

UNIVERSITÉ DE MONTPELLIER - FACULTÉ DES SCIENCES
L2 INFORMATIQUE
ANNÉE UNIVERSITAIRE 2019 - 2020
PROJET INFORMATIQUE — HLIN405

Rapport de projet T.E.R. :

Geoscape

Fusion de données pour afficher
les plus belles photos du monde

Étudiants :

NATHAN DEBART
SABRINA DJEBROUNI
TROY FAU
MARWAN MASHRA

Encadrant :

PASCAL PONCELET



Sommaire

Introduction	2
1 Technologies utilisées	5
1.1 Langages de programmation	5
1.2 Outils	5
2 Développement de Geoscape	6
2.1 Lancement de l'application	7
2.2 Recherche d'images	8
2.2.1 Récupération des liens <i>EarthPorn</i>	8
2.2.2 Extraction des toponymes	9
2.3 Géolocalisation des photos	10
2.4 Affichage des photos sur la carte	12
2.5 Stockage des résultats	13
2.6 Implémentation de Geoscape	13
2.6.1 Architecture logicielle	15
2.6.2 Programmation événementielle	16
2.6.3 <i>Web scraping</i> de Reddit	17
2.6.4 Encapsuler les recherches sur GeoNames	17
2.6.5 Génération de la carte	17
2.6.6 Base de données dans le <i>cloud</i>	19
3 Analyse des Résultats	20
3.1 Les cycles de recherche sur GeoNames	20
3.2 Amélioration de Geoscape	21
3.2.1 Analyse des photos signalées	21
3.2.2 Traitement des résultats	23
4 Gestion du projet	24
4.1 Diagramme de Gantt	24
4.2 Organisation du travail	25
4.3 Réunion avec l'encadrant	25
Conclusion et perspectives	26
Bibliographie	27
Annexes	28

Introduction

Contexte

Dans le cadre du module HLIN405-Projet de Programmation du semestre 4 de la Licence d'informatique, nous avons développé une plate-forme web permettant de visualiser des photos du monde entier sur une carte.

Notre groupe est composé de quatre étudiants, Nathan Debart, Sabrina Djebrouni, Troy Fau et Marwan Mashra, et nous sommes encadrés par M. Pascal Poncelet du LIRMM. La réalisation de ce projet s'est déroulée sur une période de 16 semaines, de mi-janvier à mi-mai 2020.

Motivations du projet

Chaque jour, des millions de photos sont prises et mises en ligne sur différentes plate-formes. Ces photos forment un énorme ensemble de données qui présente beaucoup d'intérêts à la fois du point de vue de la recherche et de l'exploitation commerciale. Un sous-ensemble de ces photos est celui des photos de paysages naturels, à vocation essentiellement esthétique.

Malheureusement, ces photos sont rarement bien présentées sur le web. Elles sont souvent mal organisées, et affichées avec très peu de contexte. Une telle présentation ne les met pas en valeur, nuit à l'expérience de visualisation des utilisateurs et ne retient donc pas assez leur attention. La valeur esthétique des photos est ainsi mal exploitée.

Un moyen de résoudre ce problème est la fusion de données : présenter ces photos sur une plate-forme unique, de manière dynamique et avec un fort impact visuel. Il faut alors déterminer sur quels critères procéder à cette fusion et, une fois mise en place, mesurer son efficacité.

Objectif du projet

L'approche que nous avons retenue est d'enrichir les images en les replaçant dans leur contexte géographique, et ceci de manière graphique. Cet aspect visuel retiendra beaucoup plus l'attention des utilisateurs qu'une description textuelle.

Pour réaliser cet objectif, nous avons décidé de créer un site web capable de récupérer des photos de paysages du monde entier depuis la plate-forme ciblée, puis de les présenter à l'utilisateur en les plaçant sur une carte interactive en fonction des coordonnées géographiques des lieux pris en photo.

Sachant que toute image présente sur le site ciblé est légendée selon un format à peu près standard, le meilleur moyen d'identifier l'endroit pris en photo est d'utiliser ce titre. La figure 1 montre un exemple idéal, où le titre est une phrase simple ne contenant qu'un seul nom de lieu.



FIGURE 1 – Objectif : placer la photo du Mont Blanc aux coordonnées géographiques du Mont Blanc.

Identifier le lieu peut être plus ou moins difficile selon le titre. Dans le cas de phrases qui mentionnent plusieurs endroits (*c.f. figure 2*), comment sélectionner le bon ? Le risque d'erreurs est plus important. Mais la recherche doit pouvoir être améliorée grâce au *feedback* des utilisateurs.

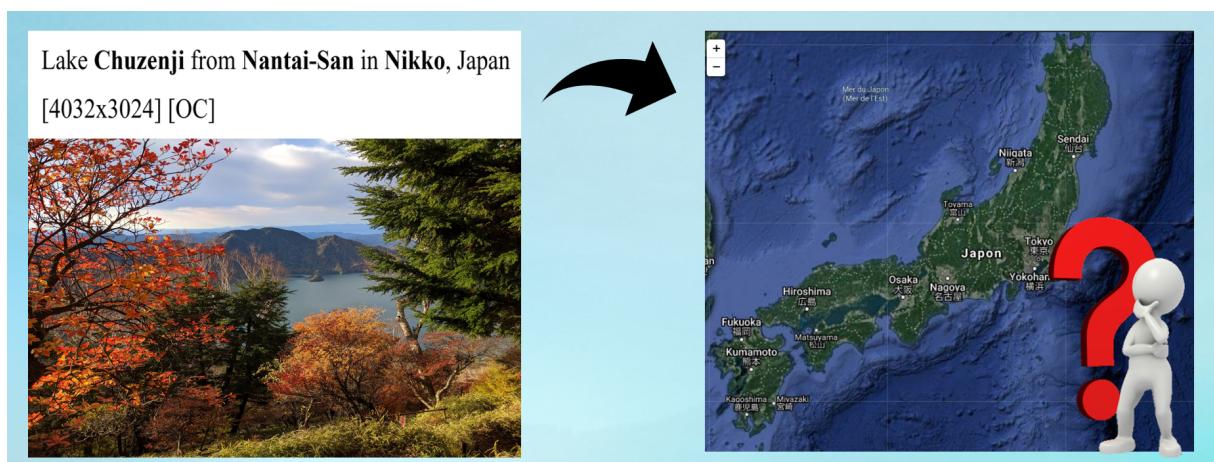


FIGURE 2 – Quel lieu choisir ? Un lecteur humain identifiera rapidement le bon endroit, mais un programme informatique aura besoin d'un traitement plus sophistiqué.

