

UNIVERSITÉ DE MONTPELLIER  
L3 INFORMATIQUE

---

# Natural Disastweet

Localisation d'événements importants grâce à Twitter

---

RAPPORT DE PROJET T.E.R.  
PROJET INFORMATIQUE — HLIN601

**Étudiants :**

M. Jean-Charles ALLA  
M. Mathieu LADEUIL  
M. Florentin DENIS  
Mme. Élodie LABORDE  
M. Khélian LARVET

**Année :** 2020 - 2021

**Encadrant :**

M. Pascal PONCELET



*Nous tenons à remercier notre encadrant M. Pascal PONCELET pour son accompagnement, sa bienveillance et ses conseils précieux tout au long de ce projet qui nous ont permis de le porter à son état actuel. Nous remercions également Pascal LARVET et Julie BADIA, pour leur relecture attentive de ce rapport.*

# Table des matières

|   |           |
|---|-----------|
| <b>Introduction</b>                                   | <b>3</b>  |
| <b>1 Organisation du projet</b>                       | <b>4</b>  |
| 1.1 Attribution et planification des tâches . . . . . | 4         |
| 1.2 Technologies utilisées . . . . .                  | 4         |
| <b>2 Tweets et API</b>                                | <b>6</b>  |
| 2.1 Wrapper . . . . .                                 | 6         |
| 2.1.1 Définition . . . . .                            | 6         |
| 2.1.2 API Twitter . . . . .                           | 6         |
| 2.1.3 Architecture du wrapper . . . . .               | 7         |
| 2.2 Parser . . . . .                                  | 9         |
| 2.2.1 Définition . . . . .                            | 9         |
| 2.2.2 Formatage JSON des données . . . . .            | 9         |
| 2.3 Génération des tweets . . . . .                   | 10        |
| 2.3.1 Fake tweets . . . . .                           | 10        |
| 2.3.2 Nos tweets . . . . .                            | 11        |
| <b>3 Transfert dans MongoDB</b>                       | <b>13</b> |
| 3.1 Sécurisation des échanges . . . . .               | 13        |
| 3.2 Données Transférées . . . . .                     | 13        |
| <b>4 Analyses morphosyntaxiques</b>                   | <b>16</b> |
| 4.1 Analyse des tweets par Spacy . . . . .            | 16        |
| 4.2 Heuristiques d'analyse . . . . .                  | 16        |
| <b>5 Interfaçage web</b>                              | <b>18</b> |
| 5.1 Les mots-clefs . . . . .                          | 18        |
| 5.2 La carte Leaflet . . . . .                        | 18        |
| 5.3 Extraction de Tweets . . . . .                    | 20        |
| 5.4 Validation Backoffice . . . . .                   | 20        |
| 5.5 Administration . . . . .                          | 21        |
| <b>6 Test de montée en charge</b>                     | <b>23</b> |
| 6.1 Affichage sur Leaflet . . . . .                   | 23        |
| <b>7 Bilan et difficultés rencontrées</b>             | <b>25</b> |
| 7.1 Avancement du projet . . . . .                    | 25        |
| 7.2 Changements majeurs . . . . .                     | 25        |
| 7.3 Difficultés rencontrées . . . . .                 | 25        |
| <b>Conclusion</b>                                     | <b>26</b> |
| <b>Annexes</b>  | <b>27</b> |

|                             |           |
|-----------------------------|-----------|
| <b>H Diagramme de Gantt</b> | <b>30</b> |
| <b>I Diagramme UseCase</b>  | <b>31</b> |
| <b>Bibliographie</b>        | <b>32</b> |

# Introduction

Dans le cadre du module de projet informatique du second semestre de L3, nous avons développé le site Disastweet qui est disponible en ligne à l'adresse suivante : <http://disastweet.flareden.fr>. Il permet d'informer les utilisateurs sur les catastrophes naturelles dans le monde grâce à l'**API** de Twitter.

Notre groupe est composé de cinq personnes, Jean-Charles ALLA, Mathieu LADEUIL, Florentin DENIS, Élodie LABORDE, Khelian LARVET et nous sommes encadrés par M. Pascal PONCELET. La réalisation du projet s'est déroulée du 18 janvier au 30 avril 2021.

## Motivations du projet

Notre première motivation sur ce projet a été la possibilité d'analyser lexicalement en profondeur le contenu des tweets pour en tirer des informations précises. Les langues étant des langages non formels avec de multiples règles grammaticales pour exprimer une idée, nous avons été fascinés par les possibilités offertes par Spacy : allant d'une simple reconnaissance par symbole (DET, NOUN, VERB, etc...) à la reconnaissance d'entités complexes (PERSON, GPE, EVENT, etc...).

Le fonctionnement des API a également été un point clé sur notre motivation. Les applications disposant d'une API permettent de communiquer leurs informations à d'autres programmes avec un formatage de données concis. Nous avons voulu comprendre comment pouvait s'effectuer une communication au travers d'une API, surtout celle mise à disposition par Twitter, utilisé environ 500 millions de fois par jour.

De plus, ce projet a capté notre attention sur son aspect dynamique et visuel. En effet, l'affichage web avec une carte interactive aide à maintenir l'attention des utilisateurs sur notre site. De plus les données sont actualisées en temps réel avec l'API de Twitter, ce qui permet de dynamiser notre application. C'est dans cette optique que nous avons choisi de le réaliser en ligne afin de le rendre beaucoup plus accessible.

Il nous semblait donc parfait pour comprendre les rudiments des analyses morphosyntaxiques par l'intermédiaire d'un site visuel et dynamique.

## Objectifs du projet et cahier des charges

Deux objectifs ont été fixés. Le premier a été l'écriture d'un code permettant l'extraction, le stockage et l'analyse de tweets. Le second s'est porté sur la réalisation d'une application web permettant d'afficher les résultats de nos analyses sur les tweets et de les modifier si besoin. Notre encadrant nous a imposé le cahier des charges suivant pour notre application :

- **Créer des heuristiques pour prédire la géolocalisation et les événements des tweets.**
- **Utiliser le système MongoDB pour le stockage des tweets.**
- **Utiliser la librairie Spacy pour l'analyse des tweets.**
- **Utiliser la librairie Leaflet pour la visualisation des tweets sur une carte.**

# Chapitre 1

## Organisation du projet

### 1.1 Attribution et planification des tâches

Nous nous sommes mis d'accord, sur la répartition des rôles selon les préférences de chacun. Deux équipes ont été mises en place afin de travailler de manière efficace :  
**Une équipe Web**, constituée de Élodie et Florentin.  
**Une équipe Python**, avec Charles, Mathieu et Khélian.

Une fois par semaine en utilisant Discord, nous avons pu faire le point sur notre avancement personnel, et par la même occasion, organiser les différents rendez-vous avec notre encadrant. Discord nous permettait également de garder une trace écrite de nos recherches et des points abordés lors des rendez-vous.

Le projet peut être représenté simplement sous la forme d'une chaîne de traitement en 5 étapes :  
**Étape 1** : Reconnaître les tweets associés à des catastrophes naturelles avec l'API Twitter, ayant un indicateur de localisation valide.  
**Étape 2** : Extraire ces tweets dans une base de données MongoDB.  
**Étape 3** : Analyser de manière morphosyntaxique les tweets avec Spacy pour en déduire leur lieu.  
**Étape 4** : Obtenir les coordonnées GPS d'un lieu avec la base Geoname.  
**Étape 5** : Afficher les coordonnées sur une carte Leaflet.

La première phase du projet s'est donc basée sur la recherche des différents éléments constituant notre application. C'est dans cette phase que nous avons pris la décision de mettre en place un serveur permettant d'héberger le projet en ligne. Nous avons également créé un compte Twitter développeur dans l'objectif d'utiliser son API et d'extraire des tweets.

La seconde phase du projet s'est portée sur le développement. L'équipe Python s'est chargée de modéliser un "Wrapper" qui communique avec l'API Twitter, et un "Parser" pour formater les données reçues. Quant à l'équipe Web, elle s'est concentrée sur l'affichage des données, leur véracité, et la communication entre le site et Python.

Enfin, nous avons traqué les bugs résiduels et par la même occasion effectué des tests de montée en charge pour optimiser les ressources utilisées par notre application. Nous détaillerons ces optimisations dans la suite du rapport.

