Rapport IFT 3150 - Projet d'informatique

Système automatisé d'analyse de Fact-Checkers

Armel Ivan Bado

Sous La supervision de :
Sabrine AMRI (PHD Candidate)
Prof. Esma Aïmeur

DIRO
Université de Montréal
Canada
Automne 2021

1 Introduction

La venue des réseaux sociaux a marqué un tournant dans la manière donc nous consommons l'information. Les nouvelles sont maintenant accessibles à un plus grand nombre et ce en un laps de temps. Ce changement de paradigme se révèle être un couteau à double tranchant. D'une part nous assistons à l'essor d'un monde plus connecté, informé mais néanmoins vulnérable. Parmis ses dangers, la désinformation puis mésinformation massive, des fléaux que cherchent à combattre les Fact-Chekers.

Dans le cadre du présent projet, nous nous sommes intéressé à développer une solution qui a pour but d'assister l'utilisateur dans sa prise de décision quant à la véracité d'un article donnée. Dans le présent document nous présenterons le problème d'agrégation des valeurs de vérité des Fact-checker ainsi qu'une façon de le résoudre. Nous montrerons par la suite une implémentation de la dite solution.

2. Notions clefs

Dans cette section, l'on définit des notions et termes importants qui reviendront tout au long de notre exposé.

Fact-checkers : Entités publiques ou privées qui analysent les informations circulant dans les médias et leur attribuent une valeur de véracité

Assertion / Claims : Il sera ici question des assertions tenues par les médias (article, photo, blog-post) à valider ou invalider.

Valeurs de véracité : Échelle propre aux différents Fact-checker qui fait office mesure du degré de vraisemblance d'une affirmation.

Métadonnées : Données descriptives contenues dans le code source d'une page web. Elles sont le plus souvent non visibles par l'utilisateur.

Désinformation vs mésinformation : Deux termes assez proches. Le premier renvoie à l'utilisation des médias pour faire passer un message susceptible de tromper ou d'influencer l'opinion publique.

Quant au suivant, il s'agit de poser l'acte en connaissance de cause. [1]

Sementic Similarity (similarité sémantique) : Mesure de la distance ou proximité entre différents termes sur la base de la proximité en sens qu'ils partagent. Elle s'oppose à la proximité lexicale qui s'appuie sur la récurrence de termes similaire entre deux expressions.

IFCN: International Fact-Checking Network, unité de l'institut Poynter ayant pour but de recenser et authentifié les Fact-Checkers à l'échelle internationale.

3. Problématique et solution proposée

3.1 Problématique

L'on dénombre une multitude de Fact-chekers. En 2021 L'IFNC en recense 102 vérifiés [2].

Chacun des organismes a une échelle de valeur de vérité qui lui est propre et ces échelles peuvent varier drastiquement les unes par rapport aux autres.

- False
- True
- Misleading
- Farcial
- Withholding Judgment

Figure 3.1-Figure présentant l'échelle de véracité de Fact-Can

- True
- Mostly true
- Mixture
- Mostly False
- False
- Unproven
- Outdated
- Miscaptioned
- Correct attribution
- Misatributted
- Scam
- Legend
- Labeled satire
- etc…

Figure 3.2-Figure présentant l'échelle de véracité de Snopes

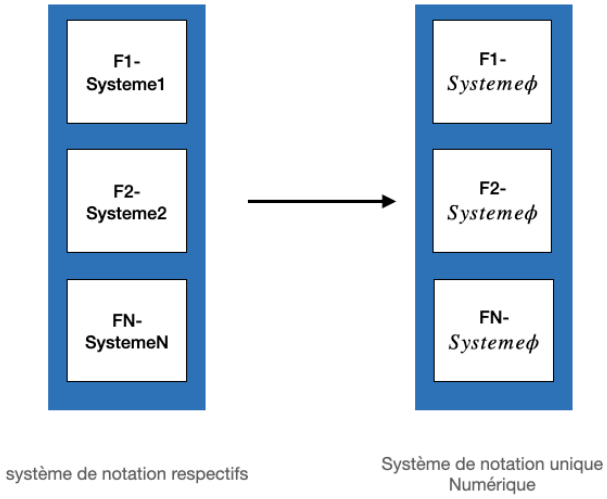
Pour un même article, il peut y avoir divergences de valeurs de véracité d'un Fact-check à l'autre. L'on cherche donc un modèle nous permettant d'agréger ces valeurs.

On veut, pour un article ou une assertion donnée, identifier les analyses faites par des Fact-checkers et convertir les valeurs de vérité dans un système unifié afin de calculer un score final grâce à une fonction d'agrégation.