Le cahier des charges

À partir de l'objectif que nous nous sommes donné, nous avons établi le cahier des charges suivant.

Recherche de photos : en réponse à la requête d'un utilisateur, notre application doit pouvoir récupérer les liens vers les photos présentes sur le site ciblé.

Détection du lieu pris en photo : le titre associé à l'image sera analysé et le lieu pris en photo correctement identifié. Les coordonnées géographiques de cet endroit seront alors récupérées sur un site tiers.

Placement des photos sur une carte interactive : notre application doit générer une carte interactive et y placer les photos à leurs coordonnées respectives.

Amélioration de la recherche avec contribution des utilisateurs : pour vérifier l'efficacité de la recherche, le site doit proposer aux utilisateurs la possibilité de signaler une image mal placée. Les utilisateurs experts analyseront les titres des photos mal placées afin d'améliorer la recherche.

Il s'agit donc d'un projet combinant la programmation web interactive, la recherche d'information (plus précisément la recherche d'entités nommées) et la collaboration avec les utilisateurs de Geoscape. Une démonstration vidéo de l'application est disponible à l'adresse <https://www.youtube.com/watch?v=tnf1evhDI74>.

Dans la suite du rapport, nous commençons dans la partie 1 par un bref exposé des langages de programmation et des outils informatiques que nous avons utilisés pour mener à bien le projet. Ensuite, la partie 2 présente concrètement Geoscape, en expliquant le fonctionnement de l'application et son implémentation, et en justifiant les choix effectués.

Nous nous penchons alors dans la partie 3 sur l'analyse et l'amélioration des résultats des recherches et des tests effectués. Enfin, après une présentation de notre gestion du travail dans la partie 4, nous établissons le bilan de notre projet, et évoquons les enseignements que nous en avons tiré ainsi que les perspectives qu'ils nous offre.

Partie 1

Technologies utilisées

Pour développer Geoscape, nous avons utilisé les langages de programmation et les outils suivants.

1.1 Langages de programmation

- **Python** : l'un des langages de script les plus utilisés aujourd'hui. Parmi les nombreux modules qu'il propose on trouve en particulier Flask, Praw et Pymongo. Le premier facilite le développement et le déploiement de notre application, les deux autres simplifient les interactions avec les sites tiers.
- **JavaScript** : un langage côté client omniprésent sur le web. Il nous permet de développer un site dynamique, avec mise à jour des pages sans recharge. La bibliothèque Leaflet en particulier permet de créer des cartes interactives.

1.2 Outils

- **Reddit** : un site web communautaire constitué d'un ensemble de *subreddits* thématiques, où les utilisateurs soumettent des liens et des messages. *EarthPorn*¹, consacré à la photographie de paysages naturels, est la cible de notre projet.
- **TreeTagger**² : un étiqueteur morpho-syntaxique qui analyse un texte et procède à la catégorisation grammaticale de son contenu.
- **GeoNames**³ : une base de données stockant plus de 25 millions de toponymes et les coordonnées géographiques associées.
- **MongoDB Atlas**⁴ : un système de gestion de base de données NoSQL (non dérivé de l'algèbre relationnelle) permettant le stockage des données dans le *cloud*.
- **GitHub** : un service web d'hébergement basé sur le logiciel de gestion de versions Git. Il facilite le développement concurrent grâce au système de branches.

1. <https://www.reddit.com/r/EarthPorn/> (Dernière consultation le 6 mai 2020)

2. <https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/> (Dernière consult. le 6 mai 2020)

3. (<https://www.geonames.org/>) Dernière consultation le 6 mai 2020)

4. <https://www.mongodb.com/cloud/atlas> (Dernière consultation le 6 mai 2020)

Partie 2

Développement de Geoscape

Pour l'utilisateur, notre site se présente sous la forme d'une page web d'où il lance des requêtes pour visualiser des photos. En réponse à sa demande, une carte est générée et les images affichées à leurs coordonnées respectives.

Notre application doit communiquer avec deux sites web : le premier est Reddit, le site où nous effectuons le *web scraping*¹, et plus particulièrement le subreddit *EarthPorn*, consacré aux photos de paysages naturels. Le deuxième est GeoNames, une base de données de géolocalisation où nous récupérons les coordonnées de lieux. La figure 2.1 détaille les étapes de cette communication. La recherche prend un certain temps, et afin de pouvoir rapidement réutiliser les images déjà trouvées pour des affichages ultérieurs, nous stockons les résultats dans une base de données.

