

UNIVERSITE DE MONTPELLIER  
RAPPORT DE PROJET

## Claims checking

*Belkassim BOUZIDI*  
*Chakib ELHOUITI*  
*Massili KEZZOUL*  
*Abdelkader NEDJARI*  
*Ramzi ZEROUAL*

*Encadrant :*  
*M<sup>r</sup> Konstantin TODOROV*



UNIVERSITÉ  
DE MONTPELLIER



10 mai 2020

# Remerciements

Tout d'abord nous souhaitons adresser nos remerciements au corps professoral et administratif de la faculté des sciences de Montpellier qui déploient des efforts pour assurer à leurs étudiants une formation actualisée.

En second lieu, nous tenons à remercier notre encadrant M<sup>r</sup> Konstantin TODOROV pour ses précieux conseils et son aide durant toute la période du travail.

Nos vifs remerciements vont également aux membres du jury pour l'intérêt qu'ils ont porté à notre projet en acceptant d'examiner notre travail.

Nous remercions M<sup>r</sup> Yahia Zeroual pour sa relecture attentive de ce rapport.

# Table des matières

<b>1</b>	<b>Organisation du projet</b>	<b>3</b>
1.1	Méthodes d'organisation . . . . .	3
1.2	Découpage du projet . . . . .	3
1.2.1	Phase de modélisation . . . . .	3
1.2.2	Phase de développement . . . . .	3
1.2.3	Finalisation du projet . . . . .	3
1.3	Outils de collaboration . . . . .	4
<b>2</b>	<b>Introduction au sujet</b>	<b>5</b>
2.1	Fact-checking . . . . .	5
2.1.1	Présentation du principe de fact-checking . . . . .	5
2.1.2	Présentation de ClaimsKG . . . . .	5
2.1.3	Travail à réaliser . . . . .	6
2.2	Technologies utilisées . . . . .	6
<b>3</b>	<b>Conception et implémentation du projet</b>	<b>7</b>
3.1	Conception et Modélisation . . . . .	7
3.1.1	Analyse de ClaimKG . . . . .	7
3.1.2	Conception . . . . .	9
3.2	Implémentation . . . . .	10
<b>4</b>	<b>Analyse des résultats</b>	<b>12</b>
4.1	Résultats . . . . .	12
4.2	Problèmes rencontrés . . . . .	13
<b>5</b>	<b>Bilan et conclusions</b>	<b>14</b>
5.1	Perspective . . . . .	14
5.2	Conclusion . . . . .	14
<b>A</b>	<b>Annexe</b>	<b>15</b>
A.1	Code de la méthode <i>extract_claim_and_review</i> . . . . .	15
A.2	Code de la traduction . . . . .	15
A.3	Code de la méthode <i>retrieve_urls</i> . . . . .	16
A.4	Code de la méthode <i>extract_links</i> . . . . .	17

# Chapitre 1

## Organisation du projet

### 1.1 Méthodes d'organisation

Afin de mener à bien le développement du projet, nous avons décidé de travailler un maximum de temps ensemble et de manière très régulière. Nous nous sommes réunis trois à quatre fois par semaine, en vue de faire le point sur l'avancement du projet et de définir les objectifs restants à atteindre.

Ainsi, selon l'état de progression de la conception, nous réalisions les tâches en retard durant le week-end pour ne pas cumuler de retard et respecter l'intégralité du cahier de charges.

Toutes les semaines, nous nous sommes réunis avec notre encadrant, M<sup>r</sup> Konstantin TODOROV. Lors de ces réunions de mises au point relatifs au projet, nous furent prodigués, cela nous a permis de bénéficier de précieux conseils.

### 1.2 Découpage du projet

Nous avons découpé la réalisation du projet en trois grandes phases :

#### 1.2.1 Phase de modélisation

Durant cette étape, nous nous sommes réunis pour définir les fonctionnalités demandées par le projet. Notamment séparer les fonctionnalités importantes de celles moins importantes. Nous avons également choisi les outils de travail collaboratifs et les principales technologies utilisées, ainsi qu'une première modélisation du projet.