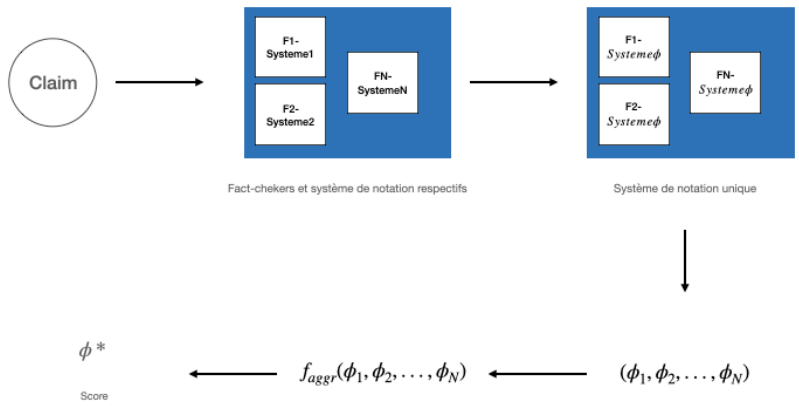
3.2 Solution

Nous proposons le modèle suivant. Dans un premier temps, nous convertissons les notations des différentes échelles de véracité vers une échelle numérique (1 à 5 dans notre cas). Dans le contexte de notre implémentation, cette étape est faite manuellement, mais on pourrait trouver une façon de l'automatiser.

On établit donc un mapping qui dépend entre autres de la taille de l'ensemble initial ainsi que de la distance relative entre les différentes valeurs de véracité.



Pour une assertion donnée, on va donc vérifier l'ensemble des Fact-checkers concernés et extraire leurs notes respectives avant d'y appliquer notre fonction d'agrégation pour obtenir le résultat final.



L'algorithme est le suivant :

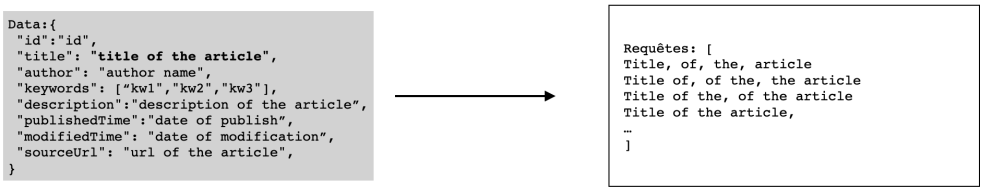
1. Considérer l' URL de l'assertion en input
2. Joindre la page vers lequel renvoie l'URL
3. Extraire les métadonnées présentes sur la page
4. Générer un ensemble de requêtes sur la base des métadonnées
5. Trouver l'ensemble des Fact-Checkers ayant traité cette assertion
6. Évaluer la pertinence des résultats trouvés
7. Convertir les scores obtenus dans le système unifié
8. Calculer et retourner le score grâce à la fonction d'agrégation.

3.2 Algorithmes Clefs

Dans cette section , nous décrivons quelques algorithmes importants qui sont utilisés tout au long de notre processus.

3.2.1 Generation de requêtes grâces aux métadonnées

Une fois les métadonnées extraites, nous voulons formuler un ensemble de requêtes vers un API pour obtenir l'ensemble des Fact-chekers associés. Dépendamment de la requête passée en input, on obtient différentes valeurs. On cherche donc à maximiser les valeurs atteintes.



L'algorithme dans sa première version considère uniquement le titre et génère un ensemble de phrases allant de 1 à n mots, ou n est la longueur du titre.

Ici on considère uniquement les chaines composées de mots successifs.

On effectue un post-traitement sur l'ensemble des requêtes générées pour éliminer les éventuels doublons.

Une éventuelle amélioration de l'algorithme pourrait attribuer une étiquette aux différents mots (article, nom, verbe, ponctuation) et éliminer les requêtes sans valeurs sémantiques réelles :

Ex pour "Title of the Article, and more":
- Article tout seul : of, the
- Succession d'articles : of the
- Ponctuation et article : , and

```
//Time complexity:O(n²)
ListOfRequest =[]
begin
function generateRequest(title)
    n <- length(title)
    for i  in (0,n)
        for j in (i+1,n+1)
         add to listOfRequest title [i:j]
         end for
    end for
end function


//amélioration doublons O(n)
removeDoubles(ListRequest):
    return List(set(ListOfRequest))
```

La fonction de génération dépend également d'autres variables, les champs de notre métadonnée. On peut élargir ou restreindre le nombre de valeurs atteintes en jouant sur ces paramètres et en y appliquant une fonction de génération de requêtes M.

$$M\begin{bmatrix} v_1 : title \\ v_2 : author \\ \cdots \\ v_N : keywords \end{bmatrix} \rightarrow$$

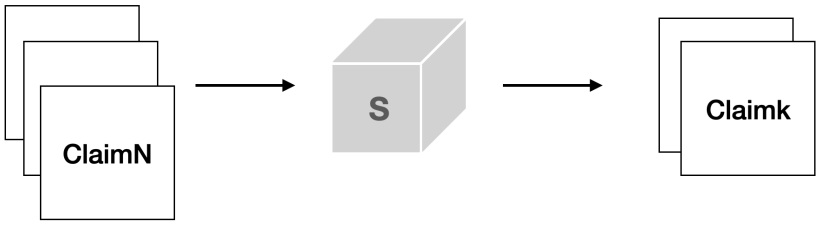
```
Requêtes: [
Title, ~~of~~, ~~the~~, article
Title of, ~~of the~~, the article
~~Title of the~~, of the article
Title of the article,
…
]
```

Cette fonction est d'autant plus importante lorsque l'on considère une métadonnée contenant uniquement des mots-clefs…

3.2.2 Algorithme de similarité sémantique

Parmi les analyses de fact-checkers reçues, on veut s'assurer de filtrer celles qui ne sont pas pertinentes.

