|  |
| --- |
| Master ISD |
| Rapport du projet Dashboard Tweeter |
| Langage dynamique |

|  |
| --- |
| Boudeville Antoine – Difallah Zakaria |

Table des matières

[Présentation du projet 2](#_Toc41140763)

[Interface utilisateur 2](#_Toc41140764)

[Technologies utilisées 4](#_Toc41140765)

[Client 4](#_Toc41140766)

[Serveur. 4](#_Toc41140767)

[Outils développement et de déploiement 4](#_Toc41140768)

[Architecture Globales 5](#_Toc41140769)

[Les acteurs 5](#_Toc41140770)

[Interaction entre les acteurs 5](#_Toc41140771)

[Cache 5](#_Toc41140772)

[Architecture micro 6](#_Toc41140773)

[Routeur 6](#_Toc41140774)

[Cache 6](#_Toc41140775)

[Orchestrateur coté serveur 6](#_Toc41140776)

[Orchestrateur coté client 7](#_Toc41140777)

[Problèmes 7](#_Toc41140778)

# Présentation du projet

L’objectif de ce projet est de réaliser une application web permettant de visualiser un ensemble de données provenant du réseau social Tweeter. Cette visualisation doit se faire à travers d’un tableau de bord contenant des informations précises définies. Les données sont importées par un ensemble de fichiers.

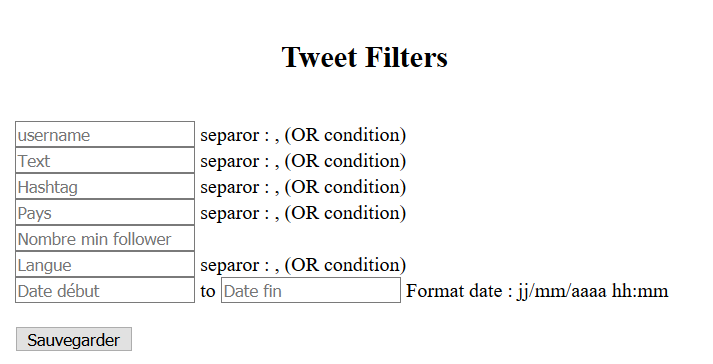
Exemple de cas d’utilisation :

* Visualiser les mots ou hashtag, *mot-clé précédé du signe #, permettant de retrouver tous les messages d'un microblog qui le contiennent,* en fonction d’un pays ou d’une langue
* Visualiser les tweets, *messages postés sur tweeter*, pour un ou plusieurs utilisateurs pendant une période
* Visualiser la langue des messages en fonction de leur position géographique

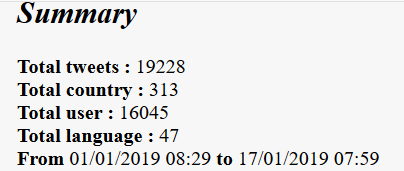
# Interface utilisateur

L’interface de l’application web se compose de plusieurs sous-élément :

Un formulaire permettant à l’utilisateur de filtrer les tweets selon différents critères.