#### 1.2.2 Phase de développement

Durant cette phase, nous avons commencé à implémenter les différentes fonctionnalités que nous avons modélisées lors de l'étape précédente, tout en améliorant la modélisation au fur et à mesure de l'avancement de notre projet. Nous avons notamment réalisé des tests pour les différents modules afin de s'assurer de leur bon fonctionnement.

#### 1.2.3 Finalisation du projet

Cette étape a consisté en la réalisation des tests finaux afin de s'assurer que les différents scripts fonctionnent en toute circonstance et éventuellement corriger les bogues qui peuvent apparaître.

### 1.3 Outils de collaboration

Afin de s'organiser, nous avons décidé d'utiliser Git à travers le serveur GitLab hébergé par le service informatique de la faculté. En effet le logiciel libre Git a facilité grandement la collaboration entre nous. Le serveur GitLab quant à lui est fourni gratuitement par le service informatique de la faculté.

En ce qui concerne la rédaction de ce rapport, nous avons utilisé  $\text{\LaTeX}$ , système de composition de documents créé par Leslie Lamport, pour faciliter la rédaction à plusieurs.



Schéma 1.1 – Logo du GitLab



Schéma 1.2 – Logo de Latex

# Chapitre 2

## Introduction au sujet

### 2.1 Fact-checking

#### 2.1.1 Présentation du principe de fact-checking

Le fact-checking ou la vérification des faits, est une technique consistant à vérifier la véracité des faits et l'exactitude des chiffres présentés dans les médias, les différents réseaux sociaux, les blogs, etc... Cette notion est apparue aux États-Unis dans les années 1990. Elle a été mise en pratique par des journalistes d'investigation dans le cadre de leur profession, la méthode s'est démocratisée grâce à des logiciels aidant les particuliers à vérifier les faits.

L'entrée officielle du fact-checking en France date de 1995, quand est créée l'association Acrimed, qui se présente comme « l'observatoire des médias ».

En vue de la prolifération très rapide des informations, il devient de plus en plus important de s'assurer de la véracité des informations qui se trouvent partout sur Internet et autres médias. Tant du point de vue de la société que de celui de la recherche. De nombreuses approches récentes dans diverses communautés scientifiques portent sur des problèmes tels que la vérification des faits, la détection de la pertinence ou de point de vue des documents par rapport à des revendications particulières.

#### 2.1.2 Présentation de ClaimsKG

Le LIRMM<sup>1</sup> en collaboration avec 2 équipes allemandes (L3S Hannover et l'institut de sciences sociologiques GESIS à Cologne) a construit et mis à disposition la base de connaissance ClaimsKG<sup>2</sup> qui regroupe les informations et méta-données provenant d'un grand nombre de sites journalistiques internationaux de fact checking, tels que Politifact<sup>3</sup> ou Snopes<sup>4</sup>. ClaimsKG est un graphe de connaissances d'assertions annotées et liées qui facilite la création de requêtes structurées sur les assertions, leurs valeurs de vérité (True, Mostly False, etc...), leurs auteurs, date de publication, etc... ClaimsKG est généré par un pipeline entièrement automatisé qui collecte des assertions et des métadonnées à partir des sites de fact-checking, il transforme les données en graphes de connaissances selon un modèle établi, et annote les assertions avec des entités DBpedia (Wikipedia). La base actuelle comprend plus de 32 000 assertions publiées depuis 1996 et est mise à jour régulièrement.

---

1. Le Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier

2. <https://github.com/claimskg>

3. <https://www.politifact.com/>

4. <https://www.snopes.com/>

### 2.1.3 Travail à réaliser

Le sujet du TER consiste en l'enrichissement de cette base de connaissances avec des nouvelles données provenant des sites web suivants :

**Fatabyyano** <sup>5</sup> est un site jordanien en arabe. Fatabyyano (terme en arabe qui veut dire "Alors montrez-le") est la première et la seule plateforme arabe certifiée par l'IFCN <sup>6</sup> ;

**Vishvas.news** <sup>7</sup> est un site Internet de vérification des faits multilingue qui s'engage à combattre la désinformation et les informations erronées.