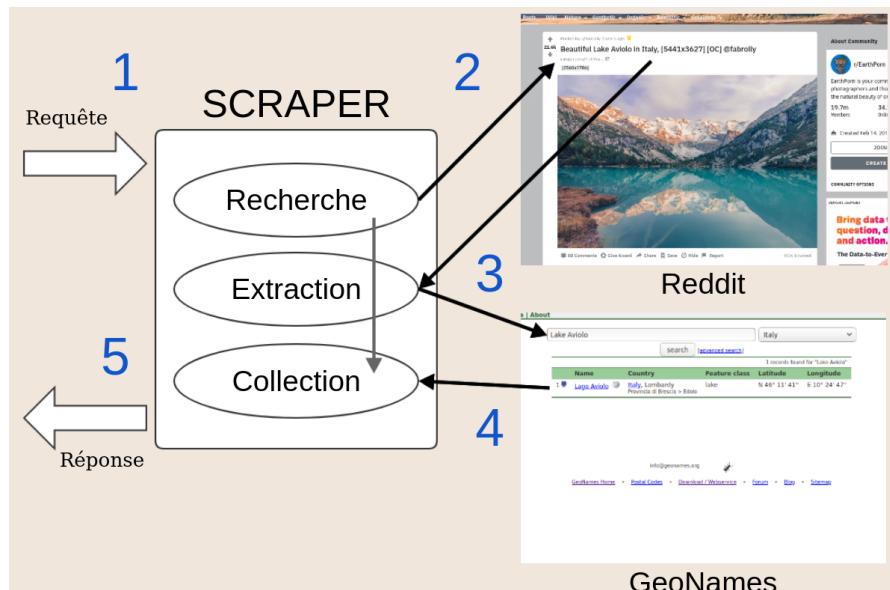


FIGURE 2.1 – Les étapes de la collecte d'images. (1) Une requête d'un utilisateur est reçue depuis notre site. (2) Le *scraper* lance une recherche sur Reddit paramétrée par la requête. (3) Les titres des liens sont extraits et analysés ; nous en tirons des toponymes qui sont géolocalisés sur GeoNames. (4) Les données relatives aux liens sur *EarthPorn* ainsi que les coordonnées trouvées sur GeoNames sont collectées et (5) renvoyées au site.

1. Extraction de données d'une page web.

Dans la suite de cette partie, nous suivons le déroulement des étapes décrites à la figure 2.1 avec l'exemple de la figure 2 de l'introduction : un lien *EarthPorn* vers une photo d'un lac prise depuis une montagne qui le surplombe, avec pour titre « Lake Chuzenji from Nantai-San in Nikko, Japan ». Cela nous permettra de décrire le fonctionnement de Geoscape à l'aide d'un cas pratique.

2.1 Lancement de l'application

La majeure partie de l'interface utilisateur est consacrée à l'affichage des images sur la carte. Ces images sont des photos de paysages naturels, et nous avons donc fait le choix d'une carte basée sur l'imagerie satellite. Elle permet de replacer les photos dans leur environnement physique, ce qui maximise l'impact visuel de notre site.

Afin de mieux contrôler l'expérience de l'utilisateur, nous avons défini les modalités de sa recherche. Comme on peut le voir à la figure 2.2, Geoscape est paramétré par trois options, que nous expliquons ici.

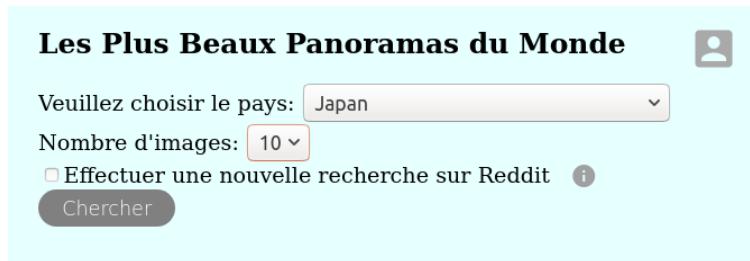


FIGURE 2.2 – L'écran de sélection. Le point de départ de Geoscape.

Le pays : l'utilisateur doit choisir le pays où les photos ont été prises. Ce critère est en adéquation avec les titres des liens sur *Earthporn*, qui incluent généralement le nom du pays, ce qui fournit un motif peu ambigu à filtrer lors de la phase de recherche. Ce critère est également bien adapté à l'affichage : à la génération de la carte, celle-ci est centrée sur le pays cible.

Le nombre de photos : le nombre de photos sur le *subreddit* est non borné. Pour éviter de « noyer » l'utilisateur sous les photos, c'est à lui d'établir une borne supérieure sur le nombre d'images qui lui sont renvoyées.

Lancer un *scraping* : nous stockons les liens déjà trouvés dans une base de données. Ainsi, par défaut, l'utilisateur consulte des photos issues de recherches antérieures et reçoit une réponse rapide à sa requête. Il doit explicitement indiquer qu'il souhaite procéder à un *scraping*, ce qui implique qu'il tolère un certain temps d'attente.

2.2 Recherche d'images

2.2.1 Récupération des liens *EarthPorn*

Les données fournies par l'utilisateur de Geoscape servent à construire la requête vers EarthPorn. Suivant la syntaxe définie par Reddit, cette requête contient la chaîne de caractère <nom du pays>, la forme `title:<nom du pays>`, pourtant autorisée par le site, ne semblant pas fonctionner pour certaines pages (nous avons supposé qu'il s'agissait d'une indexation incomplète de Reddit).

La requête renvoie des liens en résultat, sous forme d'objets JSON². Parmi les éléments stockés dans un objet, on trouve l'adresse du lien, qui correspond à l'adresse de la photo proprement dite, son titre, l'adresse de la page de commentaires sur EarthPorn, l'adresse du compte de l'auteur, la date. Ce sont ces objets que nous traitons, pour en récolter les informations nécessaires à un affichage correct sur la carte.

En premier lieu, nous devons vérifier la pertinence des résultats. Nous filtrons les titres des liens par deux expressions régulières, tout d'abord pour confirmer la présence du nom du pays, et ensuite pour débarrasser les chaînes de caractères des informations techniques relatives à la photo, comme l'illustre la figure 2.3. Ensuite, la chaîne nettoyée peut être analysée pour y trouver des toponymes (noms de lieu) potentiels.