### 1.2 Technologies utilisées

Le choix a été fait en accord avec notre référent de module.  
Nous avons donc utilisé dans notre partie logiciel :

**Python :**

Langage de programmation interprété multiparadigmes et multiplateformes. Nous l'utilisons afin d'établir une liaison avec la base de données MongoDB et d'extraire des tweets grâce à l'API Twitter.

**PHP 7.4 :**

Langage de programmation populaire permettant la production de pages web dynamiques. Il nous permet de créer la liaison entre notre site et notre base MongoDB.

**Symfony :**

Framework PHP, il nous fournit des fonctionnalités modulables et adaptables qui facilitent et accélèrent le développement de notre site web.

**MongoDB :**

Système de gestion de base de données **NoSQL** orientée documents. Il est adapté aux applications de gestion en temps réel et nous permet de stocker nos tweets facilement.

**Spacy :**

Bibliothèque Python de traitement automatique des langues. Elle nous est utile afin d'analyser morphosyntaxiquement nos tweets pour définir s'ils sont valides ou non.

Puis pour l'interface utilisateur nous avons utilisé :

**Leaflet :**

Bibliothèque JavaScript de cartographie en ligne. Nous l'avons intégrée à notre site dans le but d'indiquer visuellement sur une carte où se situent les catastrophes naturelles relatives aux tweets.

**Bootstrap :**

Framework CSS utile à la création du design des sites. Il nous aide à accélérer la mise en page de notre site.

**JQuery UI :**

Collection de widgets, effets visuels et thèmes implémentés avec jQuery. Nous l'utilisons également pour accélérer la mise en page de notre site.

Enfin, pour rester coordonnés nous avons utilisé :

**Discord :**

Logiciel de VoIP et messagerie instantanée gratuite fonctionnant par serveurs. Il nous permet de communiquer au sein de l'équipe, de s'entraider, et d'avoir une trace de nos discussions.

**GitHub :**

Service d'hébergement et de gestion de développement informatique en ligne, bâti sur le logiciel de versionnage Git. Il nous permet de garder une trace de chaque version de fichiers en ligne, lors du développement de notre projet.

# Chapitre 2

## Tweets et API

### 2.1 Wrapper

#### 2.1.1 Définition

Un wrapper est un framework qui encapsule les appels à une Application Programming Interface (API) dans une couche d'abstraction pour faciliter son utilisation. L'utilisation d'un wrapper dédié à une certaine API va pouvoir récupérer les données sans avoir à gérer lui-même les requêtes et leurs comportements.

Les fonctions du wrapper ne correspondent pas forcément à l'appel d'un unique **end-point**. L'avantage du wrapper est double : nous pouvons combiner plusieurs appels à des end-points différents pour en récupérer des données précises, et nous pouvons également limiter le nombre d'appels effectués.

En effet, dans le cadre de ce projet, nous utilisons l'API de Twitter v2.0. Celle-ci possède 2 limites relatives à notre compte de développeur : une limite pour une fenêtre de 15 minutes et une autre mensuelle. Le wrapper permet donc de respecter cette limite de façon transparente pour l'utilisateur. Tout comme l'authentification requise par l'API.

Enfin cela nous permet également d'utiliser un système de cache afin de stocker et d'optimiser l'appel des requêtes les plus souvent appelées et d'optimiser le temps d'exécution des requêtes les plus récurrentes.

#### 2.1.2 API Twitter

Twitter a sorti une nouvelle version de son API (le 12 Octobre 2020), en accès anticipé nommée "Twitter API v2.0". Bien que ne disposant pas encore de tous les end-points de la version 1, elle possède des accesseurs en lecture qui sont essentiels pour le bon fonctionnement de notre application. De plus, elle possède de nombreuses fonctionnalités (figure 2.1) telles que :

- Le choix des champs des données dans les réponses.
- De nouveaux champs de données.
- Des annotations d'entités telles que "Place" (lieu d'un tweet) et des contextes.
- Plus de 20 opérateurs pour filtrer les requêtes.

|                        |   |
|------------------------|---|
| <b>Tweet lookup</b>    | Look up Tweets by ID.   |
| <b>User lookup</b>     | Look up users by name or ID.  |
| <b>Recent search</b>   | Query the most recent seven days of Tweets, and receive full-fidelity responses with this first release of our search Tweets functionality. |
| <b>Timelines</b>       | Returns the Tweets composed by, or mentioning, a specified Twitter account.   |
| <b>Filtered stream</b> | Filter the complete stream of real-time public Tweets.  |
| <b>Sampled stream</b>  | Stream a sample of new Tweets as they are published, across ~1% of all public Tweets in real-time.  |
| <b>Hide replies</b>    | Hides or unhides a reply to a Tweet.  |
| <b>Follows lookup</b>  | Retrieve an account's followers and who they are following using their user ID.   |

FIGURE 2.1 – Fonctionnalités Twitter API v2.0

### 2.1.3 Architecture du wrapper

Après avoir étudié plusieurs wrappers disponibles sur GitHub, notamment le wrapper développé par l'équipe de Twitter, nous avons constitué le nôtre. Nous avons gardé les modules que nous avons jugé important tout en restant simple et efficace. Notre wrapper permet aussi la gestion des exceptions et de l'authentification.

#### Authentification

L'authentification dans notre wrapper se fait au moyen d'un **Bearer Token** et déclenche une exception s'il n'est pas fourni :

```

1 if bearer_token:
2     self.headers = self._create_headers(bearer_token)
3 else:
4     raise AuthenticationError("The wrapper.Api instance must be authenticated.")

```

Ensuite nous créons un header pour nos requêtes avec le Bearer Token afin d'être autorisés à recevoir des réponses à nos requêtes dans l'API Twitter :

```

1 headers = {"Authorization": "Bearer {}".format(bearer_token)}

```

#### Cache des requêtes

Le **cache** de nos requêtes est réalisé grâce à la librairie "requests\_cache" qui s'utilise très simplement ainsi :

```

1 requests_cache.install_cache('wrapper_cache')

```

Cette librairie s'occupe ensuite du cache sans que nous ayons à intervenir. Notons que pour les requêtes de **stream**, il nous faut désactiver temporairement le cache sinon nous ne pouvons pas avoir de résultats :

```

1  with requests_cache.disabled():
2      response = requests.request("GET", url, headers=self.headers, stream=True)

```

### Limitation des appels

La limitation des appels des requêtes est réalisée grâce à la classe RateLimit. Celle-ci contient les RateLimit des 6 end-points que nous utilisons :

```

1  from collections import namedtuple
2
3  EndpointRateLimit = namedtuple('EndpointRateLimit', ['limit', 'remaining',
4      ↵  'reset'])
5
6  class RateLimit(object):
7      """RateLimit class for Twitter api V2.0"""
8
9      def __init__(self):
10          """ Instantiates a RateLimit Object """
11          self._rate_limit = {}
12          self._rate_limit["recent_search"] = EndpointRateLimit(180,180,0)
13          self._rate_limit["tweet"] = EndpointRateLimit(300,300,0)
14          self._rate_limit["sample_stream"] = EndpointRateLimit(50,50,0)
15          self._rate_limit["search_stream"] = EndpointRateLimit(50,50,0)
16          self._rate_limit["rules"] = EndpointRateLimit(450,450,0)
17          self._rate_limit["post_rules"] = EndpointRateLimit(450,450,0)

```

Les RateLimit sont codés en "dur" pour plus de clarté et parce qu'il n'est pas prévu d'utiliser d'autres appels dans notre wrapper. De plus nous avons ajouté des méthodes pour connaître le nombre d'appels encore disponible sur nos end-points.

### Exceptions

Nous avons déclaré 3 types d'exceptions assez explicites :

- Une pour les erreurs liées à Twitter.
- Une pour les erreurs liées à l'authentification.
- Une pour les erreurs liées aux requêtes.

```

1  class TwitterError(Exception):
2      """Base class for Twitter errors"""
3
4  class AuthenticationError(Exception):
5      """Base class for Authentication errors"""
6
7  class RequestError(Exception):
8      """Base class for Request errors"""

```

### Utilisation du wrapper

Pour utiliser le wrapper après l'avoir importé, il suffit de créer un objet Api avec deux paramètres. Le premier étant le Bearer Token sous la forme d'une chaîne de caractères et le second est un booléen

pour spécifier si l'on utilise le parser ou non.

Ensuite, comme dans un moteur de recherche classique, nous définissons des mots-clefs (fields) que l'on veut récupérer dans Twitter. Nous pouvons optionnellement ajouter une chaîne de caractères (query) à rechercher en plus des mots-clefs.