Le but du TER sera d'identifier les assertions individuelles dans chaque histoire, ainsi que leur label de véracité et par la suite identifier les relations entre elles, ainsi que les relations entre ces histoires. Les données produites par ce projet seront intégrées à la base de connaissance ClaimsKG.

La principale difficulté qu'on s'attend à rencontrer est la gestion des différentes langues proposée par ces sites web. Notamment dans l'identification des différentes relations entre les assertions. En effet afin de reconnaître les différents mots-clés du sujet traité par les articles, on utilise habituellement TAGME, un puissant outil de reconnaissance d'entités nommées dans un texte. Les langues utilisées par ces sites web ne sont pas prises en charge par cet outil. Il s'agit donc de trouver une alternative afin de reconnaître ses différentes assertions. Ce problème ainsi que sa résolution seront détaillés plus tard dans ce rapport.

## 2.2 Technologies utilisées

Nous avons implémenter l'application en Python, choix qui s'impose de lui même car couplé à la bibliothèque BeautifulSoup, il devient très facile de faire du web scraping <sup>8</sup>.

En ce qui concerne la traduction, nous avons choisi d'utiliser Yandex Translator <sup>9</sup>. En effet en vue de la taille des données à traduire, nous n'avons pas pu utiliser l'API de traduction de Google puisque cette dernière devient payante à partir d'un certain nombre de caractères.

Enfin, la reconnaissance des différents entités concernées est faite grâce à l'outil TAGME <sup>10</sup> a travers son API python <sup>11</sup>.



Schéma 2.1 – Logo de Python



Schéma 2.2 – Logo de BeautifulSoup



Schéma 2.3 – Logo de Yandex

5. <https://fatabyyano.net/>

6. International Fact-Checking Network : <https://www.poynter.org/ifcn/>

7. <https://www.vishvasnews.com/>

8. Le web scraping est une technique d'extraction du contenu de sites Web, via un script ou un programme

9. <https://translate.yandex.com/>

10. <https://tagme.d4science.org/tagme/>

11. Lien Github vers l'api Python de TAGME : <https://github.com/marcocor/tagme-python>

## Chapitre 3

# Conception et implémentation du projet

### 3.1 Conception et Modélisation

#### 3.1.1 Analyse de ClaimKG

Lors de cette première phase, le plus important a été de comprendre et s’imprégner du code et de la structure déjà mis en place et mis à notre disposition (Claimskg), comprendre les outils utilisés ainsi que la structure déjà établie.

##### La structure

Comme le projet est d’une envergure immense, bien comprendre la structure était primordiale afin de respecter un maximum les outils utilisés. Notre encadrant nous a mis à disposition un écrit (que vous trouverez dans le dossier autre/ sous le nom de Claimskg.pdf) qui explique la structure globale du projet dont voilà ci-dessous un résumé.



Schéma 3.1 – ClaimKG

Comme vous pouvez le voir sur le schéma, Claimskg est découpé en plusieurs phases. La première consiste à extraire les données brutes à partir de site web de fact-checking et ensuite les structurer afin de produire un fichier CSV. La deuxième phase quant à elle, consiste à faire passer ce fichier CSV dans un générateur de graphes de connaissances<sup>1</sup>. Dans notre cas, on s’occupe de la première phase, c’est-à-dire, extraire les données brutes du site web afin de produire le fichier CSV.

Le fichier CSV en question contient l’ensemble des faits traités par un site web. Chaque fait est passé à un script de scraping afin d’extraire chaque information et la structurer. La structure d’un fait est réalisée selon un modèle précis dont voici une partie :

**rating value** : La valeur de véracité du fait ;

---

1. Knowledge Graph Generator





### 3.1.2 Conception

Dans cette phase, nous allons présenter les étapes de conception que nous avons suivies pour bien implémenter notre application. La première étape a été de concevoir quelques diagrammes UML utiles à la bonne compréhension du projet.