(1)

France

Valle Frances at sunset - Torres del Paine, Chile [OC] [4000x2513]

\WFrance\W

Valle Franees at sunset - Torres del Paine, Chile [OC] [4000x2513]

(2)

Japan

Lake Chuzenji from Nantai-San in Nikko, Japan [4032x3024] [OC]

^(?:[\[\(\)\.*\[\]\)])?)?([^\[\(\]+)

var ← Lake Chuzenji from Nantai-San in Nikko, Japan

FIGURE 2.3 – Filtrage à l'aide d'expressions régulières. (1) La requête France renvoie un titre contenant ce mot comme sous-mot de *Frances*. Mais la *regex* de contrôle ne reconnaît pas la chaîne car elle ne prend en compte le mot (inséré dans l'expression par une variable) que s'il est délimité de chaque côté par un caractère non-alphanumérique. (2) Le jargon technique entre crochets est éliminé par la *regex* de nettoyage, qui ignore au plus un ensemble de caractères délimités par des crochets ou parenthèses en début de chaîne, puis capture le reste jusqu'à rencontrer un.e crochet|parenthèse ouvrant.e.

2. *JavaScript Object Notation*, un format de données très utilisé sur le web pour communiquer entre client et serveur, ou entre applications.

2.2.2 Extraction des toponymes

L'analyse lexicale d'un texte consiste à segmenter la chaîne de caractères selon des règles prédéfinies. Celles-ci définissent des motifs, appelés lexèmes, qui sont reconnus et extraits pour former une liste de tokens (ou unités lexicales). Dans notre cas, les textes sont en langage naturel et les tokens sont des mots ou des signes de ponctuation.

L'étiquetage morpho-syntaxique associe à chaque token d'un texte une étiquette encodant sa nature, au sens grammatical. Ce critère nous est très utile : quelle que soit sa fonction dans une phrase (sujet, complément...), un toponyme est un **nom propre**.

Les étiqueteurs utilisent différents modèles statistiques pour accomplir leur tâche. Celui que nous utilisons, TreeTagger, se distingue par l'utilisation d'un arbre de décision, et a une précision similaire aux étiqueteurs utilisant des modèles de Markov cachés classiques, supérieure à 96% (SCHMID, 1994). Un exemple d'étiquetage est donné dans la table 2.1.

Last	week	,	I	visited	Mont	Blanc	/	France
ORD	NN1	PUN	PNP	VVD	NP0	NP0	PUN	NP0

Etiquettes	
ORD	Adjectif numéral ordinal
NN1	Nom, singulier
PUN	Ponction
PNP	Pronom personnel
VVD	Verbe, passé
NP0	Nom propre
NP0	Nom propre
PUN	Ponction
NP0	Nom propre

TABLE 2.1 – Un premier exemple d'étiquetage morpho-syntaxique, en analysant le texte de la figure 1 de l'introduction avec TreeTagger. L'étiqueteur utilise les *tags* (étiquettes) du *tagset* du British National Corpus.

Les titres des liens sur EarthPorn sont en anglais, et donc l'étiqueteur doit être réglé pour analyser l'Anglais, c'est-à-dire alimenté par les résultats de son entraînement sur un corpus en langue anglaise. Mais les toponymes qui se trouvent dans ces titres peuvent être en n'importe quelle langue, puisque les photos proviennent du monde entier (ils sont néanmoins toujours écrits en alphabet latin).

Est-ce un obstacle ? Selon (CHERNIAK, 1997), les mots inconnus sont le plus souvent des noms propres. Une heuristique utile est la présence ou non d'une majuscule : '*words starting with capital letters are likely to be proper nouns*' (JURAFSKY & MARTIN, 2019). Mais la méthode la plus fiable est l'analyse des suffixes. Or TreeTagger est équipé d'un lexique suffixe (SCHMID, 1994).

Ainsi quand TreeTagger paramétré en anglais rencontre un mot inconnu pour lequel il n'a pas de données, ce qui est le cas pour la grande majorité des toponymes, nous avons émis l'hypothèse qu'il leur attribuerait le plus souvent l'étiquette correspondant à un nom propre.

En conséquence, l'heuristique de base que nous avons adopté pour trouver un lieu est de sélectionner un mot identifié comme étant un nom propre, ou un groupe de nom propres voisins. Nous stockons ces toponymes potentiels dans une liste, avec en sus des nombres entiers représentant les suites de tokens non sélectionnés, comme le montre la table 2.2. C'est à partir des noms contenus dans cette liste que nous lançons nos recherches sur GeoNames.

Lake	Chuzenji	from	Nantai-San	in	Nikko	,	Japan
NP0	NP0	PRP	NP0	PRP	NP0	PUN	CTY
Lake	Chuzenji	1	Nantai-San	1	Nikko		2

TABLE 2.2 – Extraction de toponymes. Rappel : le *tag* NP0 désigne un nom propre. *Japan* est bien vu comme un nom propre par TreeTagger, mais avant le passage de l'algorithme de construction de la liste de lieux nous remplaçons cette étiquette pour ne pas inclure le nom du pays, qui n'est utilisé qu'en dernier recours.

2.3 Géolocalisation des photos

D'après (JURAFSKY & MARTIN, 2019), la présence d'un mot dans un index géographique (*gazetteer*) est un des critères à utiliser lors de la recherche d'entités nommées. Pour nous qui recherchons des lieux, il s'agit de l'étape qui suit l'étiquetage morpho-syntaxique, et nous utilisons à cette fin la base de données GeoNames.