Pour ce faire, on compare le titre de notre article à la description extraite de l'analyse. On en déduit un score de similarité sémantique (abstraite à la librairie Science-Transformer dans notre cas) et acceptons uniquement celles qui sont au-dessus du **seuil** fixé. [3]



La plupart des métadonnées contiennent également une description. La validation peut donc également se faire en deux étapes.

1. Validation(titre, description de l'analyse) > Seuil
2. Validation(description métadonnée, description de l'analyse) > Seuil

→ considérer la moyenne ou le max ou le min des scores

Étant donné la présence possible de valeurs communes pour des requêtes similaires, on garde en mémoire (Hashmap) les scores obtenus pour les différentes analyses afin de ne pas répéter certaines opérations. On peut accéder aux scores en complexité de temps constante.

3.2.3 Algorithme de calcul de score v.1

L'algorithme de score s'inspire de la technique employée dans le papier de *Zalek Malik* et *Athman Bouguettaya*.[4] Dans cet article on propose une méthode pour attribuer un score de véracité aux "reviews" des internautes dans le contexte des plateformes de e-commerce.

Nous commençons par considérer chacun des scores dans notre système unifié puis nous attribuons une valeur de crédibilité à chacune de ses valeurs.

Pour une échelle de 1 à 5 (écart est 4) nous aurons un ensemble contenant :

- n_1 scores de valeur 1
- n_2 scores de valeur 2
- …
- n_5 scores de valeur 5
-

$\rightarrow f_{agg}(n_1, n_2, n_3, n_4, n_5)$

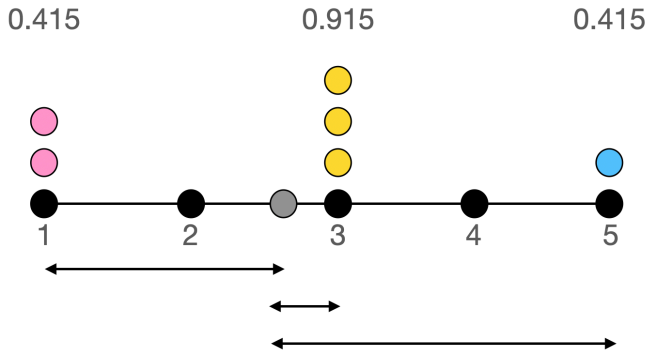
L'idée est qu'une note éloignée de la majorité des autres notes est moins susceptible d'être fiable, donc on veut réduire son influence.

On calcule le centroïde des valeurs : $c = \frac{\sum n_i * i}{\sum n_i}$

et on déduit la distance entre chaque point du centroïde: $d_i = |i - c|$

Le degré de crédibilité est inversement proportionnel à la distance:

$cred_i = 1 - \frac{d_i}{4}$



La fonction d'agrégation devient la moyenne pondérée des scores par les facteurs de crédibilité:

$$\phi * = f_{agg}(n_i, \dots) = \frac{\sum cred_i * n_i * i}{\sum n_i}$$

3.2.4 Amélioration à l'algorithme de score v. 2

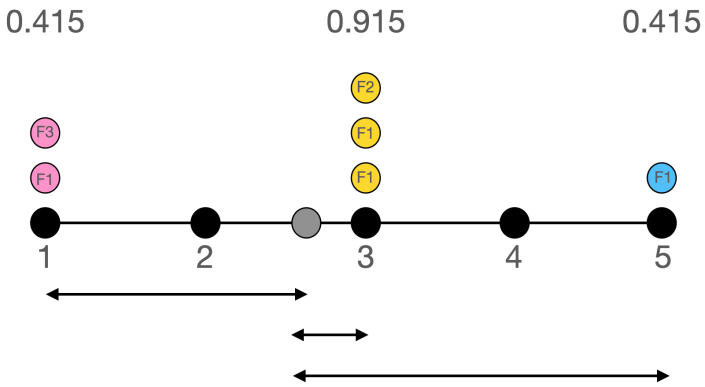
L'algorithme précédent dépend uniquement des scores, ils sont évalués distinctement . Cependant, on peut avoir des scores différents rattachés au même fact-checker pour des dates différentes.

On veut introduire un facteur de poids pour prioriser les scores les plus récents au détriment des plus anciens.

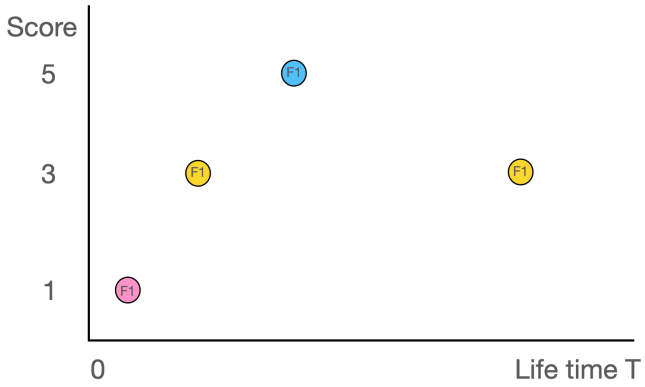
Pour les valeurs suivantes (FactCheker, score, Date) :

1. (F1, 2, 11-11-2021)
2. (F1, 5, 20-12-2021)
3…
n. (F1,5,12-01-2022)

Pour chaque nouvelle valeur donnée de score rattachée à un Fact-checker donné, on calcule $T = mostRecentDate - lessRecentDate$



En fonction de la durée de vie de la valeur à partir du score le plus récent, on attribue un poids qui est inversement proportionnel.