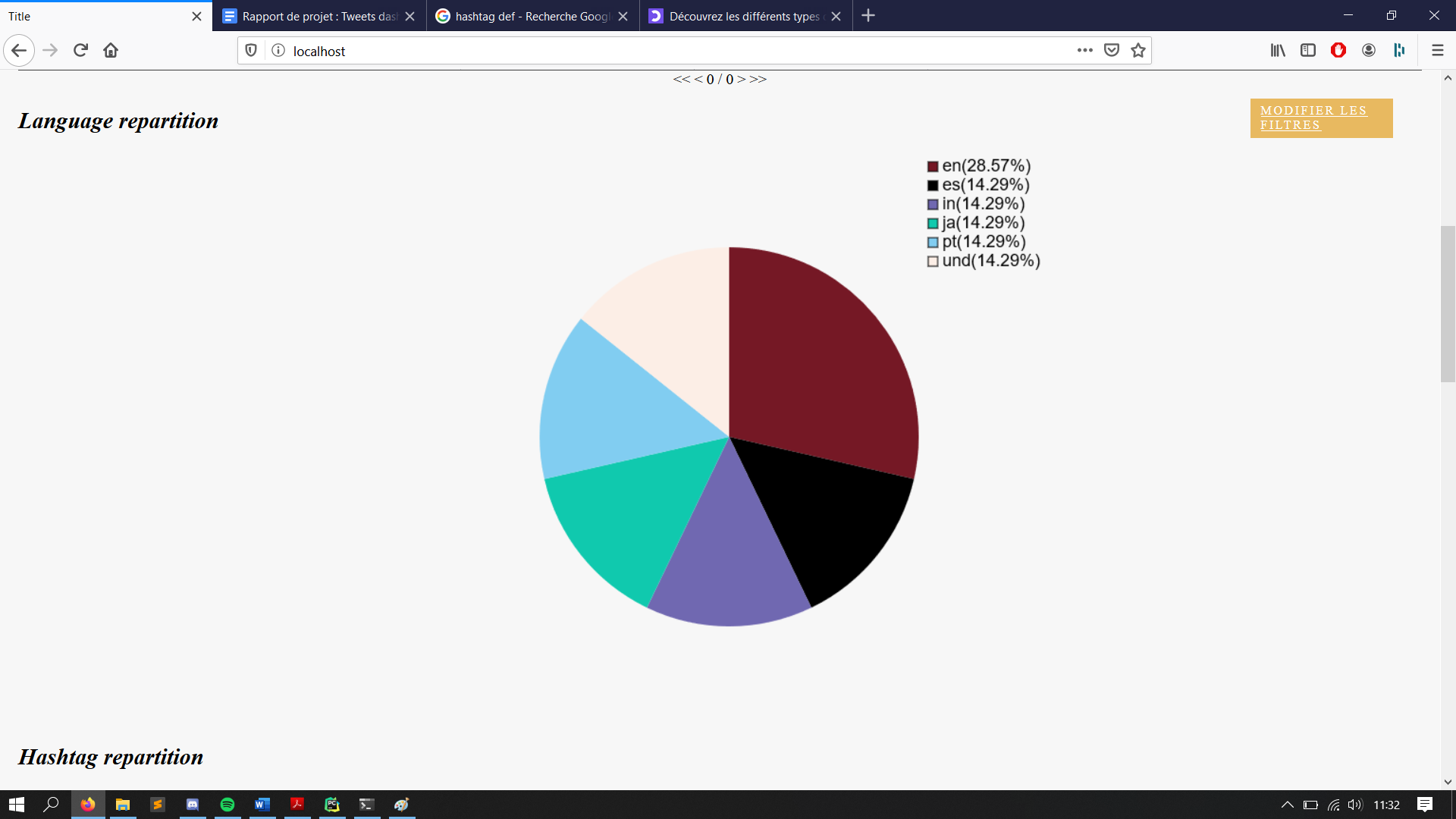
Un résumé global sur la totalité des tweets obtenue après le filtrage, comme le nombre de tweet par pays ou par langue, la période de temps…