Le moteur de recherche de ce site est imprévisible. En effet, il privilégie en général la géographie humaine et politique, les localités étant prioritaires par rapport aux divisions administratives, qui sont elles-mêmes prioritaires par rapport aux entités géographiques, mais ce n'est pas une constante. Quelques exemples de résultats de recherche sont donnés dans la table 2.3. Même entre entités de même type, GeoNames ne renvoie pas forcément l'entité exacte en premier : dans le dernier exemple de la table, le Main est non seulement très mal classé, mais il n'est même pas le premier cours d'eau.

GeoNames propose néanmoins des fonctionnalités de recherche « avancée » : la recherche de correspondances exactes, et la recherche par *feature class*. Cette dernière se fonde sur le classement par catégories des lieux stockés : villes, divisions administratives, montagnes, îles, etc. Au vu du comportement idiosyncratique du moteur de recherche, nous avons naturellement décidé d'utiliser ces fonctionnalités.

Requête	Pays	Premier Résultat	Rang A	Rang G
Vosges	France	Gérardmer	13	152
Crete	Greece	Heraklion	2	3
Cornwall	United Kingdom	Newquay Cornwall Airport	7	8
Fuji	Japan	Mount Fuji	2	-
Main	Germany	Frankfurt am Main	-	1509

TABLE 2.3 – **Exemples de résultats de requête sur GeoNames.** *Rang A* est le rang du premier toponyme correspondant exactement à la requête et désignant la ville ou la division administrative (par ex., le département des Vosges) ; *Rang G* est le rang du premier résultat exact désignant l’entité géographique (par ex., la chaîne de montagnes des Vosges).

L’algorithme 1 présente l’approche que nous avons retenue. Nous effectuons plusieurs types de recherche, chaque type étant successivement appliquée à tous les noms de la liste de toponymes. Dès qu’un résultat est trouvé, il est sélectionné et le cycle s’arrête. Si aucun lieu n’est trouvé, nous associons à l’image les coordonnées du pays ; c’est un pis-aller, mais il permet d’afficher la photo et laisse le soin à l’utilisateur de signaler le problème.

Les différents types de recherche sont définis dans la partie 3.

Algorithme 1 : algorithme de recherche sur Geonames.

Données : une liste de types de recherche R , une liste de toponymes potentiels T , un pays p , le code GeoNames du pays c

Résultat : un lieu trouvé dans la base de données GeoNames

```

1 lieu ← ∅
2 pour chaque  $r \in R$  faire
3   pour chaque  $t \in T$  faire
4     lieu ← chercher  $(r, t, c)$  dans GeoNames
5     si lieu ≠ ∅ alors
6       retourner lieu
7 lieu ← chercher  $(p, c)$  dans GeoNames
8 retourner lieu

```

Utilisons comme exemple notre liste de la table 2.2. La première recherche est l’intersection de la recherche exacte et des *feature classes* correspondant aux entités naturelles. La première chaîne, « Lake Chuzenji », ne donne aucun résultat. Avec la deuxième, « Nantai-San », un lieu est trouvé. Ses coordonnées géographiques sont le dernier élément dont nous avons besoin pour satisfaire la requête de l’utilisateur ; nous collectons les autres données du lien Reddit et renvoyons l’ensemble vers notre site.

2.4 Affichage des photos sur la carte

Grâce aux données extraites de EarthPorn et de GeoNames et retournées à notre page web au format JSON, nous pouvons procéder à l'affichage.

La carte est générée, centrée sur le pays cible, avec les photos placées à leurs coordonnées respectives. L'utilisateur n'a plus qu'à cliquer sur l'une des images pour la voir s'afficher en plus grand dans la partie gauche de l'écran, comme c'est le cas à la figure 2.4.

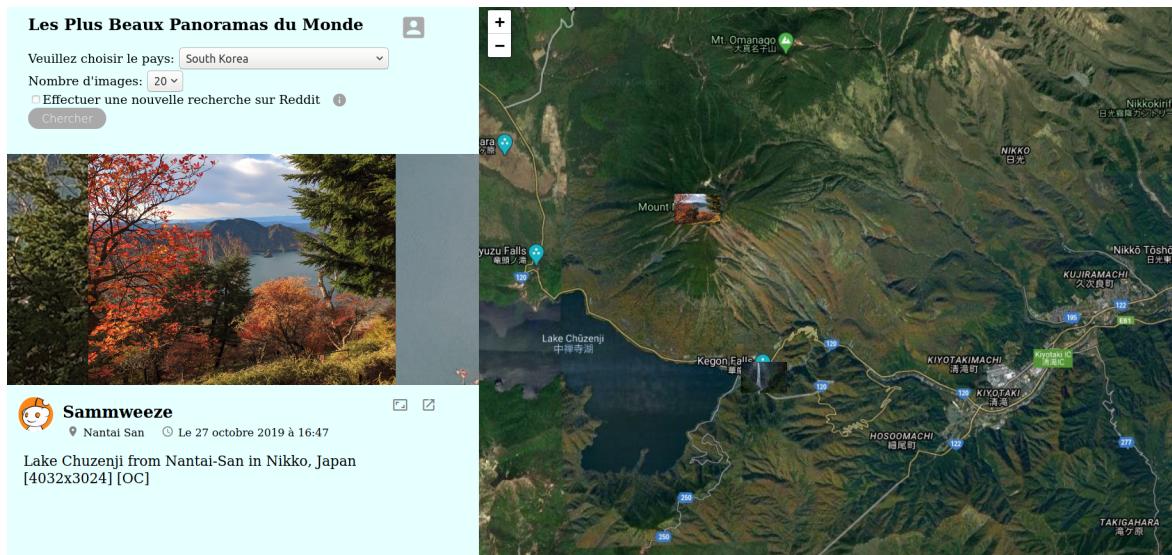


FIGURE 2.4 – La carte interactive de Geoscape. On aperçoit le lac Chuzenji au pied de la montagne ; c'est bien une entité géographique, mais GeoNames ne doit pas stocker le lac sous ce nom exact. C'est donc le lieu suivant dans la liste, le mont Nantai, qui a été identifié par la première recherche.