On aura $w_{ti} = e^{-\lambda t}$ on utilise un modèle de décroissance exponentielle ici. On pourra moduler la valeur du paramètre λ.

Le même principe de poids temporel pourrait être étendu entre les classes de Fact-checker également. w_{tj} pour chaque famille de Fact-checker $Fact_j$.

$$\phi^* = f_{agg}(ob_i, \dots) = \frac{\sum w_{tj} * w_{ti} * cred_i * n_i * i}{\sum n_i} \quad \text{où}$$

$ob_i = [Fact_i, Score_i, Date_i]$

$n_i = len(ob_i)$ ou les scores i sont les mêmes

3.2.5 Extraction de métadonnées

On considère deux façons d'extraire les métadonnées:
1. À partir d'un article sur une page web.
2. À partir d'une photo de laquelle on extrait du contenu texte.

Dans le cas de (1) on recherche les valeurs spécifiques de la page tels que:
- Le titre
- L'auteur
- Les mots clefs
- La date d'écriture
- …

On cible particulièrement le balise <meta> dans le code source ainsi que les microdonnées de type RDFa, ou JSON-LD. [5]

2. Dans le cas d'une photo, on utilise une OCR (optical character recognition tool) pour extraire le contenu texte.

Il s'agit par la suite de convertir le contenue textuel reçu en requêtes tel qu'à la section **3.2.1.**

4. Implémentation

Afin d'évaluer notre modèle, nous avons implémenté une version fonctionnelle. Dans notre implémentation nous considérons la v.1 de l'algorithme d'agrégation.

4.1 Architecture du programme

Le programme contient une interface graphique implémentée principalement en React, HTML, CSS et JS.

Le Backend est relié à une base de données locale (pour stocker les données d'authentification) et à une base de données MongoDB Atlas.

Il est également connecté à des services externes pour charger ou envoyer de la donnée.

Bien que le langage principal employé est Js et NodeJs , certaines sections sont implémentées en Python pour faciliter l'accès aux librairies de traitement de données.