L'utilisateur peut ainsi facilement accéder à l'API Twitter avec les méthodes définies ci-dessous :

- api.get\_recent\_search(query, fields)
- api.get\_tweet(ids, fields)
- api.get\_filtered\_stream(fields)
- api.get\_sample\_stream(fields)

Voici un exemple de code en Python :

```

1 from wrapper import Api
2
3 BEARER_TOKEN = "BEARER_TOKEN"
4
5 api = Api(BEARER_TOKEN, True)
6
7 tweet_fields = "expansions=geo.place_id&tweet.fields=geo,entities,author_id"
8 max_results = 50
9 max_pages = 2
10
11 search = "'Floodings'"
12 query = search+"&max_results="+str(max_results)
13 response = api.get_yielded_recent_search(query,tweet_fields,max_pages)
14
15 for tweets in response:
16     with open("test.json", "w", encoding='utf-8') as outfile:
17         json.dump(tweets, outfile)

```

## 2.2 Parser

### 2.2.1 Définition

Le parser est la partie de l'application qui va reformater et analyser les données JSON récupérées depuis Twitter. En effet il y a beaucoup de champs non pertinents qui sont envoyés par Twitter. Dans un premier temps le parser fait le tri et sélectionne les champs importants tels que "geo" et "places", correspondant respectivement aux coordonnées géographiques du tweet et de son emplacement.

Ensuite nous ajoutons nos propres champs pour classer nos tweets selon plusieurs critères. Nous aborderons plus en profondeur dans le chapitre 3 la signification de chaque champ.

Enfin nous pouvons analyser plus en profondeur le champ "text" de Twitter qui correspond au texte du tweet.

### 2.2.2 Formatage JSON des données

Voici un exemple des données JSON récupérées sans formatage par notre parser :

```

1  {
2      "data": [
3          {
4              "author_id": "1088136821928800256",
5              "geo": {
6                  "place_id": "60c390624fa83c29"
7              },
8              "id": "1358020945215127553",
9              "text": "Je suis sur Montpellier et il y a une tempête à Marseille"
10         }
11     ],
12     "includes": {
13         "places": [
14             {
15                 "full_name": "Castelnau-le-Lez, France",
16                 "id": "60c390624fa83c29"
17             }
18         ]
19     }
20 }
```

Voici un exemple des données JSON récupérées avec formatage par notre parser :

```

1  {
2      "tweets": [
3          {
4              "author_id": "1088136821928800256",
5              "geo": "NULL",
6              "id": "1358020945215127553",
7              "place": "NULL",
8              "place_user": "Castelnau-le-Lez, France",
9              "real": "True",
10             "text": "Je suis sur Montpellier et il y a une tempête à Marseille",
11             "valid": "True"
12         }
13     ]
14 }
```

## 2.3 Génération des tweets

### 2.3.1 Fake tweets

Les "FakeTweets" représentent des semblants de tweets au format JSON générés avec l'aide de la librairie Faker proposé par Python. Nous les avons utilisés pour tester nos analyses morphosyntaxiques et l'affichage sur le site web. Ces FakeTweets sont également utilisés pour les tests de montée en charge, nous y reviendrons dans le chapitre 6.

Pour les créer, nous utilisons un fichier TSV fourni par Geoname regroupant les différents lieux ainsi que leurs coordonnées géographiques. À chaque création de FakeTweet, le programme choisit une ligne aléatoirement dans ce fichier pour en extraire le nom et les coordonnées de la ville.

```

1 def rand_geo():
2     file = open("US.tsv")
3     num_lines = sum(1 for line in file)
4     choice = random.randint(0,num_lines)
5     file.close()
6
7     line = linecache.getline("US.tsv",choice)
8     tab_geo = line.split('\t')
9     return tab_geo

```

Ensuite, nous utilisons la méthode fake.sentence() de Faker pour générer des phrases aléatoires à partir d'un tableau de mots-clefs et d'un tableau de connecteurs :

```

1 connecteur = ["at", "near to", "on", "placed at", "situated at", "in",
2   ↵ "appearing in", "next to", "close to", ...]
3 hashtag = ["Avalanche", "Cataclysm", "Catastrophe", "Cyclones",
4   ↵ "Earthquakes", "Eruption", ...]
5
6 rand_c = random.choice(connecteur) + " " + geo[2]
7 rand_h = '#' + random.choice(hashtag)
8
9 text_array = [fake.sentence(), fake.sentence(), rand_h, rand_c]
random.shuffle(text_array)

```

Enfin, il ne nous reste plus qu'à assembler toutes ces données dans un format JSON :

```

1 def fake_tweet_creation():
2     infos = rand_text_longlat()
3     tweet_data = {
4         "_id": rand_ID(),
5         "geo": infos[2],
6         "place": "NULL",
7         "place_user": infos[1],
8         "real": "False",
9         "text": infos[0],
10        "valid": "?"
11    }
12    return tweet_data;

```

### 2.3.2 Nos tweets

Nous avons composé des tweets de test sur Twitter via nos comptes personnels. 5 niveaux de difficultés ont été instaurés afin de créer des heuristiques avec de bonnes prédictions sur chaque niveau :

- **Niveau 1** : un lieu simple
- **Niveau 2** : plusieurs lieux possibles
- **Niveau 3** : position imprécise
- **Niveau 4** : deux villes concernées par un événement
- **Niveau 5** : négations et métaphores

Ils nous ont donc servi à produire des données d'entraînement pour nos heuristiques, mais aussi à vérifier la fonctionnalité de l'extraction des tweets. Toutes les informations ont été stockées sur le lien GoogleSheet (figure 2.2) suivant :

<https://docs.google.com/spreadsheets/d/1GH-Mcm2VHY8cEqcsgENUGcPTMCh8aw-AJKBfJDTIUNs/edit#gid=0>

