Projet Language Dynamique
Twister
Responsable du projet :
Kim Nguyen

Elève: Brami Nathan

Rendu le :Mardi 4 décembre 2018

Sommaire :

I°) Prise en Main du projet

1°) Installation du projet

Contrairement à ce qu'annoncé dans un précèdent mail j'ai préféré ne rien gardé de l'ancien code et tout refaire.
Il était plus facile pour moi de tout refaire depuis le début que de comprendre un code non fini.
Pour l'installation lire readMe.txt

2°) Installation du projet

Pour le lancement des tests exécuter test.sh.
Il ne faudra rien toucher car c'est fais avec selenium.
Premier test avec trois clients différents.
Deuxieme test il faudra une connexion internet et mettre en plein écran.

3°) Arborescence du projet

```
├── Client
|   //Le dossier client possede toute la partie visuel du client, CAD html,css,js
|
|   ├── CSS
|   |   └── style.css
|   ├── Html
|   |   └── index.html
|   ├── Images
|   |   └── world_map.png
|   └── js
|       ├── liste_country.txt
|       ├── requetes.js
|       ├//Le script requetes possède le code qui permet d'emmètre les requetes au serveur
|       ├── pricipale.js
|       ├//Le script pricipale.js possède le code qui appelle les autres scripts.js lorsqu'un
|       | évenement se produit. Il permet aussi de faire des listes déroulantes et de décider
|         aussi ce que l'on affiche.
|       ├── map.js //crée la carte
|       └──graph.js//crée le graph
└── Serveur_BA
  | //Le dossier Serveur_BA possede toute la partie caché du site :
  |  CAD le serveur, le csv de tweet et les requetes python qui interrogent le
  |  serveur
  |
  ├── requete_and_BA
  | //Le dossier requete_and_BA contient le csv de tweets et les code des requetes
  |   ├── dist_levenstein.py
  |   ├
  |   ├── __pycache__
  |   |   ├── dist_levenstein.cpython-36.pyc
  |   |   └── requetes.cpython-36.pyc
  |   ├── requetes.ipynb //contient les requetes sous forme d'un notebook,et genère le  requete.py
  |   ├── requetes.py
  |   ├//Le fichier requetes.py contient le code des requetes
  |   └── tweets.csv
  └── Serveur
      ├── MyFiles.py
      ├── MyLinks.py
      ├── __pycache__
      |   ├── dispatcher.cpython-36.pyc
      |   ├── MyFiles.cpython-36.pyc
      |   └── MyLinks.cpython-36.pyc
      ├
      ├── server.py
      ├//Le fichier server.py contient le code du serveur
      └── tweets.csv
```

II°) Client

1°) Aspect général

i°) Ce que mon site fait

Mon site offre la possibilité d'afficher sur une carte le nombre de tweet par pays.
Il peut aussi offrir l'affichage d'un graphe montrant le nombre de tweet par pays,
Il aurait put aussi offrir l'affichage montrant le top 10 des hashtags les plus utilisés mais la partie client n'as pas été finalisé.

Nous pouvons utiliser les filtres a disposition pour :

1. Filtrer selon un mot ou plusieurs mots (séparé par un espace) contenue dans un tweet
    a. Si le mot n'est contenue dans aucun tweet le site met a disposition une liste de mot disponible dans les tweets qui se rapproche le plus du mot demandé basé sur la distance de levenstein
2. Filtrer les tweet selon un ou plusieurs pays
3. Filtrer les tweets selon les dates

Tous ces filtres sont facultatifs nous pouvons appuyer sur le bouton search avec tous les filtres vides.
Si un filtre renvoie aucun résultat il est annulé.

ii°) Présentation du fichier Json

Afin d'afficher tous ce que j'ai citer ci dessus j'ai besoin de retourner les tweets sous forme d'un fichier Json que nous appellerons pour le reste du projet : **Resumer de tweet**
Voici la description du fichier Json retourné.

Fichier Json :

```
{
  "recommended word" : //*ici le dictionnaire contenant la liste de mots corrigés des mots ayant eu des erreurs
  "recommended Country" : //*ici le dictionnaire contenant la liste de mots corrigés des country ayant eu des erreurs
  "tweet total" : 300, //*ici le nombre de tweet total
  "words_count" : 3, //*ici le nombre de mots qu'a recherché le client
  "tweets_per_country" : //*ici on retourne les X top pays ayant le plus grand nombre de tweets incluant les mot recherchés
  [ //*Pour chaque pays nous avons :
    {//*Pour la France
      "country_name" : "France", //*nom du pays
      "total_tweets" : 13, //*nombre de tweets total dans lesquels apparait au moins 1 mot recherché
      "total_tweets_per_word" : //*nombre de tweets total par mot recherché
      [
        {"word" : "mot1", "count" : 4}, //*1er mot recherché + nombre de tweets contenant ce mot
        {"word" : "mot2", "count" : 3}, //...etc
        {"word" : "mot3", "count" : 6}
      ]
    },
    {//*Pour icelland
      "country_name" : "Iceland", //...etc
      "total_tweets" : 57,
      "total_tweets_per_word" :
      [
        {"word" : "mot1", "count" : 45},
        {"word" : "mot2", "count" : 9},
        {"word" : "mot3", "count" : 3}
      ]
    }
  ],
  "tweets_latitude_longitude" : //ici on retourne les latitudes et longitudes de tous les tweets contenants les mots recherchés
  [
    //les tweets du 1er mot recherché
    {
      "word" : "mot1", //le mot recherché
      //liste d'objets, chaque objet contient la longitude et latitude d'un tweet qui contient le mot 1
      //donc 1 objet par tweet
      "coordinations" : [{"lat" : 12.3, "lon" : 45.3}, {"lat" : 56, "lon": 76.9}]
    },
    //...etc
    {
      "word" : "mot2",
      "coordinations" : [{"lat" : 1.3, "lon" : 4.3}]
    },
  ],
  " latitude_longitude_per_country" : //ici on retourne les moyennes des latitudes et longitudes de tous les tweets par pays.
  "top_hashtags" : //ici on retourne les top 10 hashtags qui apparaissent dans l'ensemble des tweets qui contiennent au moins 1 des mots recherchés
  [
    //1 objet par hashtags dans cette liste
    {"hashtag" : "#coucou", "count" : 1239}, //chaque objet = le nom du hashtag + le nombre d'occurences dans les tweets qui contiennent au moins 1 des mots recherchés
    {"hashtag" : "#hello", "count" : 1204},
    {"hashtag" : "#cv", "count" : 1200},
    {"hashtag" : "#life", "count" : 1054},
...
  ]
}
```