Si la photo à gauche ne lui suffit pas, l'utilisateur peut consulter l'image en plein écran pour en admirer le moindre pixel. Il peut également suivre le lien vers la page de commentaires sur EarthPorn, ou inspecter le profil du photographe. Bien entendu, à tout moment il peut lancer une nouvelle recherche. Celle-ci se déroule en arrière-plan et la carte et les photos restent affichées.

Mais l'utilisateur un peu plus conscientieux a la possibilité de donner son avis. En effet, il peut signaler le bon ou mauvais placement d'une photo en réponse à la question de la figure 2.5. Une réponse négative est un facteur clé pour améliorer Geoscape : l'utilisateur identifie pour nous un cas précis où l'application n'a pas fonctionné correctement, et nous fournit ainsi une donnée à étudier pour résoudre le problème.

Lake Chuzenji from Nantai-San in Nikko, Japan
[4032x3024] [OC]

Aidez-nous 🙏 | Nantai San, est-il le lieu indiqué dans le commentaire ci-dessus ?
Oui  Non 

FIGURE 2.5 – La demande de *feedback*. La photo à la figure 2.4 a été prise depuis la montagne Nantai, mais son sujet est le lac Chuzenji. Selon l’interprétation de l’utilisateur, il peut répondre dans un sens ou dans l’autre.

2.5 Stockage des résultats

Lorsqu’un lien sur EarthPorn a été traité, nous souhaitons pouvoir réutiliser la photo et ses coordonnées pour répondre à toute requête concernant son pays d’origine, et ce sans avoir à relancer la procédure de recherche. Il nous faut donc réunir les informations issues de la phase de collection et les stocker dans une base de données.

Les données extraites d’un lien Reddit forment un tableau associatif, associant clés et valeurs. Ces valeurs sont de types différents et, provenant d’un lien soumis par un utilisateur quelconque de Reddit, il n’est pas garanti qu’ils soient tous présents. Ses champs sont également dynamiques : lorsque un utilisateur signale sur la carte le bon ou mauvais positionnement de la photo associé à cet enregistrement, nous rajoutons un champs correspondant.

Ces différents facteurs nous ont fait pencher en faveur de l’utilisation d’une base de données conçue sur un modèle plus souple et simple d’utilisation que le modèle relationnel. Notre choix s’est porté sur MongoDB Atlas, une base de données orientée document. Pouvoir directement stocker toutes les informations en rapport avec un lien dans un document simplifie les requêtes, et nous évite d’avoir à faire un mapping objet-relationnel.

2.6 Implémentation de Geoscape

La figure 2.6 illustre bien le fait que l’architecture de Geoscape se décompose facilement selon le modèle client-serveur : d’une part, l’interface utilisateur côté client, d’autre part, le travail de recherche et de traitement des photos côté serveur.

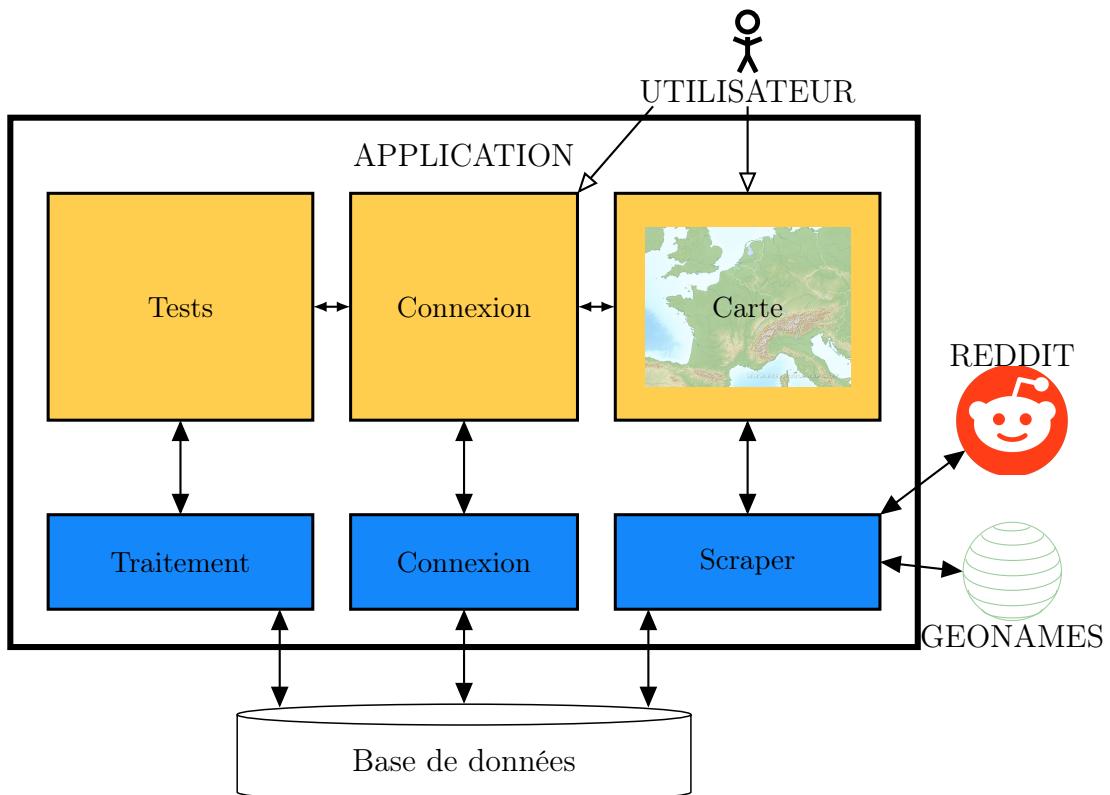


FIGURE 2.6 – L’application Geoscape et son environnement. Les pages web sont en jaune, les modules côté serveur en bleu.