| #Thunderstorms, as well as intense #Hailstorms affected the center of #Orleans.  | 1 | @AbraTERL3     | Orleans               | Hailstorms       | Oui |
|--|---|----------------|-----------------------|------------------|-----|
| Alert near rennes! a storm is coming   | 1 | @AbraTERL3     | Rennes                | Storm            | Oui |
| I'm in Nimes, there is a #Earthquake at Paris !  | 2 | @Hayao_Yorusat | Paris                 | Earthquake       | Oui |
| Lyon - a #Twister in the streets !   | 2 | @Hayao_Yorusat | Lyon                  | Twister          | Oui |
| #Cyclones Poll 2021  | 2 | @Hayao_Yorusat | Poil                  | Cyclones         | Oui |
| I'm in Toronto but there is snowstorm near Mexico city   | 2 | @Doublepups    | Mexico city           | snowstorm        | Oui |
| I live in China look so much lava is flooding at Tokyo that's not cool   | 2 | @Doublepups    | Tokyo                 | flood ? lava ?   | Oui |
| Moscow : people try to escape earthquakes but it's so fast   | 2 | @Doublepups    | Moscow                | earthquakes      | Oui |
| Casablanca : a gigantic avalanche coming from the nearest mountains.   | 2 | @Doublepups    | Casablanca            | avalanche        | Oui |
| I believe the #cyclone are chasing me. The last time that was in Man, Now it's in Casablanca                                   | 2 | @alphaina9     | Casablanca            | cyclone          | oui |
| Cyclone present on California!!! run to New York   | 2 | @KrisEvior     | California            | Cyclone          | oui |
| I admire the tornadoes of Reims from Narbonne #tornado #storm  | 2 | @AbraTERL3     | Reims                 | Tornado          | Oui |
| You can feel the earthquakes of Marseille from Montpellier! #earthquake  | 2 | @AbraTERL3     | Marseille             | Earthquake       | Oui |
| like Paris, Montpellier had severe floods #Extreme Precipitation #flooding   | 2 | @AbraTERL3     | Montpellier           | Flood            | Oui |
| the #Nantes newspaper analyzes the explosion of the volcano near Narbonne #Eruptions   | 2 | @AbraTERL3     | Narbonne              | Volcano          | Oui |
| a snow avalanche trapped a ski resort in the direction of Grenoble. Fortunately all is well in Chambéry #Avalanche.            | 2 | @AbraTERL3     | Grenoble              | Avalanche        | Oui |
| I am in Nantes and there are heavy rains near Rennes #Flooding #Extreme Precipitation  | 2 | @AbraTERL3     | Rennes                | Flood            | Oui |
| an earthquake shakes Dijon without touching Paris #earthquakes   | 2 | @AbraTERL3     | Dijon                 | Earthquake       | Oui |
| Avalanches in Strasbourg #Avalanche! Do not hesitate to come and see me on Angers :)   | 2 | @AbraTERL3     | Strasbourg            | Avalanche        | Oui |
| Strong winds and heavy rain around the North of France. #storm   | 3 | @AbraTERL3     | France                | Storm            | Oui |
| A very strong #sandstorm affected the South of France, severely disrupting the lives of the population.                        | 3 | @AbraTERL3     | France                | Sandstorm        | Oui |
| Between Lyon and Limoges is a solar eruption!  | 4 | @KrisEvior     | Lyon/Limoges          | Eruption         | non |
| Heavy rains and thunderstorms have broken out over Narbonne and Perpignan  | 4 | @AbraTERL3     | Narbonne & Perpignan  | Thunderstorm     | Oui |
| Extreme rains caused major flooding in Marseille and Toulon  | 4 | @AbraTERL3     | Marseille & Toulon    | Flood            | Oui |
| The forests of Bordeaux and Libourne are on fire! #Wildfires   | 4 | @AbraTERL3     | Libourne & Bordeaux   | Wildfire         | Oui |
| A tidal wave hit Marseille and Nice #TidalWave   | 4 | @AbraTERL3     | Marseille & Nice      | Tidalwave        | Oui |
| Metz and Nancy trembled last night #Earthquakes #Catastrophe #Seismic Wave   | 4 | @AbraTERL3     | Metz & Nancy          | Earthquake       | Oui |
| The #Tsunami in Toulouse is FAKE   | 5 | @Hayao_Yorusat | Toulouse              | Tsunami          | Non |
| I'm fan of #Tornadoes, and i live in Monaco  | 5 | @Hayao_Yorusat | Monaco                | Tornadoes        | Non |
| There is no #Flood in Nimes  | 5 | @Hayao_Yorusat | Nimes                 | Flood            | Non |
| Seismic waves are dangerous be careful people at home.   | 5 | @Doublepups    | None                  | Seismic waves    | Non |
| I live in Montpellier and I like to tweet about natural disasters.   | 5 | @Doublepups    | Montpellier           | Natural Disaster | Non |
| I tweet sometimes but not in Italy this is my Storm cat @alphaina9 <a href="https://t.co/HN5pZtVtg">https://t.co/HN5pZtVtg</a> | 5 | @Doublepups    | Italy                 | Storm            | Non |
| Before leaving Marseille for Amsterdam, I witnessed a #floodwater  | 5 | @alphaina9     | Marseille, Amstersdam | floodwater       | non |
| I read somewhere there is a Flood in Limoges but there isn't !   | 5 | @KrisEvior     | Limoges               | California       | oui |
| I dream about a Tropical Cyclone in France just to live somes adrenaline !   | 5 | @KrisEvior     | France                | Tropical Cyclone | non |
| do you think it's possible to get a #landfall in Gqeberha ?  | 5 | @KrisEvior     | Gqeberha              | Landfall         | non |

FIGURE 2.2 – Nos tweets

# Chapitre 3

## Transfert dans MongoDB

Le SQL étant utilisé dans la gestion des bases de données relationnelles, il peut se montrer insuffisant lorsqu'on évoque des problèmes liés aux 3V (Volume, Velocity, and Variety).

Le "volume" fait référence à la quantité de données, la "variété" représente le nombre de types de données utilisable et la "vitesse" indique la rapidité du traitement des données.

C'est dans ce contexte que la famille des langages NoSQL devient intéressante. En effet elle nous propose une autre manière d'interroger les données, mais aussi de les stocker.

Nous avons opté pour MongoDB qui est une base de données NoSQL orientée documents. Elle est très bien adaptée aux applications de gestion en temps réel.

### 3.1 Sécurisation des échanges

MongoDB prend en charge le protocole **Transport Layer Security (TLS)**, pour crypter tout son trafic réseau. En quelques mots, TLS est un protocole de sécurisation des échanges par réseau informatique, notamment par internet. Il assure la garantie que le trafic des informations ne se fasse que entre la base de données et le client.

PyMongo est un module Python permettant de travailler avec MongoDB. Il nous a servi pour :

- Les processus de connexion à la base de données "Disastweet"
- Sélectionner la collection "Spacetweets"
- Transférer des informations au format **JSON** dans la collection.

Le protocole TLS est spécifié lors de la connexion à la base de données

### 3.2 Données Transférées

**Les informations sont stockées sous forme de documents.** Par défaut, les tweets sont sauvegardés avec un identifiant généré automatiquement par MongoDB. Cet identifiant peut être retrouvé avec la clef "`_id`". Afin d'éviter la génération des doublons dans la base de données, l'identifiant du tweet a été affecté comme valeur à la clef "`_id`".

Voici toutes les informations contenues dans un document :

- `_id` : identifiant du tweet.
- `place` : emplacement géographique ("zone") basé sur des probabilités ("proba").
- `geo` : coordonnées géographiques de la zone d'émission du tweet.
- `valid` : évaluation de la véracité du tweet. La valeur est à "True" ou "False" après l'évaluation du tweet par un utilisateur. Sinon la valeur est à "?" pour indiquer un tweet non évalué.
- `place_user` : emplacement défini par l'auteur du tweet.
- `real` : évaluation de la provenance du tweet. La valeur est à "True" si le tweet a été extrait de Twitter et "False" dans le cas contraire (un "fake tweet" créé par nos programmes).

- **text** : corps du tweet.
- **author\_id** : identifiant de l'auteur du tweet.
- **spacy** : informations obtenues depuis Spacy en analysant le tweet. Il contient :
  - **nouns** : noms figurant dans le tweet.
  - **events** : événements (catastrophes naturelles) dans le tweet.
  - **candidates** : lieux candidats pour l'événement en question.

Voici un exemple de tweet non évalué :

```

1 {
2   "_id": "1389682777231183877",
3   "place": [
4     "zone": "Nepal",
5     "proba": 0.7669
6   ],
7   "geo": "NULL",
8   "valid": "?",
9   "place_user": "NULL",
10  "real": "True",
11  "text": "Quake info: Light mag. 4.8 earthquake - Tibet, 78 km north of
→ Kathmandu, Province 3, Nepal, on Tuesday, 4 May 2021 1:58 pm (GMT +8) -
→ VolcanoDiscovery https://t.co/nmw0jHh6AB",
12  "author_id": "1205549473",
13  "spacy": {
14    "nouns": ["Quake info", "Light", "4.8 earthquake - Tibet", "Kathmandu",
→ "Province", "Nepal", "Tuesday", "GMT +8", "VolcanoDiscovery"],
15    "events": [],
16    "candidates": ["Tibet", "Kathmandu", "Nepal"]
17  }
18 }
```

Une fois qu'un tweet est évalué par un utilisateur, les informations suivantes viennent compléter le document concerné :

- **validation** : après l'évaluation d'un tweet par un utilisateur, de nouvelles informations apparaissent telles que "places" (contenant elle-même le nom de la ville ("place")) ainsi que ses coordonnées GPS ("lng", "lat") et "events" (nom de la catastrophe naturelle).
- **reported\_by** : identifiant de l'utilisateur ayant évalué le tweet.

Voici un exemple de tweet valide évalué par "user@user.fr" :

```
1  {
2      "_id": "1375140229699485700",
3      "place": [
4          "zone": "Montpellier",
5          "proba": 0.9138
6      ],
7      "geo": "NULL",
8      "valid": "true",
9      "place_user": "NULL",
10     "real": "True",
11     "text": " The risk of tsunamis has been increased in recent years at
12         ↪ Montpellier",
13     "author_id": "1366442435753234440",
14     "spacy": {
15         "nouns": ["The risk", "tsunamis", "recent years", "Montpellier"],
16         "events": ["Tsunamis"],
17         "candidates": []
18     },
19     "reported_by": ["user@user.fr"],
20     "validation": {
21         "places": [
22             "place": "montpellier",
23             "lng": "3.87635",
24             "lat": "43.61093"
25         ],
26         "events": ["tsunamis"]
27     }
}
```

# Chapitre 4

## Analyses morphosyntaxiques

### 4.1 Analyse des tweets par Spacy

Pour analyser les tweets, nous utilisons Spacy qui est une bibliothèque Python de traitement automatique des langues. Elle nous permet de partitionner le texte en ce qu'on appelle des tokens, qui seront labélisés avec leurs fonctions grammaticales ainsi que leur "named entity" (figure 4.1). Les entités nommées sont déterminées selon le contexte et correspondent à des catégories d'entités. Ces catégories sont, par exemple, un nom de ville ou de pays (GPE), ou bien le nom d'une personne (PERSON).