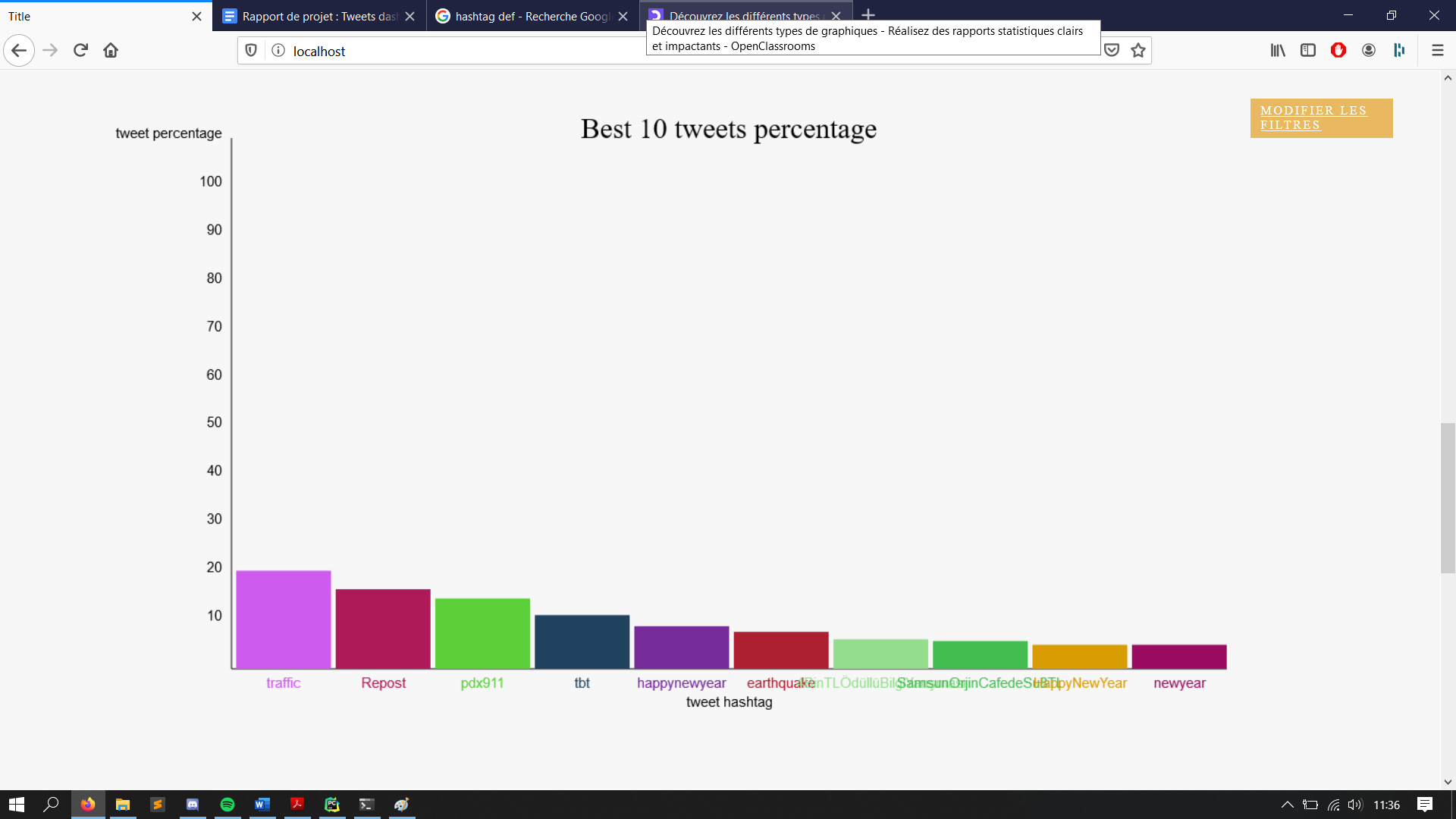
La liste des tweets obtenue après filtrage, avec un système de navigation.