Pour implémenter Geoscape nous avons utilisé le *framework* de développement web Flask³, qui nous sert d’interface entre le site web et le serveur codé en Python. Avec une syntaxe très concise, illustrée à la figure 2.7, Flask nous permet de mettre en relation les URLs (adresses) de notre site avec les différentes fonctions qui traitent les requêtes de l’utilisateur.

```
@app.route('/map.html')
def map():
    return render_template('map.html')
```

FIGURE 2.7 – Une « route » Flask. L’URL concaténant l’adresse du serveur local et la route /map.html est liée à la fonction map(), qui renvoie un document HTML (*stricto sensu* un *template*—patron—contenant du HTML et du code supplémentaire permettant d’y insérer des données).

Nos ambitions étant grandes mais nos moyens logistiques hélas petits, nous avons limité l’accès à notre site au serveur local de l’ordinateur sur lequel Geoscape est installé.

3. <https://flask.palletsprojects.com/en/1.1.x/> (Dernière consultation le 6 mai 2020)

2.6.1 Architecture logicielle

Pour utiliser Geoscape et ses modules Python dépendants en l'isolant de l'installation Python sur une machine donnée, nous avons mis en place un environnement virtuel avec le module `venv` de la bibliothèque standard. Une copie de l'interpréteur Python et les dépendances nécessaires sont placées dans ce répertoire. A l'activation de l'environnement et au démarrage du serveur Flask, les scripts côté serveur sont lus par cet interpréteur et non par l'original.

L'application elle-même est installée dans le répertoire correspondant au serveur local, sous forme de *package* Python tel que Flask puisse le lire correctement. L'agencement du *package* est présenté à la figure 2.8.

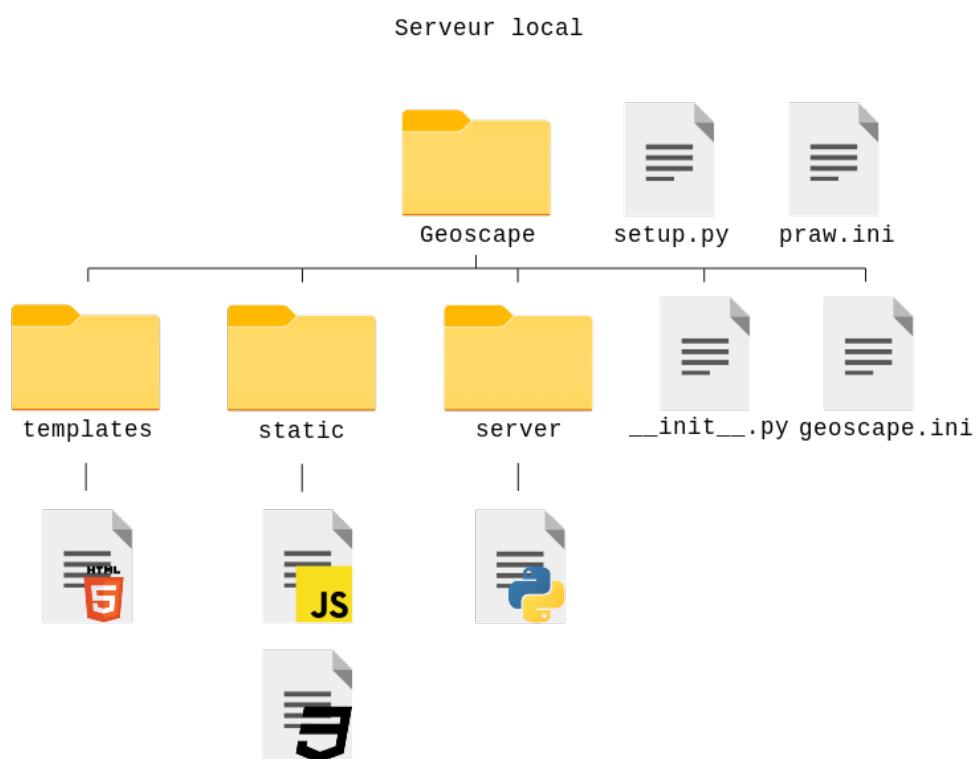


FIGURE 2.8 – L’application Geoscape. Les fichiers `praw.ini` et `geoscape.ini` contiennent nos codes d’accès aux sites tiers. Le répertoire `templates` contient les documents HTML ; `static` contient les fichiers JavaScript ainsi que les feuilles de style CSS ; enfin `server` renferme les fichiers Python.

Par ailleurs, l'utilisation des API de Reddit et de GeoNames nécessite un compte sur chacun des deux sites, et l'accès à notre base de données est protégé par un code d'accès. Ces paramètres secrets ne doivent pas être directement lisibles dans nos scripts, et en conséquence nous les plaçons dans deux fichiers de configuration qui sont lus au lancement de l'application. Cela facilite aussi la prise en main par un nouvel administrateur côté serveur, celui-ci pouvant facilement substituer ses propres identifiants et codes à ceux de son prédécesseur.

2.6.2 Programmation événementielle

Nous avons implémenté l'interface utilisateur avec la méthode Ajax (*Asynchronous JavaScript and XML*), qui permet de développer une application interactive à travers des pages web dynamiques, et ce grâce au langage JavaScript. L'écouteur d'évènement tapi dans le document HTML détecte la demande de l'utilisateur et lance une requête asynchrone en arrière-plan, celle-ci n'interférant pas avec l'affichage. Une fois les liens reçus, la carte est générée sans recharge de la page.