Ousted **WeWork** founder **Adam Neumann** lists his **Manhattan** penthouse for **\$37.5 million**

[organization]      [person]      [location]      [monetary value]

FIGURE 4.1 – Exemple de tokenisation

Ainsi, nous pouvons extraire les localisations grâce à Spacy en sélectionnant dans les textes des tweets, les tokens labelisés avec l'étiquette **GeoPolitical Entity (GPE)**.

Finalement l'objet JSON passé dans notre parser aura un nouveau champ "spacy" dans lequel nous mettrons les entités GPE trouvées par Spacy. Ces GPE sont des candidats possibles pour définir le lieu où se situe l'événement catastrophique, et sont répertoriées dans le dictionnaire "candidates".

Nous ajoutons également dans ce champ "spacy" les dictionnaires :

- "**nouns**", qui est simplement la liste de tous les noms que Spacy a trouvé.
- "**events**", qui est la liste des événements que l'on a construit en comparant les "nouns" avec notre propre liste de mots-clefs. Cela nous permet de classifier nos tweets par catastrophe.

### 4.2 Heuristiques d'analyse

Nous avons mis en place deux heuristiques dans notre classe "parser" pour prédire la localisation de nos tweets :

Notre première heuristique "heuristicEmptyCandidate", consiste à ajouter des candidats pour la localisation d'un tweet (dans la partie "spacy" du dictionnaire). Cet ajout a lieu si la liste des candidats de Spacy est vide et si l'API de Twitter possède les informations relatives à l'emplacement du tweet (accessible depuis la clé "place").

```
1 def heuristicEmptyCandidate(self, tweet):
2     if not tweet["spacy"]["candidates"]:
3         if isinstance(tweet["place"], list):
4             tweet["spacy"]["candidates"] = [x['zone'] for x in tweet["place"]]
```

La seconde heuristique "heuristicSameCandidate", permet de ne garder que les emplacements à la fois indiqués par l'API Twitter et par Spacy.

```
1 def heuristicSameCandidate(self, tweet):
2     new_candidates = []
3     for candidat in tweet["spacy"]["candidates"]:
4         for elem in tweet["place"]:
5             if candidat == elem['zone']:
6                 new_candidates.append(candidat)
7     tweet["spacy"]["candidates"] = new_candidates
```

# Chapitre 5

## Interfaçage web

### 5.1 Les mots-clefs

Les **mots-clefs** sont essentiels et sont définis par l'utilisateur. Ceux-ci sont reliés aux modules "Carte" et "Extraction", permettant de définir quels mots-clefs l'utilisateur désire voir sur sa carte, ou extraire de Twitter s'il possède une **clef Twitter**.

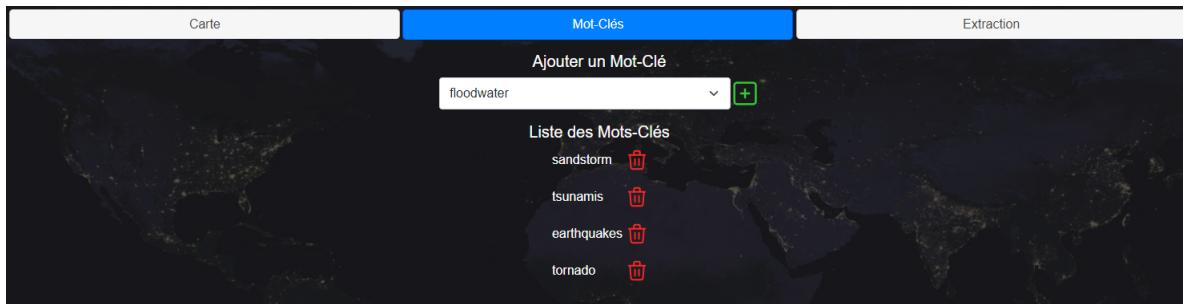


FIGURE 5.1 – Choix des mots-clefs

### 5.2 La carte Leaflet

Nous avons utilisé la librairie Leaflet afin de visualiser la géolocalisation de nos tweets. Chaque tweet possède un mot-clef permettant de les répertorier et de les afficher par catégorie de catastrophe.

Pour des raisons de performance et de clarté, nous avons mis en place une carte thermique grâce aux différentes fonctionnalités offertes par Leaflet.

Chaque **tweet valide** est un marqueur cliquable (figure 5.3) sur la carte permettant d'afficher ses informations relatives. En fonction du niveau de zoom sur la carte, les marqueurs sont remplacés par des zones (figure 5.4) respectant un code couleur pour indiquer le niveau de concentration des marqueurs dans la zone (Rouge - Jaune - Vert - Bleu).

Nous pouvons ainsi observer la concentration des tweets en fonction des lieux et par conséquent faire des suppositions sur la véridicité de l'événement et de son ampleur.

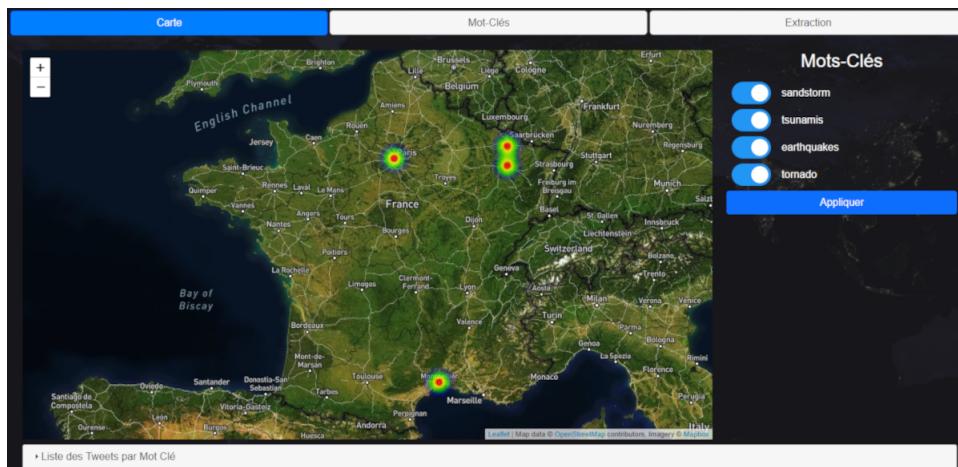


FIGURE 5.2 – Carte

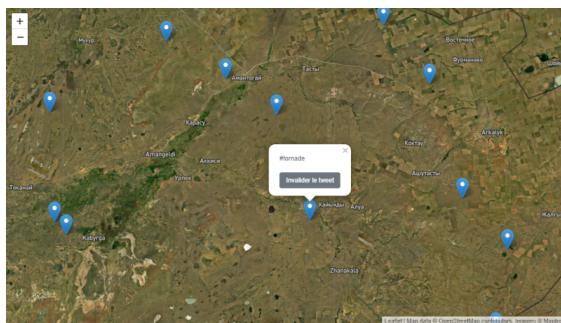


FIGURE 5.3 – Marqueurs

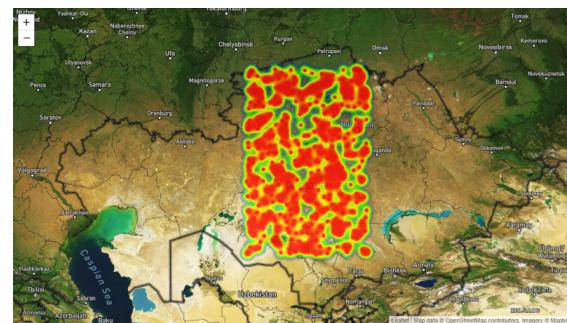


FIGURE 5.4 – Carte thermique

Voici un extrait de la commande permettant de définir la carte thermique en fonction du zoom :

```

1  this.map.on('zoomend', function (value) {
2      let zoom = value.target.getZoom();
3      if (zoom >= zoomMarker) {
4          isHeatmap = false;
5      } else {
6          isHeatmap = true;
7      }
8      mtc.heatmapLayer.cfg.radius = initialRadius - (zoom * radiusPas);
9      mtc.showDatas(mtc.lastKeywords);
10 });

```

Une fonction de signalement a également été ajoutée à nos marqueurs cliquables (figure 5.5) pour pouvoir signaler les erreurs. En effet, la vérification des tweets est effectuée par des heuristiques et ne sont pas justes à 100%, il faut donc pouvoir remettre en question notre système.