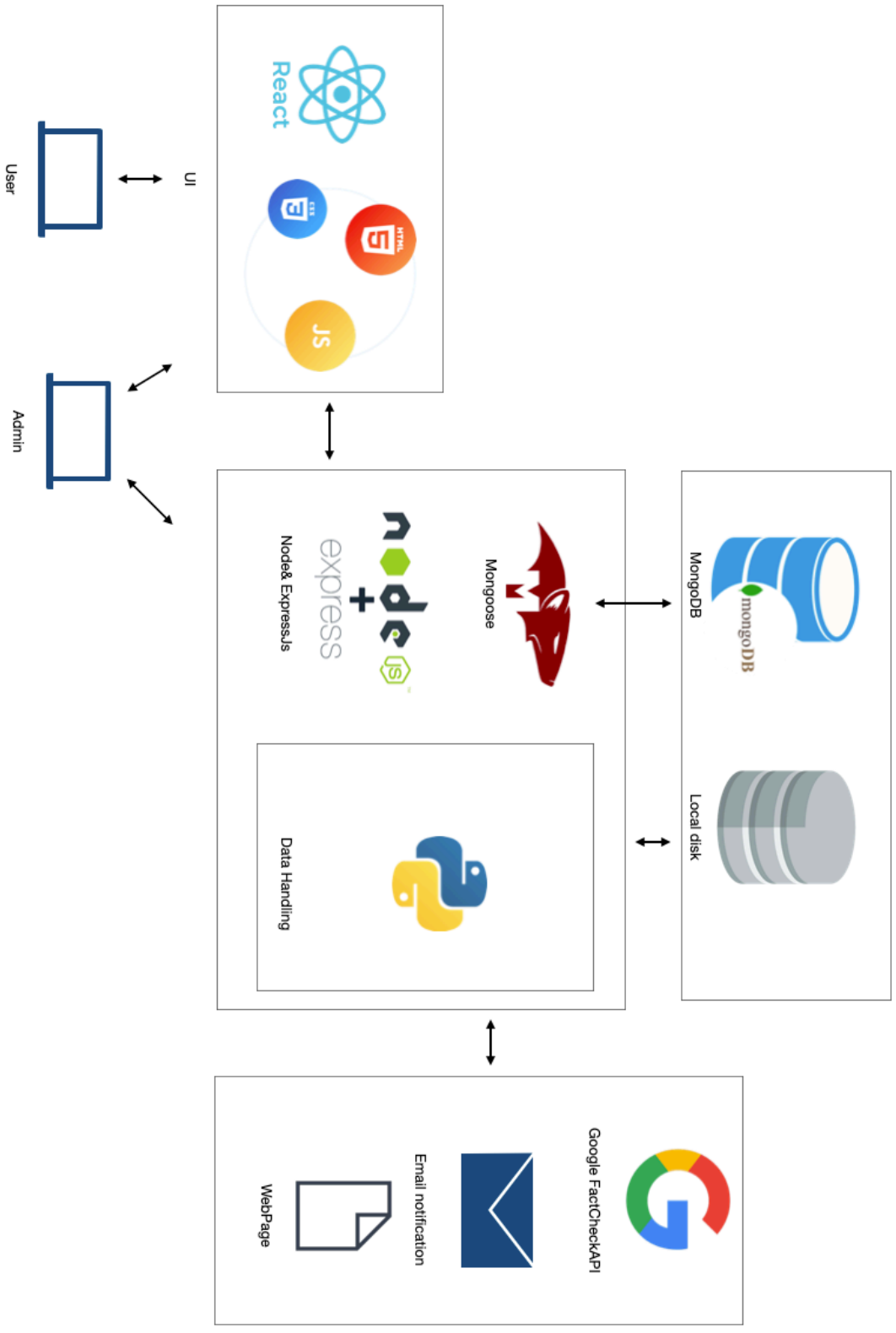


Figure 4.0 Architecture du Programme

4.1 Modules et composantes

4.1.1 Modules Python

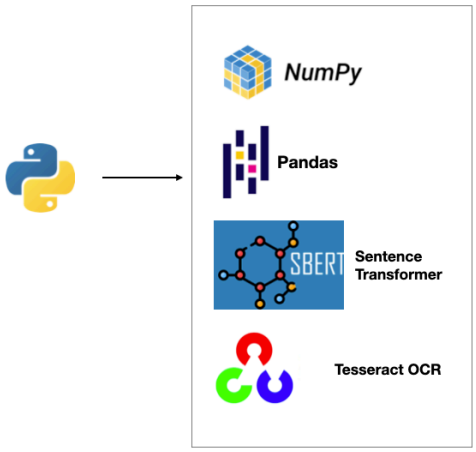


Figure 4.1.1 Modules Pythons

Numpy et Pandas : modules permettant de faciliter le traitement de jeux de données en permettant notamment les manipulations de listes. On l'utilise dans le cadre dans notre fonction d'agrégation.

Sentence Transformers : module permettant de manipuler des phrases et voir les rapprochements entre celles-ci, on l'utilise dans notre fonction de similitude avec le modèle ***sentence-transformers/stsb-roberta-large*** .

Tesseract OCR: module optique de reconnaissance de caractères. Utilisé lorsque l'URL considérée dans notre programme renvoie vers une image.

BeautifulSoup: module employé pour scrapper ou crawler des sites web. On l'utilise pour extraire les métadonnées des pages web.

4.1.2 Modules Node JS

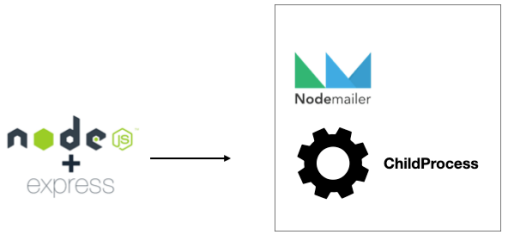


Figure 4.1.2- Modules Node

Nodemailer : module permettant d'envoyer des emails à partir de Node JS. On l'utilise dans le code pour notifier l'administrateur de l'ajout d'un nouveau Fact-checker non encore identifié, sans score, dans la base de données.

ChildProcess : module permettant de créer des sous-processus dans un programme. On l'utilise pour exécuter le bloc de codes en Python et les retourner vers Node JS.

4.1.3 Google Fact-Check API

Google Fact-Check API est un API de Google qui compile, met à jour et rend accessible des analyses d'assertions et articles effectuées par des Fact-checkers du monde entier.

L'outil nous permet d'éviter une part importante de complexité qui est celle du "scrapping" puis structuration des données dans un format unique.
Une requête vers l'API retourne un objet JSON au format suivant:

```
{
  Claims : [
     {claim1},
```

```
      …,
      {claimN},
]
}
```

Il s'agit d'une liste d'assertions où chacune est dans le format suivant:

```
{
      text :"text value",
      claimant:"claimant value ",
      claimDate:"2021-11-01T00:00:00Z",
      claimReview:[
            {
                  publisher:{
                        name:"India Today",
                        site:"indiatoday.in"
                  },
                  url:"theUrl.com",
                  title:"this is the title",
                  reviewDate:"2021-11-08T00:00:0",
                  textualRating:"Mostly false",
                  languageCode:"en"
            }
      ]
}
```

La consistance du format de retour facile le traitement des données. L'API nous permet de spécifier entre autres la langue des claims ainsi que le Fact-checker qui y est associé lors de notre requête.

4.2 Objets et base de données

La base de données est structurée de la façon suivante:

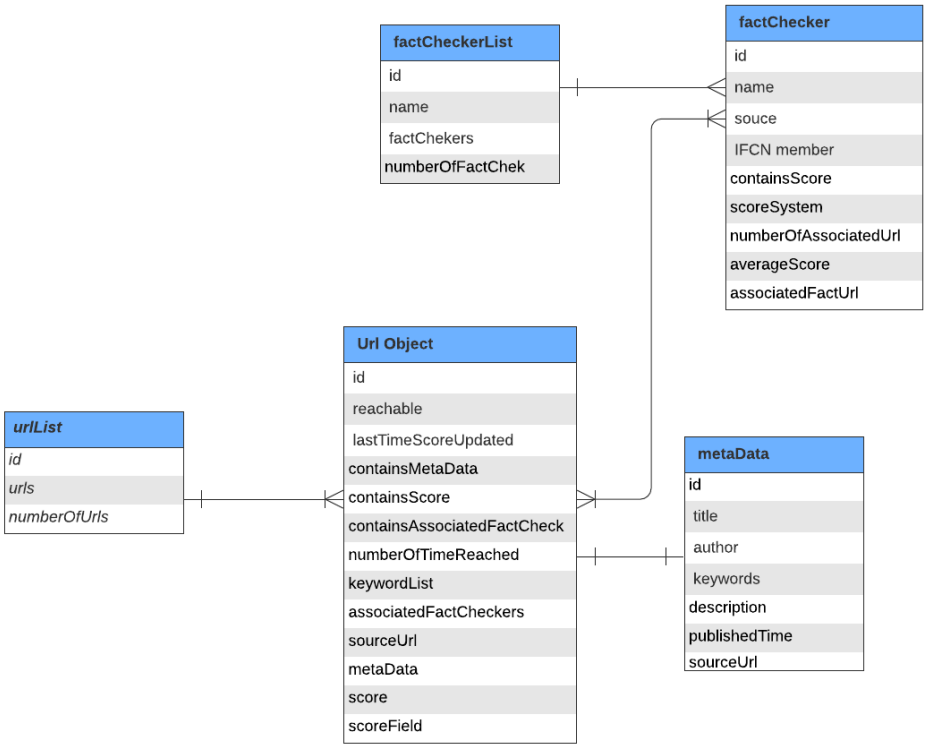


Figure 4.2-UML schema de la Base de données

On a 5 types d'objets stockés dans MongoDB.

Les donnés de relatives d'authentification sont stockées localement dans le fichier .env.

4.3 Fonctions et opérations

Nous avons opté pour une approche fonctionnelle. Les fonctions sont pour la plupart regroupées dans des fichiers selon leurs fonctionnalités et chaque tâche est déléguée à une fonction:

Les actions principales sont contenues dans les fichiers suivants:

- *contains.js*: fonctions validant la présence de données dans la BD
- *get.js* : fonctions permettant d'obtenir des valeurs de la BD
- set.js : fonctions permettant d'inscrire des valeurs dans la BD
- *score.js :* fonction relative au calcul du score.

Figure 4.3 - Interface visuel du programme

Le point d'entrée d'exécution est la fonction *main.js* .

Suite à l'entrée de l'URL dans la barre de saisie de l'interface graphique, différents cas sont pris en compte et des valeurs différentes sont retournées:

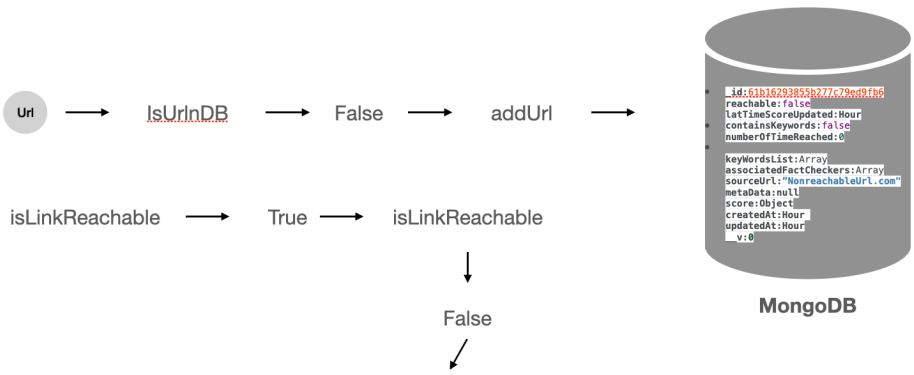
1. **Requête inconnue** → {"score":*null*,"code":*100*,"URL":URL,"message":"unknown Process"}
2. **Lien non joignable** → { "score":*null*,"code":*10*,"URL":URL,"message":"Link cannot be reached" }
3. **Métadonnées non extraites** → { "score":*null*,"code":*11*,"URL":URL,"message":"No metaData was extracted from the website " }
4. **Aucun fait correspondant** → { "score":*null*,"code":*12*,"URL":URL,"message":"No claims extracted from the API Extracted it is empty" }
5. **Fact-chekers non dans la BD**→ {"score":*null*,"code":*13*,"url":URL,"message":"The URL has no associated fact-checkers score"}
6. **Score calculé** →{"score":score,"code":*14*,"url":URL,"message":"The score has been computed"}
7. **Score mis à jour** →{"score":score,"code":*15*,"url":URL,"message":"The score has been updated "}
8. **Score non mis à jour** →{"score":*null*,"code":*16*,"url":URL,"message":"No different between current and precedent score"}
9. **Erreur** →{success:false, data:[], error:error.message}

4.4 Uses cases et données expérimentales

Le code a été exécuté sur un MacBook Pro 2020 avec les caractéristiques suivantes:

macOs version 11.2
Mémoire : 8GB 133 MHz LPDDR3
Processeur: 1,4 GHz Intel Core i5, 4 coeurs