2°)JavaScript
i°) Graphe et map

La map est une image du monde. Les points sont placés sur la carte avec une position left et top calculé en fonction de la taille de l'image et de la moyenne de la latitude et de la longitude.
La taille de la bulle correspond a un pourcentage du nombre de tweets.

Le graphe a été constuit pour être le plus dynamique possible.
Actuellement elle affiche le nombre de tweet par pays classé par ordre décroissant.
Mais elle a été codée afin de choisir le nombre de colonnes que l'on veut afficher (le pourcentage se règlera sur le nombre de colonne que l'on voulons afficher), ou si l'on veut afficher les top_hashtag.
La partie cliente n'eut fut pas implémenté.

ii°) Requête ajax asynchrone

Afin de récupérer les tweets nécessaire à l'affichage des graphes j'effectue diverses requêtes

Les requêtes se font de manière asynchrone, avec une limite de temps de 8 secondes.

Afin de réaliser une requête je fais appel a Mediaquerry qui vas mettre en forme l'url avec les bon paramètres afin de respecter la mise en forme suivante:
/mon_Url_?parametre1= "valeur1"?parametre2= "valeur2"

Afin d'interroger le serveur j'utilise des requêtes ajax qui ont pour particularité d'être asynchrone.
L'asynchrone est géré à l'aide des promesses qui ont une limite de temps de 8 secondes. Cette limite s'effectue grâce a une promesse qui se manifeste au bout de 8 secondes en parallèle de la promesse qui réalise une requete ajax et promise.race choisit la première promesse qui manifeste un resultat.

III°)Serveur et Base de données
1°) Serveur htpp.serveur
i°) Propriétés et variable globales

Mon serveur est multithread et permet à des clients de faire des requêtes simultanément.
Pour cela j'ai fais en sorte a ce qu'il n'y ai aucun conflit sur la base de donnée des tweets et les autres bases de données.
Afin de respecter cela, j'ai chargé la base de donnée sous forme de dataFrame pandas dans une variable globale appelé **tweet** dans le fichier requetes.py.

Mais afin de proposer un système de correction et d'avoir un système de **mémoïsation,**

j'ai eu aussi besoin de stocker mes résultats dans le dataFrame : **dataframe_word** et **contry_for_leven**.
Le but aussi de faire cela me permet d'exécuter mon code dans du code déjà compiler a l'aide de pandas car la complexité calculant la distance de Levenshtein est d'ordre de $O(n^2)$.
La complexité est donc optimisée car toutes les opérations se font le maximum possible à travers du code compilé.

Mais la mémoire aussi est optimisée car nous n'avons que 3 dataFrames partagé à tous mes Thread en lecture (n'occasionnant aucun conflit) et faisant des copies que de petites partie du dataFrame pour faire des écritures dans des variables temporaires.

ii°)Structure et Fonctions

Afin de créer mon serveur, j'ai crée une class MyHttpRequestHandler qui hérite de la class htpp.serveur.SimpleHTTPRequestHandler.
Pour créer le serveur j'initialise un thread ThreadingHandler dans la méthode get_server.
Mon serveur a une méthode do_get qui vérifie si la ressource demandée est un fichier dans MyFile ou alors une requête dans MyLinks.

2°)Requetes pandas

Afin de retourner les tweets nécessaire a l'affichage des graphes j'effectue divers afin de filtrer les tweets demandé selon les paramètres suivant :

- Requete par Mot
- Requete par Pays
- Requete par Date

Pour les Mots et les Pays je vérifie si mon tweet contient le paramètre exact demander,insensible a la casse. Si nous prenons par exemple le mot Salat et la liste de mot suivant :

1. salaT
2. Salat
3. Salate

Les mots garder seront le 1 et le 2.

Pour les requetes par mots nous pouvons demander plusieurs mots a la fois de la forme: "mot1 mot2 " et le resultat renvoyer sera le resultat concatener du résumer de tweet de tous les mots demander

IV°) Conclusion

Pour conclure le projet a été codé de manière multithread afin de n'avoir aucun conflit et le plus optimal possible que ce soit en complexité ou en terme de mémoire.

Le code a été conçu et préparer afin qu'il puisse être le plus dynamique possible suivant la volonté du client, juste en changeant les paramètres de graph.js.

Chaque requête du client a été faite afin qu'elle soit insensible a la casse et afin que le site puissent afficher toujours quelque chose.

Les filtres sont robuste et quelque soit les combinaisons de requêtes demandées elle ne fera pas bugée le site.

Pour aller plus loin il serait intéressant de finaliser les graphes. Mais aussi d'améliorer le système de correction de mot qui est trop couteuse en s'inspirant du système de correction de google (mais je n'ai pas assez trouvé de documentation la dessus).