Cette fonctionnalité n'est pas à la portée de tous pour éviter qu'un utilisateur lambda puisse perturber notre système. Nous verrons plus bas les différents droits pour chaque rôle dans la partie "Administration".

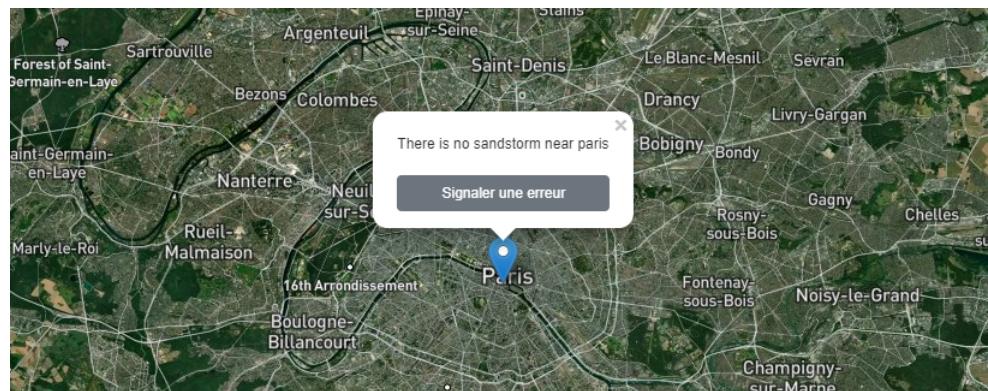


FIGURE 5.5 – Signalement d'un tweet

### 5.3 Extraction de Tweets

Ce module permet à un utilisateur ayant une clef Twitter, d'extraire des tweets dans tout Twitter. L'extraction se fait en fonction des mots-clefs préalablement choisis, agissant comme des filtres sur la requête API. Les tweets ainsi extraits sont automatiquement traités par nos programmes et peuvent être directement affichés sur la carte.

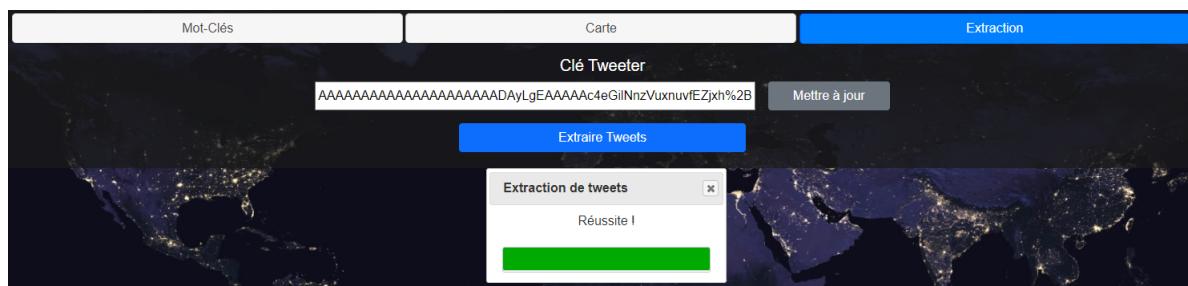


FIGURE 5.6 – Extraction des tweets

### 5.4 Validation Backoffice

Nous avons réalisé un système de vérification de tweets permettant à des personnes avec un rôle spécifique, de vérifier les tweets récupérés depuis Twitter.

Ces personnes sont qualifiées de "validateur" et ont la possibilité de vérifier les tweets qui n'ont pas encore été validés. Cette vérification implique de choisir un lieu et un événement selon les propositions obtenues par Spacy ou en sélectionnant un élément dans la phrase. (figure 5.7)

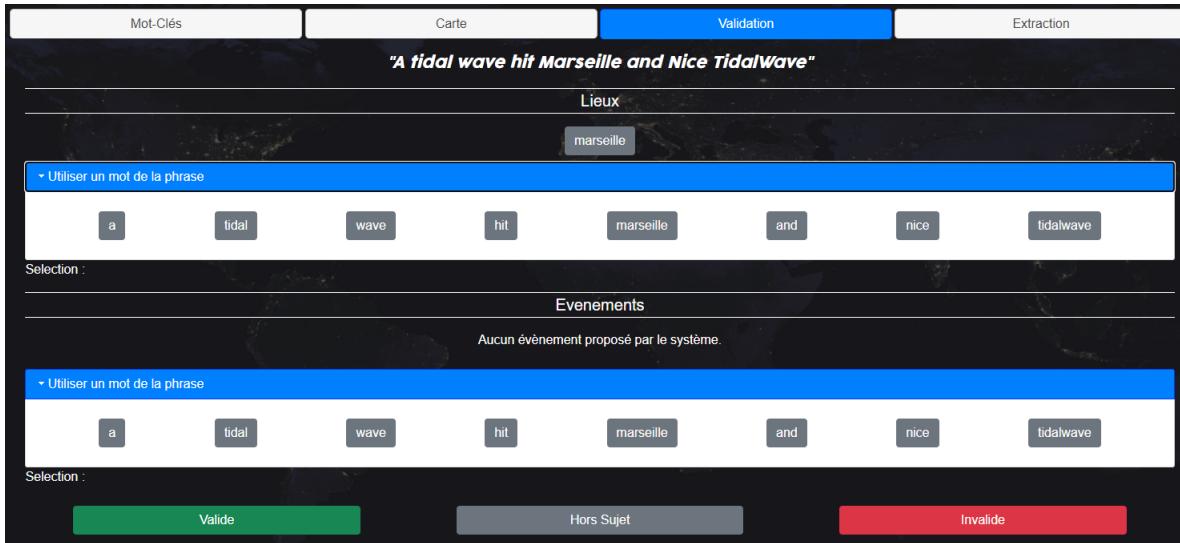


FIGURE 5.7 – Validation d'un tweet

De ce fait, ces personnes peuvent choisir si le tweet est **valide avec un lieu et un événement**, ou si le tweet est **invalide car il n'a aucun lieu adéquat pour l'événement**.

Si le tweet est considéré comme hors-sujet, il est supprimé de notre base de données. Les tweets hors-sujet ne parlent pas vraiment de catastrophes et sont en général des métaphores comme : "Elle risque d'inonder Montpellier à force de pleurer".

## 5.5 Administration

Un système d'administration a été mis en place comportant quatre niveaux :

- **Niveau 0** : L'utilisateur lambda.
- **Niveau 1** : Le validateur.
- **Niveau 2** : L'admin.
- **Niveau 3** : Le super-admin.

Ce système a pour but de réduire la confiance que l'on attribue aux utilisateurs inconnus afin de garder une base de données cohérente et valide.

Les Niveaux 2 et 3 peuvent voir l'ensemble des individus des niveaux inférieurs, et modifier leurs rôles. Par conséquent les administrateurs peuvent ajouter ou retirer le droit de validation, et les super-administrateurs peuvent en plus du droit de validation, ajouter ou retirer le droit d'administration.

L'accès à l'administration permet également de gérer les tweets signalés (figure 5.9). Deux alternatives sont possibles :

- Le tweet est valide par conséquent on supprime le signalement.
- Le tweet est invalide et on invalide le tweet.

Mot-Clés      Carte      Validation      Extraction      **Administration**

**Utilisateurs**

| Email         | Alias         | Admin                               | Validator                           |
|---------------|---------------|-------------------------------------|-------------------------------------|
| Flareden      | Flareden      | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Dorian        | Dorian        | <input type="checkbox"/>            | <input checked="" type="checkbox"/> |
| ReallFlareden | ReallFlareden | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| utilisateur   | Utilisateur   | <input type="checkbox"/>            | <input checked="" type="checkbox"/> |
| validator     | Validator     | <input type="checkbox"/>            | <input checked="" type="checkbox"/> |
| admin         | Admin         | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Doublepups    | Doublepups    | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |

Sauvegarder      Annuler

FIGURE 5.8 – Administration Super-admin

Mot-Clés      Carte      Validation      Extraction      **Administration**

**Signalement**

*"There is no sandstorm near paris"*

| Lieux        | Evenements                        |
|--------------|-----------------------------------|
| Lieu : paris | Long : 2.3488      Lat : 48.85341 |
|              | sandstorm                         |
|              | Signalements                      |
|              | 1 signalement(s)                  |

Supprimer Signalement      Invalider

FIGURE 5.9 – Signalement Super-admin

# Chapitre 6

## Test de montée en charge

### 6.1 Affichage sur Leaflet

Outre les simplifications algorithmiques python mineures, nous nous sommes concentrés sur la complexité de notre carte Leaflet. La carte étant un élément avec une consommation de ressource exponentielle en fonction du nombre de marqueurs, il était impératif pour nous de pouvoir simplifier son affichage et d'éviter de charger des données superflues.