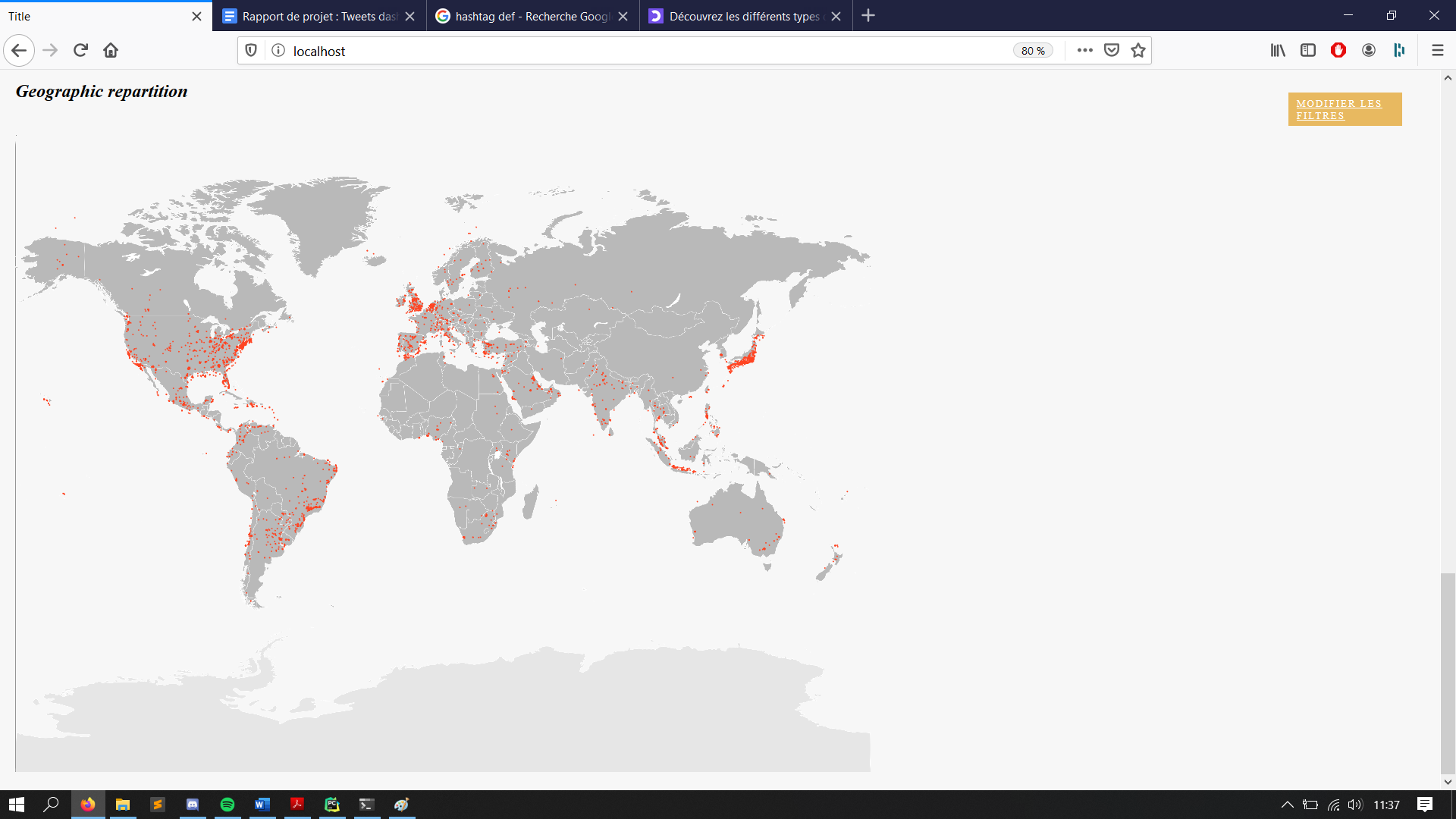
Un diagramme circulaire présentant la répartition des langues .



Un histogramme présentant les hashtags les plus populaires dans les tweets filtré .



Une carte du monde représentant la répartition géographiques des tweets.



# Technologies utilisées

Cette application est développée selon le modèle client-serveur. L’application est hébergée sur un serveur distant. Pour accéder à l’application, l’utilisateur doit rentrer l’url dans un navigateur internet (Firefox, Chrome, Explorer, …).

## Client

Les langages utilisés sont HTML, CSS, et Javascript avec l’architecture AJAX.

## Serveur.

Les langages utilisés sont python avec les librairies SimpleHTTPServer, Pandas, Thread, Yaml. L’envoi d’information vers le client utilise le format JSON.