#### Diagrammes UML

Lors de la phase de la modélisation, nous avons vu qu'il était primordial de concevoir un diagramme de classe pour connaître la manière dont les classes interagissent. Comme la figure suivante le montre, des classes ont été rajoutées par rapport aux classes déjà développées.

Dans les sous-classes de FactCheckingSiteExtractor, la méthode "extract\_claim\_and\_review" a été implémentée pour transformer une page donnée en paramètre en une instance de claim. La méthode "get\_all\_claims" quant à elle, extrait les liens des articles du site web. Les données, une fois extraites seront stockées dans un fichier CSV où chaque ligne représente un article et chaque colonne les informations le concernant.

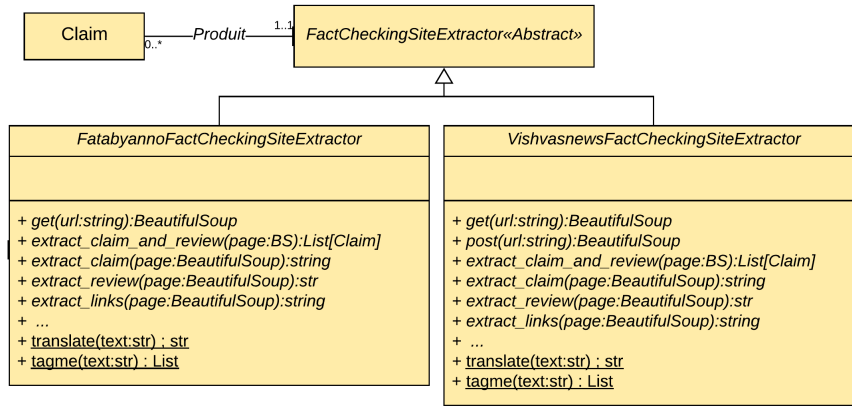


Schéma 3.4 – Structuration de claimsKG-extractor

Le déroulement de l'extraction des données a été quant à elle modélisée par un diagramme de séquence. La figure ci-dessous représente le fonctionnement de l'extraction du site Fatabayano.

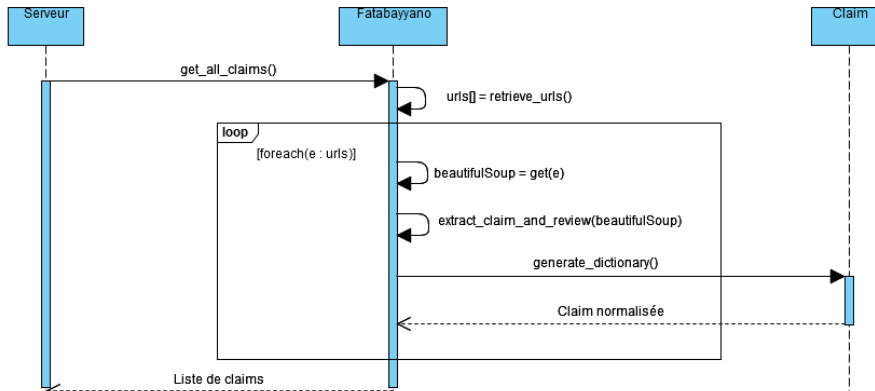


Schéma 3.5 – Structuration de claimsKG-extractor

## Représentation des données

La deuxième étape consistait à trouver un site web respectant les critères de fact-checking et qui nous permet d'extraire les données essentielles (la véracité, les claims écrites, un auteur, etc...) au noyau du projet (ClaimsKG). Pour apporter un maximum de diversité ainsi que notre contribution personnelle, nous avons décidé, avec l'accord de notre encadrant, d'extraire les données d'un site d'une langue qui n'est pas encore présente dans le projet. Vu que tous les étudiants de ce projet sont arabophones, la langue arabe était un choix logique. Le seul site officiel de fact-checking en arabe est : <https://fatabyyano.net/>.