Pour ce faire, nous avons utilisé la fonction heatmap offerte par le framework Leaflet. Celle-ci permet au lieu de calculer et placer chaque marqueur sur une carte, de faire une moyenne entre chaque point et d'appliquer un dégradé de couleur en fonction de la concentration des marqueurs dans une zone.

Les tests de montée en charge ont été effectués sur une machine possédant un i7 (CPU), 16 Go (RAM), Nvidia 1060 GTX (GPU). Plusieurs "activités" ont été analysées :

- **Scripting** : Temps nécessaire pour analyser et évaluer le Javascript
- **Rendering** : Temps consacré au calcul des styles et des positions des éléments sur la page.
- **Painting** : Temps d'affichage dans la fenêtre du navigateur.
- **System** : Temps cumulé de toutes les activités ne concernant pas le GPU.

Les résultats obtenus sont les suivants pour une carte sans la fonction heatmap (figure 6.1) et avec la fonction heatmap (figure 6.2).

| Nb Marqueur | Scripting (ms) | Rendering (ms) | Painting (ms) | System (ms) |
|-------------|----------------|----------------|---------------|-------------|
| 100         | 104            | 68             | 14            | 67          |
| 500         | 241            | 749            | 123           | 561         |
| 1000        | 281            | 1875           | 325           | 938         |
| 5000        | 2983           | 3289           | 6998          | 1689        |

FIGURE 6.1 – Performances sans Heatmap

| Nb Marqueur | Scripting (ms) | Rendering (ms) | Painting (ms) | System (ms) |
|-------------|----------------|----------------|---------------|-------------|
| 100         | 54             | 39             | 5             | 19          |
| 500         | 110            | 23             | 3             | 36          |
| 1000        | 314            | 155            | 13            | 78          |
| 5000        | 778            | 58             | 10            | 71          |

FIGURE 6.2 – Performances avec Heatmap

Nous constatons à travers ces expériences, la consommation exponentielle des ressources GPU de la carte en fonction du nombre de marqueur. Mais également une utilisation colossale des ressources CPU de la part du Javascript. Cette augmentation s'explique par le fait que chaque marqueur est

dynamique et possède une fonction onclick(), implémentée dans la librairie heatmap.js. L'utilisation de la fonction heatmap permet donc de réduire considérablement la complexité et les ressources utilisées. En prenant l'exemple le plus flagrant, soit 5000 marqueurs :

- Sans heatmap, le site met environ **15 secondes à réagir**.
- Avec heatmap, le site met approximativement **1 seconde à réagir**.



FIGURE 6.3 – 5000 marqueurs sans heatmap



FIGURE 6.4 – 5000 marqueurs avec heatmap

# Chapitre 7

## Bilan et difficultés rencontrées

### 7.1 Avancement du projet

À l'heure actuelle, l'application est fonctionnelle, et suit au mieux le cahier des charges qui nous a été donné. Elle permet l'extraction des tweets avec l'API de Twitter, leur évaluation par un utilisateur ayant un rôle spécifique, et leur visualisation sur une carte.

Tous les bugs majeurs ont été corrigés. Néanmoins il nous reste un bug mineur connu consistant en une déconnexion de l'utilisateur lors de la modification de son rôle par un administrateur. Cette déconnexion intervient également lors du changement de la clé Twitter.

Toutefois, le nombre d'heuristiques que nous avons développés est insuffisant et ne permet pas d'automatiser la vérification des tweets.

### 7.2 Changements majeurs

L'un des changements majeurs s'est effectué lors de la décision de mise en ligne de MongoDB. Pour effectuer cette opération nous avons dû sécuriser MongoDB avec le protocole TLS pour éviter un accès public à nos données.

Nous avons également décidé de transformer le script Python en serveur d'application web. Cette transformation a été possible avec le module "http.server" de python, ce qui permet d'exécuter le script par simple requête POST et de réaliser plusieurs appels en simultané.

### 7.3 Difficultés rencontrées

Notre première difficulté, a été la liaison entre Symfony et MongoDB en tant que base de données principale pour la gestion d'utilisateurs. La documentation étant dépréciée et certains modules n'étant plus à jour, nous avons consulté plusieurs forums afin de déterminer une méthode fonctionnelle pour Symfony 5.

Dans la seconde, il a été question de l'accès des modules Python depuis le PHP. Le serveur PHP n'arrivait pas à accéder aux modules Python installés à l'emplacement par défaut. Nous avons donc été obligés d'installer ces mêmes modules directement via PHP.

Enfin, nous avons dû résoudre le problème de communication entre Symfony et Python. Nous avons essayé d'exécuter le script directement via PHP et nous avons rencontré des problèmes de droit d'accès et l'impossibilité de recevoir une réponse à nos requêtes. Pour résoudre ce problème, nous avons transformé le script Python en serveur d'application web.

# Conclusion

Pour conclure, le projet a permis l'enrichissement de notre expérience dans des domaines techniques ainsi que dans le travail d'équipe. En effet nous avons eu l'occasion de pratiquer plusieurs langages et d'acquérir de nouvelles compétences telles que :

- La gestion de l'API Twitter via Python.
- La gestion d'une base de données NoSQL avec MongoDB.
- L'analyse morphosyntaxique avec Spacy (Python).
- La mise en place d'un serveur Ubuntu en ligne.
- L'affichage des données sur une carte avec Leaflet.

Pendant notre période de développement, nous avons découvert et utilisé un nouvel IDE. Netbeans 12, qui nous a apporté la possibilité de travailler simultanément sur le PHP distant et de pouvoir visualiser notre travail en temps réel.

Pour travailler en équipe, nous avons utilisé les méthodes agiles, avec la mise en place de réunions hebdomadaires par l'intermédiaire de Discord. De la même façon, GitHub nous a aidé de maintenir une trace de nos recherches et des mises à jour.

Plusieurs améliorations restent possibles sur ce projet afin de le rendre plus complet et attrayant. Par exemple en mettant en place plusieurs heuristiques, il sera possible de se pencher sur l'apprentissage automatique. Il est également possible de rendre le site compatible avec les plateformes mobiles afin qu'il soit plus accessible au public.

## Annexes

# Glossaire

**Bearer Token** : désigne un schéma d'authentification **HTTP** qui implique des jetons de sécurité appelés bearer tokens (aussi nommés "token authentication"). 7, 8

**cache** : désigne une mémoire qui enregistre temporairement des copies de données provenant d'une source, afin de diminuer le temps d'un accès ultérieur à ces données. 6, 7

**clef Twitter** : désigne un code unique transmis à une API afin d'identifier l'application ou l'utilisateur appelant. Les clefs d'API permettent de suivre et contrôler la façon dont l'API est utilisée, par exemple pour empêcher toute utilisation malveillante ou abusive de l'API. 18, 20

**end-point** : désigne un point d'entrée (**URL**) permettant l'accès à un service par une application cliente. 6, 8

**mot-clef** : désigne un ensemble de mot définissant une catastrophe naturelle (par exemple #sand-storm). 9, 11, 16, 18, 20

**POST** : désigne une méthode d'envoi des données en HTTP . 25

**stream** : désigne dans une requête, une séquence d'éléments prenant en charge des opérations d'agrégation séquentielles et parallèles. 7