## Outils développement et de déploiement

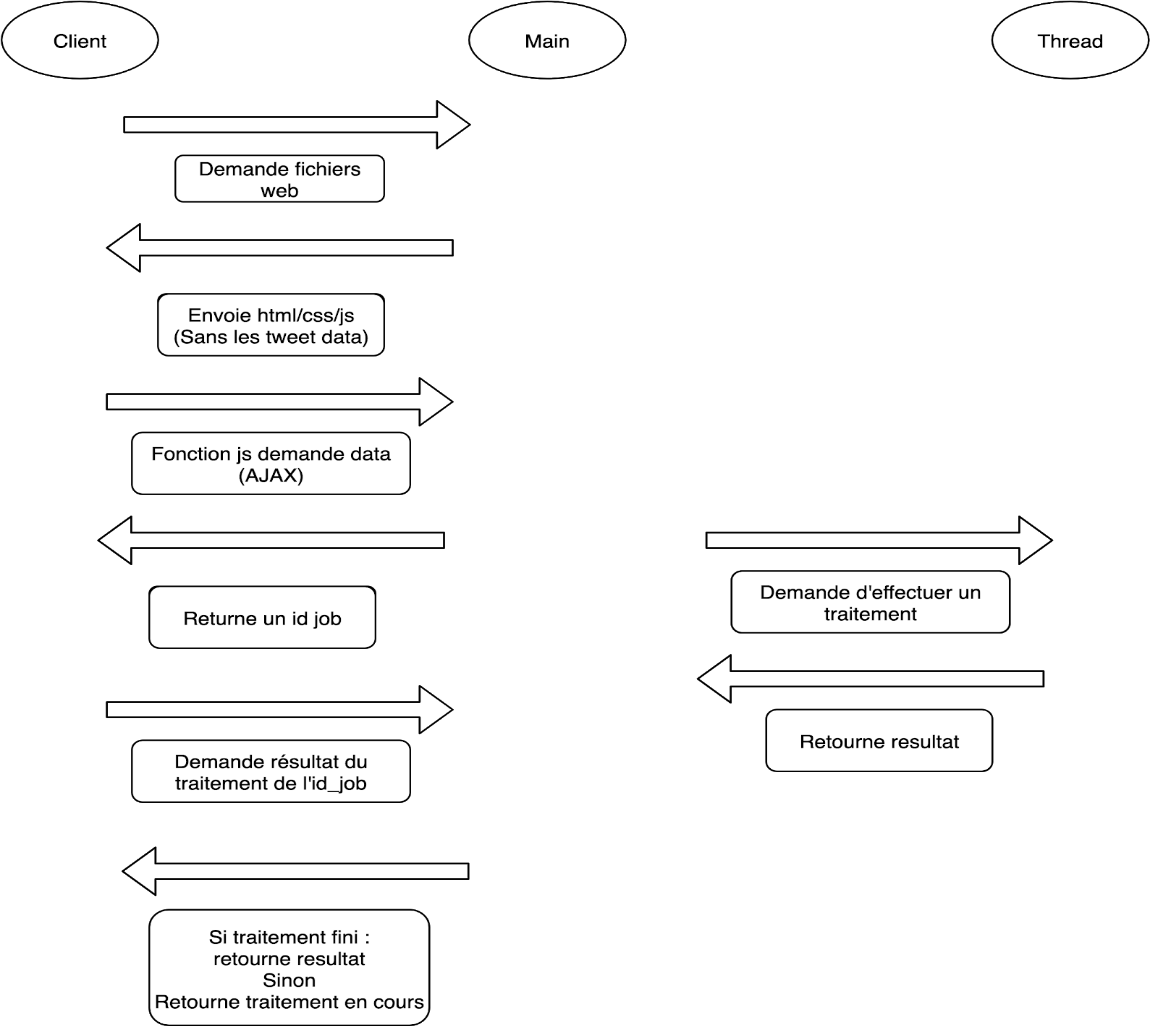
Pour développer l’application, les outils utilisés sont l’IDE Pycharm, pour gérer les sources git avec github, pour les tests pytest et pour déployer des environnements Docker.

# Architecture Globales

## Les acteurs

L’architecture global comprends 3 acteurs : le client qui demande des pages web, un serveur nommé « Main » qui traite les demandes sur client et qui gère la distribution des traitements, un serveur nommé « Thread » à qui on attribue un traitement. Le serveur « Main » et « Thread » peuvent être le même serveur physique ou il est possible de les séparer sur plusieurs serveurs (non implémenté actuellement).

## Interaction entre les acteurs

[](https://www.draw.io/?page-id=b1Qec6Jc3EDklgoQuZdC&scale=auto#G1R9KH6NnRUITyGSbs7yu81Vc_a7dMV2qB)

## Cache

Lors de l’utilisation de l’application, certains traitements sont réalisés plusieurs fois de façon identique, ainsi pour améliorer les performances, un cache a été mis en place. Ce cache sauvegarde les données des tweets une fois filtrés. Le facteur d’accélération du temps de traitement est d’environ de 2.