Nous avons donc analysé la structure du site web puis commencé son implémentation. L'étape suivante fût de trouver comment faire une "named entity recognition" sur les mêmes principes que celles déjà présentées (TagMe qui renvoie vers des liens wikipedia).

Le second site web sur lequel nous avons travaillé est <https://www.vishvasnews.com/>, il regroupe 11 langues différentes : L'anglais, Hindi, Pendjabi, Ourdou, Bengali, Tamoul, Malayalam, Goudjerati, Télougou, Marathi et L'odia , nous avons analysé la structure du site puis extrait ses données.

## 3.2 Implémentation

Tout comme la partie conception, nous avons décidé de répartir l'implémentation du projet en X parties

Nous avons commencé par implémenter les classes que nous avons modélisées dans la partie conception, puis nous nous sommes concentrés sur la manière d'extraire les liens de chaque article. Pour cela, nous avons tout d'abord cherché dans le site où et comment les liens sont présentés en inspectant les pages web ainsi que leur code source. Une fois les liens trouvés et après avoir bien cerné la structure du site web, nous avons commencé à écrire la méthode « get(url:string) » qui permet, à partir d'un lien URL de retourner une instance de la classe « BeautifulSoup ». Cette classe permet de garder en mémoire le code source d'une page web de façon structurée, et qui permet avec ses méthodes, de recherche et de manipulation. Elle facilite aussi grandement l'extraction des données. Une fois cette méthode achevée, on a pu commencer à écrire la méthode « retrieve\_url(page:BeautifulSoup) » qui recupere l'ensemble des liens des articles.

L'ensemble des liens maintenant récupéré, nous avons implémenté les méthodes d'extraction - « extract\_claim(page:BeautifulSoup) », « extract\_date(page:BeautifulSoup) »...etc - en analysant la position de chaque information une à une dans la structure du code source.

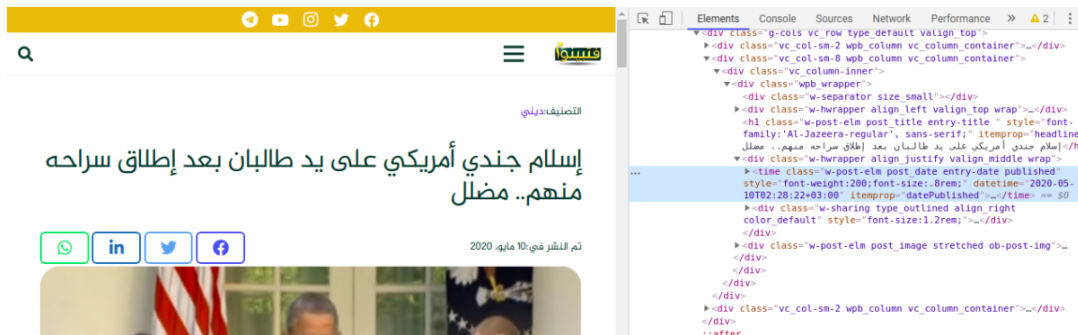


Schéma 3.6 – Extraction de la date du site web Fatabyyano

Ensuite une fois les données extraite du site, nous avons réfléchi à un moyen d'extraire les entités présentes dans la claim et dans la review de l'article. Pour se faire, nous avons cherché une alternative à TAGME pour la langue arabe (pour le site web Fatabyyano), après plusieurs recherches et tests, nous n'avons trouvé aucune solution plausible pour le remplacer. Nous avons opté pour une traduction de l'arabe vers l'anglais en utilisant une API de Yandex, pour ensuite





[illegible][illegible]

