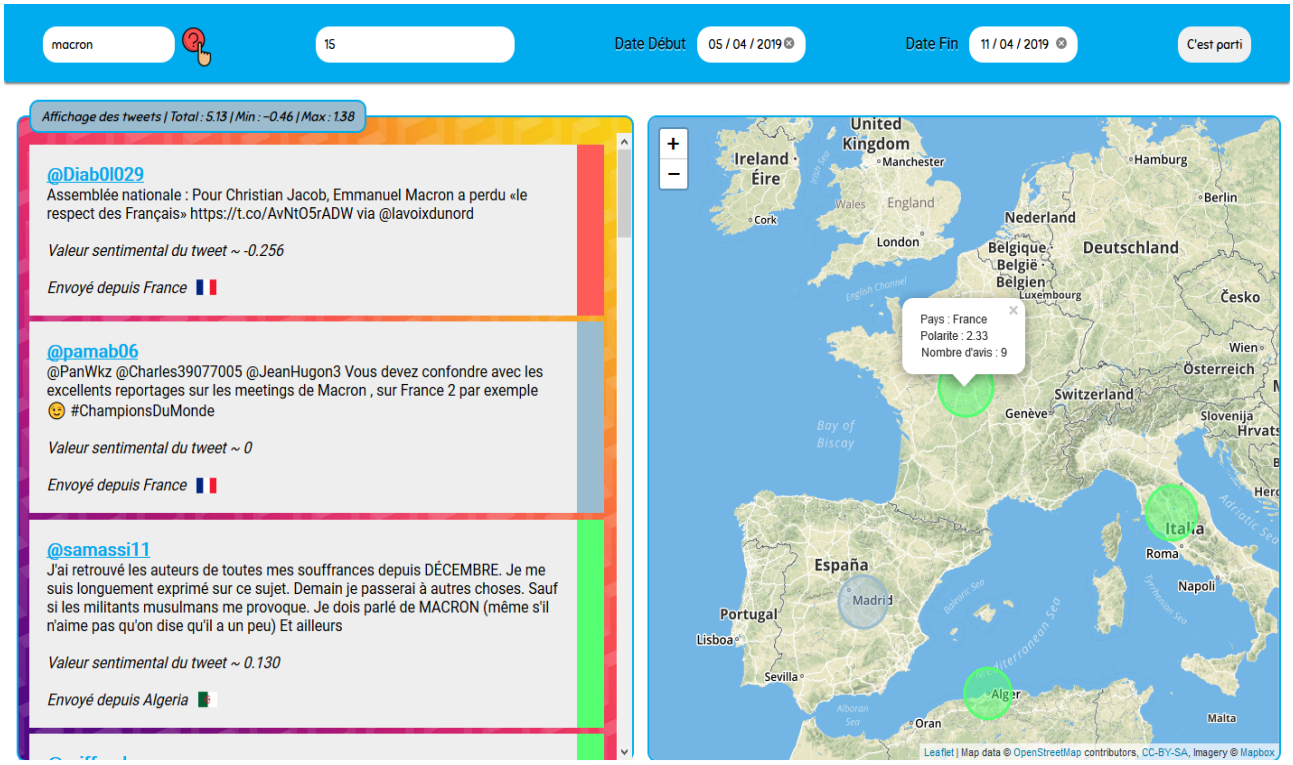


FIGURE 5.3 – Aperçu d'une tooltip de la map monde revenir au texte

FIGURE 5.4 – Aperçu de la page offrant plus de détails, revenir au texte