# Architecture micro

Différents classe et fonctions ont été développées pour ce projet, cette section présente les parties les plus intéressantes. Il est conseillé d’avoir des connaissances dans les langages python et javascript pour comprendre.

## Routeur

L’objectif de cet outil est de permettre d’associer une route à une fonction python. Plus concrètement, lorsque l’utilisateur tape *localhost/count\_tweet*, le path est */count\_tweet* qui associé à notre fonction python count\_tweet(). Ainsi cette fonction retourne un fichier JSON qui sera renvoyé au client en réponse.

Afin d’automatiser ce routeur pour être facilement utilisé dans le reste du code, un décorateur a été créé. Quand le développeur souhaite coder une nouvelle fonction il lui suffit de mettre le décorateur « @set\_route (*path*) ».

Le routeur retourne une page d’erreur spécifique si le path demandé n’existe pas.

## Cache

L’objectif de cet outil est de permettre d’éviter de faire plusieurs fois un même traitement, en particulier le traitement de filtrage des tweets. Ce cache contient une taille définie dans les fichiers de configuration de l’application. Quand le cache est rempli, les données les plus anciennes sont supprimées. Afin de se rappeler l’ordre, le cache contient une liste de clé, avec un curseur modulo la taille qui parcourt cette liste et un dictionnaire qui contient en valeur la donnée à sauvegarder.

## Orchestrateur coté serveur

L’objectif de cet outil est de mettre en place un système automatique d’exécution d’un traitement dans un thread. Plus concrètement, lorsque le serveur reçoit une demande de traitement du client, le serveur crée un identifiant pour ce traitement, exécute ce traitement dans un thread, puis retourne immédiatement cet identifiant au client. Quand le thread termine le traitement demandé, il sauvegarde le résultat dans une variable du serveur main. Pour obtenir le résultat demandé, le client doit effectuer une demande au serveur en précisant l’identifiant. Ce système a été mis en place car dans certain cas le traitement peut durer plusieurs secondes, cela permet aussi d’éviter les timeout.

Pour faciliter l’utilisation de cet outil pour le développement, un décorateur a été créé. Quand le développeur souhaite coder une nouvelle fonction qui doit utiliser cet outil, il lui suffit de mettre le décorateur « @set\_orchestrator ()» sur sa fonction, tout le processus est complètement automatisé.

## Orchestrateur coté client

L’objectif de cet outil est de permettre d’automatiser l’orchestrateur coté client. En effet, comme vu précédemment, pour certains traitements, le serveur ne retourne pas un résultat mais un identifiant lié à ce résultat. Ainsi il est nécessaire de réaliser des requêtes supplémentaires pour récupérer ces résultats. Pour cela une classe Orchestrateur a été implémenté en javascript. Lorsque l’utilisateur modifie un filtre, le javascript ne lance pas une requête ajax pour récupérer le résultat mais passe par l’orchestrateur, avec une demande de traitement. L’orchestrateur envoi une requête au serveur puis récupère tout l’identifiant. Puis toutes les secondes il envoie une requête au serveur pour lui demander le résultat. Quand le résultat est disponible, il va exécuter la fonction de callback qui permet d’afficher le résultat dans la page HTML.

# Problèmes lors du développement

Lors du développement de l’application quelques petits problèmes très légers ont été rencontrés.

Un des premiers problème fut l’absence de tests de non régression, ce qui a évidemment conduit lors du développement de nouveau code, d’erreurs dans du code existant.

Une seconde erreur fut l’oublie de verrou sur les Thread qui à produit des erreurs très étranges.

Enfin un dernier problème fut la déformation de l’image de la carte du monde lors de son ajout dans la page HTML. En pensant que l’erreur venait de la fonction contenant l’algorithme Mercator.

# Organisation

Le partage des fonction à développer a été respecté comme demandé dans le sujet. Une personne a réalisé l’orchestrateur en javascript, les graphiques et les traitements sur le dataframe en python, l’autre personne a développé l’orchestrateur et le routeur en python et le système de filtres en javascript.