## 4.2 Problèmes rencontrés

Le problème majeur suivant a été de trouver un moyen de faire une « named entity recognition » en langue arabe, car le programme "tagme" ne peut être utilisé sur l'arabe. Nous avons donc cherché des logiciels en open source qui pouvait nous aider à résoudre ce problème, malheureusement, aucun logiciel n'a été satisfaisant ; certains nous ont permis comme "Arabic NER" ou "NERAr" de faire des entity recognition, mais sans retourner les liens Wikipedia de ces derniers, d'autres comme "AiDA" qui semblaient être la solution ne fonctionnaient tout simplement pas. Nous avons donc décidé de traduire les claims et leurs reviews afin d'appliquer le programme "tagme", mais là aussi, les API de traduction ne marchaient pas correctement comme googletrans (googletrans est différente de google translate qui n'est pas gratuite) ou étaient limitées par jour comme celle que nous avons utilisé "Yandex". Le souci avec cette api était que le nombre de caractère traduit quotidiennement était limité.

## Chapitre 5

# Bilan et conclusions

### 5.1 Perspective

Le scrapping des sites s'effectue parfaitement, aucune amélioration n'est nécessaire au script en lui-même, la seule perspective pour notre projet est la traduction des 10 langues présentes dans le deuxième site non prises en compte par le programme "tagme", deux idées peuvent être avancées :

Continuer à utiliser Yandex malgré ces capacités limitées tel qu'expliqué plus haut en changeant uniquement le script de traduction du premier site web ou utiliser une autre API, de préférence Google Traduction qui est cependant payante, mais dont l'efficacité est sans conteste la meilleure.

### 5.2 Conclusion

Pour conclure, ce projet nous a permis d'apprendre beaucoup de nouvelles notions. Il a renforcé notre travail en groupe et le sens de la communication, de la synchronisation ainsi que de l'organisation. Il nous a aussi permis de découvrir les principes du web scrapping et l'utilisation de nouvelles API et bibliothèques.

# Annexe A

## Annexe

### A.1 Code de la méthode *extract\_claim\_and\_review*

```
0 def extract_claim_and_review(self, parsed_claim_review_page: BeautifulSoup, url
1 : str) -> List[Claim]:
2     self.claim = self.extract_claim(parsed_claim_review_page)
3     self.review = self.extract_review(parsed_claim_review_page)
4
5     claim = Claim()
6     claim.set_rating_value(
7         self.extract_rating_value(parsed_claim_review_page))
8     claim.set_alternate_name(FatabyyanoFactCheckingSiteExtractor.
9         translate_rating_value(
10             self.extract_rating_value(parsed_claim_review_page)))
11     claim.set_source("fatabyyano")
12     claim.set_author("fatabyyano")
13     claim.setDatePublished(self.extract_date(parsed_claim_review_page))
14     claim.set_claim(self.claim)
15     claim.set_body(self.review)
16     claim.set_refered_links(self.extract_links(parsed_claim_review_page))
17     claim.set_title(self.extract_claim(parsed_claim_review_page))
18     claim.set_date(self.extract_date(parsed_claim_review_page))
19     claim.set_url(url)
20     claim.set_tags(self.extract_tags(parsed_claim_review_page))
21     # extract_entities returns two variables
22     json_claim, json_body = self.extract_entities(self.claim, self.review)
23     claim.set_claim_entities(json_claim)
24     claim.set_body_entities(json_body)
25
26     return [claim]
```

### A.2 Code de la traduction

```
@staticmethod
def translate_rating_value(initial_rating_value: str) -> str:
    return {
        "صحيح": "TRUE",
        "رائف جزئياً": "MIXTURE",
        "عنوان مضلل": "OTHER", # ?
        "رأي": "OTHER", # ? (Opinion)
        "ساخر": "OTHER", # ? (Sarcastique)
        "غير مؤهل": "FALSE", # ? (Inéligible)
        "خادع": "FALSE", # ? (Trompeur)
        "رائف": "FALSE"
    }[initial_rating_value]
```