4.4.1 URL non joignable



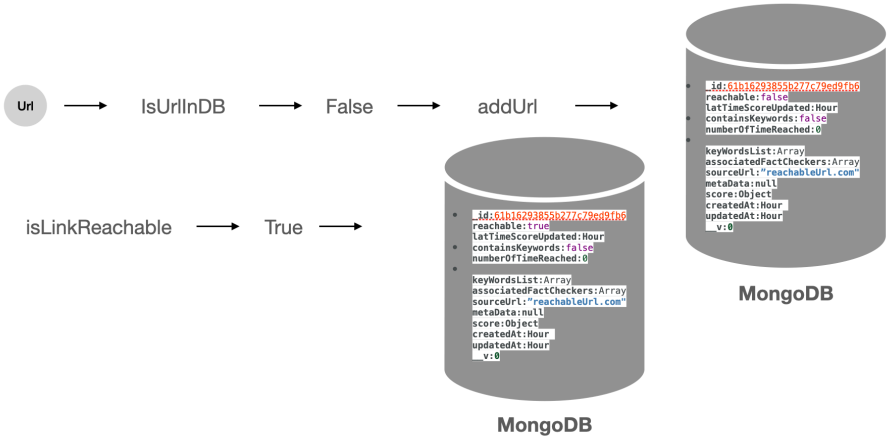
{"score":null,"code":10,"url":NonreachableUrl.com,"message":"Link cannot be reached"}

Pour l'input URL:"http://zxlllzcn.com/" , le temps d'exécution moyen est de **157 ms** (sur 20 mesures).

Lorsqu'on exécute la requête après que l'objet ait été créé en base de données le temps d'exécution moyen est de **80.22 ms** (sur 10 mesures)

4.4.2 URL joignable et non présente dans la base de données

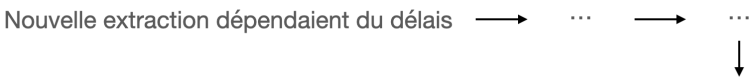
Considérons à présent un lien joignable à partir duquel on peut extraire les métadonnées. Il y aura beaucoup plus d'échanges avec la base de données.

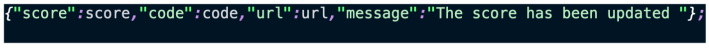


On obtient finalement notre réponse :
{“score":score,"code":14,"URL":URL,"message":"The score has been computed"}

L’exécution du programme prend un temps relativement important. Ce temps dépend entre autres du nombre de requêtes générées et du nombre de claims retournés par requête.

Cela est dû à la communication avec les serveurs de *Sentence-Transformers* pour chacune des claims évalués.





Si on exécute encore la même requête, le temps de réponse est beaucoup plus court cette fois parce que les différentes valeurs sont déjà présentes. Il ne s’agira que d’une opération de lecture.

Par contre si la requête est effectuée après un interval fixé (INTERVAL) , le programme effectue encore une nouvelle extraction et met à jour les objets et le score final.

Pour l'input URL : "https://www.bbc.com/news/world-africa-59337953" , le temps d'exécution moyen pour l'extraction et la vérification sémantique est de **16.23 min**(10 mesures)**.**

Le temps d'exécution moyen pour une requête exécutée juste après est de **1.21s** (10 mesures).

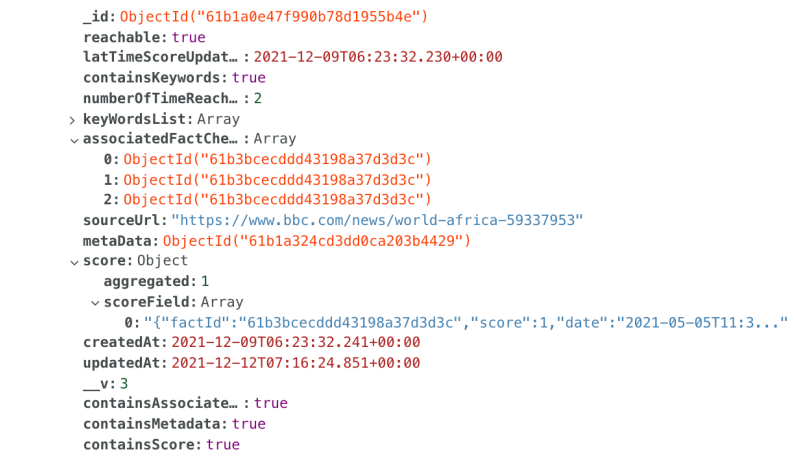
L'objet métadonnée extrait est le suivant :


```
_id: ObjectId("61b1a324cd3dd0ca203b4429")
title: "Uganda's Kampala bombings: Muslim cleric accused of jihadist links sho..."
author: null
keywords: null
description: "Sheikh Muhammad Abas Kirevu was killed by security forces, who said he..."
publishedTime: null
modifiedTime: null
sourceUrl: "https://www.bbc.com/news/world-africa-59337953"
createdAt: 2021-12-09T06:33:08.523+00:00
updatedAt: 2021-12-09T06:33:08.523+00:00
__v: 0
```