**tweet valide** : désigne un tweet associé à une catastrophe naturelle ayant une géolocalisation valide. 18

# Acronyms

**API** Application Programming Interface. 3–6, 9, 16, 17, 20, 25, 26, 28

**CPU** Central Processing Unit. 23

**GPE** GeoPolitical Entity. 3, 16

**GPS** Global Positioning System. 4, 14

**GPU** Graphics Processing Unit. 23

**HTTP** Hypertext Transfer Protocol. 28

**IDE** Integrated Development Environment. 26

**JSON** JavaScript Object Notation. 9–11, 13, 16

**NoSQL** Not Only SQL. 5, 13, 26

**RAM** Random Access Memory. 23

**SQL** Structured Query Language. 13, 29

**TLS** Transport Layer Security. 13, 25

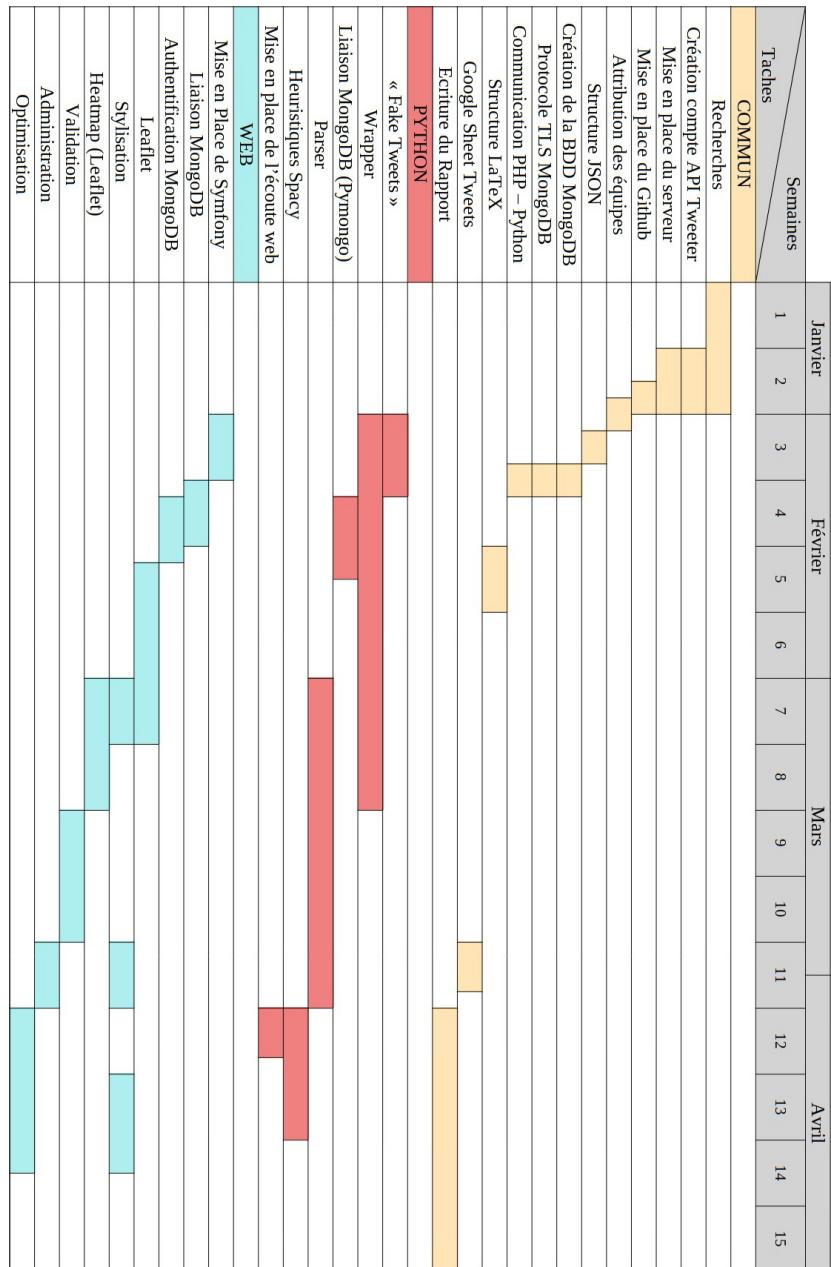
**TSV** Tabulation Separated Values. 10

**URL** Uniform Resource Locator. 28

**VoIP** Voice Over Internet Protocol. 5

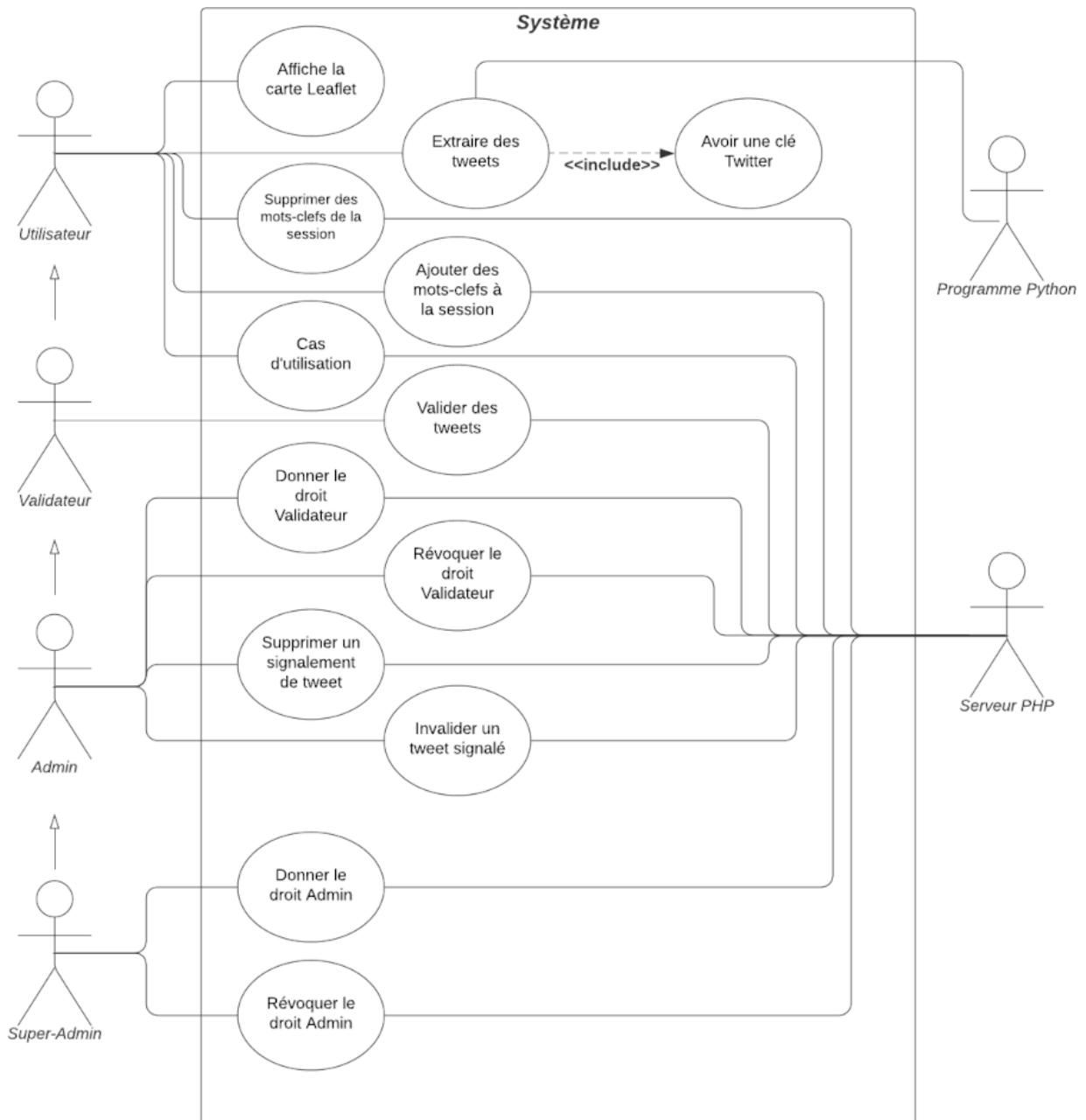
## Annexe H

### Diagramme de Gantt



## Annexe I

### Diagramme UseCase



# Bibliographie

- [1] MongoDB, site officiel, 2021  
<https://www.mongodb.com/fr>
- [2] MongoDB avec Doctrine, 2021  
<https://www.doctrine-project.org/projects/doctrine-mongodb-odm/en/2.2/index.html>
- [3] Symfony, site officiel, 2021  
<https://symfony.com>
- [4] Twig avec Symfony, 2021  
<https://twig.symfony.com>
- [5] Twitter API, 2021  
<https://developer.Twitter.com/en/docs/Twitter-api>
- [6] Spacy, site officiel, 2021  
<https://spacy.io/>
- [7] Leaflet JS, site officiel, 2021  
<https://leafletjs.com>
- [8] Leaflet, Heatmap, 2021  
<https://www.patrick-wied.at/static/heatmapjs/example-heatmap-leaflet.html>
- [9] Jquery UI, site officiel, 2021  
<https://jqueryui.com/>
- [10] Pymongo, documentation, 2021  
<https://pymongo.readthedocs.io/en/stable/>