Schéma A.1 – Code traduction



```

0  @staticmethod
1  def translate(text: str) -> str:
2      """
3          :text:  -> The text in arabic
4          :return: -> return a translation of :text: in english
5      """
6      if text == "":
7          return ""
8      self = FatabyyanoFactCheckingSiteExtractor
9      yandexAPI = self.YANDEX_API_KEY
10     yandex = YandexTranslate(yandexAPI)
11
12     responses = []
13     try:
14         response = yandex.translate(text, 'ar-en')
15         responses = [response]
16         text_too_long = False
17     except YandexTranslateException as e:
18         if e.args == 'ERR.TEXT.TOO.LONG' or 'ERR.TEXT.TOO.LONG' in e.args:
19             text_too_long = True
20         else:
21             print("Erreur API Yandex\nCode d'erreur : " + str(e.args))
22             sys.exit(1)
23
24     text_list = [text]
25
26     while text_too_long:
27         text_too_long = False
28         try:
29             text_list = self.cut_str(text_list)
30         except ValueError:
31             print("Erreur ")
32             sys.exit(1)
33         responses = []
34         for t in text_list:
35             try:
36                 responses.append(yandex.translate(t, 'ar-en'))
37             except YandexTranslateException:
38                 text_too_long = True
39                 continue
40
41     text_list = []
42     for r in responses:
43         if int(r['code']) != 200:
44             print(
45                 "Erreur lors de la traduction\nCode de l'erreur : " + r['code']
46             )
47             sys.exit(1)
48         else:
49             text_list.append(r['text'][0])
50
51     return self.concat_str(text_list)
52
53 @staticmethod
54 def tagme(text):
55     """
56         :text:  -> The text in english after translation
57         :return: -> return a list of entities
58     """
59     if text == "":
60         return []
61     tagme.GCUBETOKEN = FatabyyanoFactCheckingSiteExtractor.TAGME_API_KEY
62     return tagme.annotate(text)

```

### A.3 Code de la méthode *retrieve\_urls*

```

0         :number_of_page:      -> number_of_page
1         :return:              -> la liste des url de toutes les claims
2     """
3     links = []
4     select_links = 'ul.listing li div.imagecontent h3 a'
5     # links in the static page
6     claims = parsed_listing_page.select(
7         "div.ajax-data-load " + select_links)
8     for link in claims:
9         if link["href"]:
10             links.append(link["href"])
11
12     # for links loaded by AJAX
13     r = re.compile(
14         "https://www.vishvasnews.com/(.*)/(.*)[/]").match(listing_page_url)
15
16     lang = r.group(1)
17     categorie = r.group(2)
18
19     url_ajax = "https://www.vishvasnews.com/wp-admin/admin-ajax.php"
20     data = {
21         'action': 'ajax_pagination',
22         'query_vars': '{"category-name" : "' + categorie + '", "lang" : "' +
lang + '"}',
23         'page': 1,
24         'loadPage': 'file-archive-posts-part'
25     }
26
27     response = self.post(url_ajax, data)
28
29     while True:
30         claims = response.select(select_links)
31         for link in claims:
32             if link['href']:
33                 links.append(link['href'])
34
35         if response.find("nav"):
36             data['page'] = data['page'] + 1
37             response = self.post(url_ajax, data)
38             continue
39         else:
40             break
41
42     return links

```

## A.4 Code de la méthode *extract\_links*

```

0     def extract_links(self, parsed_claim_review_page: BeautifulSoup) -> str:
1         links = []
2
3         # extracting the main article body
4         review_body = parsed_claim_review_page.select_one(
5             "div.lhs-area")
6         # making a clone
7         review_body = copy.copy(review_body)
8
9         # removing the social-media shares links
10        review_body.select_one('ul.social-icons-details').decompose()
11
12        # removing authors & tag links
13        b = False
14
15        # ( > * ) ==> direct children in css
16        for tag in review_body.select("> *"):
17            if tag.get('class') and "reviews" in tag.get('class'):
18                b = True
19            if b:

```

```
20         tag.decompose()
21     else:
22         continue
23     # extracting links
24     for link_tag in review_body.select('a'):
25         if link_tag.has_attr('href'):
26             links.append(link_tag['href'])
27
28     return self.escape(str(links))
```