Requêtes générées (65):


```
0: "Uganda's"
1: "Kampala"
2: "bombings:"
3: "Muslim"
4: "cleric"
5: "accused"
6: "of"
7: "jihadist"
8: "links"
9: "shot"
10: "dead"
11: "Uganda's Kampala"
12: "Kampala bombings:"
13: "bombings: Muslim"
14: "Muslim cleric"
15: "cleric accused"
16: "accused of"
17: "of jihadist"
18: "jihadist links"
19: "links shot"
20: "shot dead"
21: "Uganda's Kampala bombings:"
22: "Kampala bombings: Muslim"
23: "bombings: Muslim cleric"
24: "Muslim cleric accused"
25: "cleric accused of"
26: "accused of jihadist"
27: "of jihadist links"
28: "jihadist links shot"
29: "links shot dead"
30: "Uganda's Kampala bombings: Muslim"
31: "Kampala bombings: Muslim cleric"
32: "bombings: Muslim cleric accused"
33: "Muslim cleric accused of"
34: "cleric accused of jihadist"
35: "accused of jihadist links"
36: "of jihadist links shot"
37: "jihadist links shot dead"
38: "Uganda's Kampala bombings: Muslim cleric"
39: "Kampala bombings: Muslim cleric accused"
40: "bombings: Muslim cleric accused of"
41: "Muslim cleric accused of jihadist"
42: "cleric accused of jihadist links"
43: "accused of jihadist links shot"
44: "of jihadist links shot dead"
45: "Uganda's Kampala bombings: Muslim cleric accused"
46: "Kampala bombings: Muslim cleric accused of"
47: "bombings: Muslim cleric accused of jihadist"
48: "Muslim cleric accused of jihadist links"
49: "cleric accused of jihadist links shot"
50: "accused of jihadist links shot dead"
51: "Uganda's Kampala bombings: Muslim cleric accused of"
52: "Kampala bombings: Muslim cleric accused of jihadist"
53: "bombings: Muslim cleric accused of jihadist links"
54: "Muslim cleric accused of jihadist links shot"
55: "cleric accused of jihadist links shot dead"
56: "Uganda's Kampala bombings: Muslim cleric accused of jihadist"
57: "Kampala bombings: Muslim cleric accused of jihadist links"
58: "bombings: Muslim cleric accused of jihadist links shot"
59: "Muslim cleric accused of jihadist links shot dead"
60: "Uganda's Kampala bombings: Muslim cleric accused of jihadist links"
61: "Kampala bombings: Muslim cleric accused of jihadist links shot"
62: "bombings: Muslim cleric accused of jihadist links shot dead"
63: "Uganda's Kampala bombings: Muslim cleric accused of jihadist links sho..."
64: "Kampala bombings: Muslim cleric accused of jihadist links shot dead"
65: "Uganda's Kampala bombings: Muslim cleric accused of jihadist links sho..."
```

Url Object :



4.5 Set up et utilisation

Le code source est accessible sur GitHub.[6]
Pour utiliser le programme, il faut télécharger les fichiers et suivre les instructions suivantes.

0. Installer Node JS
1. Installer les différentes dépendances pythons: Bs4, Numpy, Pandas, Sentence transformer, Tesseract
2. Installer les différents packages NPM dans package.json

3. Dans Env
   - Set up un compte MongoDB atlas et le connecter au programme (dbURI)
   - Créer un compte google API et configurer la clef secrète
   - Mettre ses identifiants EMAIL et Password pour obtenir les notifications mail
   - Définir l'interval de score update.
4. Dans la base de données:
   Insérer des objets Fact-Checkers avec leurs systèmes de score tels que spécifiés dans models/factCheckers

Il est possible d'utiliser chacune des fonctions individuellement avec postman en utilisant les routes définies dans / routes.

5.Discussions et amélioration potentielles

Somme toute nous sommes assez satisfaits des tâches accomplies. Le travail théorique offre une base pour quiconque aimerait implémenter son propre modèle. Cependant quelques éléments pourraient être améliorés:

Solution :
- réduction de la complexité de l'algorithme de génération de requêtes
- Modèle automatique de conversion de score de Fact-checker vers score unifié numérique

Implémentation :
- portabilité et déploiement (peut être utilisé docker)
- Réduire le temps d'exécution du programme:
  - Temps de communication avec les API trop importants
  - Système de cache limité
  - Meilleure utilisation de l'asynchronisme
- Raccorder le module d'extraction de texte aux images au reste du programme
- Implémenter la v.2 de l'algorithme d'agrégation
- Étendre les champs extraits lors de l'extraction de métadonnées.

6. Reference

[1]. Thorne, J., Vlachos, A.: Automated fact checking: Task formulations, methods and future directions. In: Proceedings of the 27th International Conference on Computational Linguistics. pp. 3346{3359 (2018).

[2]. IFCN signatories list, adress: https://ifcncodeofprinciples.poynter.org/signatories, (accessed:15-12-21)

[3]. Sentence Transformer Documentation, adress: https://www.sbert.net/, (accessed:15-12-21)

[4]. Malik, Zaki & Bouguettaya, Athman. (2009). Rater Credibility Assessment in Web Services Interactions. World Wide Web. 12. 3-25. 10.1007/s11280-008-0056-y.

[5]. Shema Org, address : https://schema.org/docs/schemas.html(accessed: 21-12-21)

[6]. Armel Ivan Bado, Lien vers le Github contenant l'implémentation du projet, adress: https://github.com/armelivan/factCheckerCode, (accessed: 21-12-21)

[7]. Armel Ivan Bado , Lien vers le blog récapitulant l'avancée du projet : https://armelivan.github.io/