L'outil fondamental pour mener à bien ces échanges est la fonction `$.ajax()` de la bibliothèque JQuery⁴. Toute la communication entre notre site et le serveur repose sur cette fonction, appelée par des évènements différents sur les différentes pages du site. Son rôle dans la requête de l'utilisateur est mise en évidence à la figure 2.9.

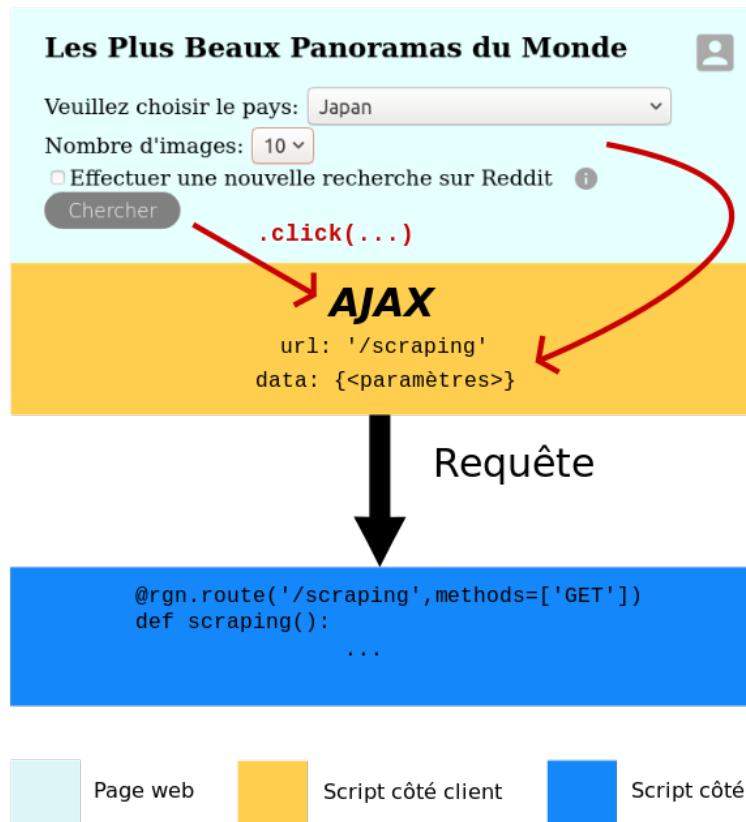


FIGURE 2.9 – Le lancement de Geoscape. A la détection de l'évènement `onclick`, la requête Ajax est dirigée vers l'adresse `/scraping`—liée par Flask à la fonction `scraping()` côté serveur—with en argument les choix de l'utilisateur effectués sur la page. Ceux-ci sont transmis en JSON, ce qui permet au code JavaScript, pour qui c'est un format « natif », de communiquer avec le script Python, qui lui est muni d'un décodeur JSON.

4. <https://jquery.com/> (Dernière consultation le 6 mai 2020)

2.6.3 Web scraping de Reddit

Comme beaucoup de sites aujourd’hui, Reddit met à disposition des développeurs son API (interface de programmation). Une API définit les services offerts par une plate-forme aux applications voulant communiquer avec elle, et permet cette interaction sans exposer son fonctionnement interne.

L’accès à l’API de Reddit par un script Python est facilité par un module *wrapper* de cette API. Une *wrapper library*, ou bibliothèque d’enveloppe, permet de convertir une interface en une forme plus adaptée ou plus simple d’utilisation. Dans notre cas, le module Praw⁵ se charge de la communication avec Reddit et de la conversion des données depuis le format JSON, en encapsulant ces opérations dans des classes et des méthodes des plus pythoniques.

Pour le filtrage par expressions régulières nous utilisons le module Re de la bibliothèque standard Python, qui emprunte sa syntaxe au langage Perl. Pour utiliser TreeTagger nous passons par un autre module Python, Treetaggerwrapper⁶.

2.6.4 Encapsuler les recherches sur GeoNames

Comme Reddit, GeoNames met à disposition son API. Et il existe aussi un *wrapper* Python pour cette interface : Geocoder⁷.

Au vu des particularités de GeoNames, nous utilisons les fonctionnalités avancées du site pour effectuer des requêtes plus ciblées. Mais dans quel ordre ? Afin de ne pas nous enfermer dans un choix et de faciliter le travail de test des différents modes de recherche, nous avons encapsulé la recherche sur GeoNames dans deux classes. L’enchaînement de requêtes que l’on souhaite faire ne dépend alors que de quelques chaînes de un à deux caractères passées en argument aux instances. La figure 2.10 modélise ces deux classes.

La classe `LocationList` est instanciée avec la liste de toponymes extraits du titre, et en supprime les entiers. Les noms potentiels sont alors recherchés séquentiellement sur GeoNames, en implémentant l’algorithme décrit à la section 2.3. En annexe, un diagramme de séquence modélise cet algorithme en réutilisant l’exemple du titre de la photo prise au Japon.

2.6.5 Génération de la carte

La carte interactive est générée par une bibliothèque JavaScript, Leaflet⁸, qui se sert de *map tiles* (littéralement « tuiles de carte ») mis à disposition par un service dont on lui fournit l’adresse. Dans notre cas, nous utilisons les *tiles* de la carte satellite Google. Celles-ci sont des carrés de 256x256 pixels qui permettent de découper la carte en

5. <https://github.com/praw-dev/praw> (Dernière consultation le 6 mai 2020)

6. <https://treetaggerwrapper.readthedocs.io/en/latest/> (Dernière consultation le 6 mai 2020)

7. <https://geocoder.readthedocs.io/index.html> (Dernière consultation le 6 mai 2020)

8. <https://leafletjs.com/> (Dernière consultation le 6 mai 2020)

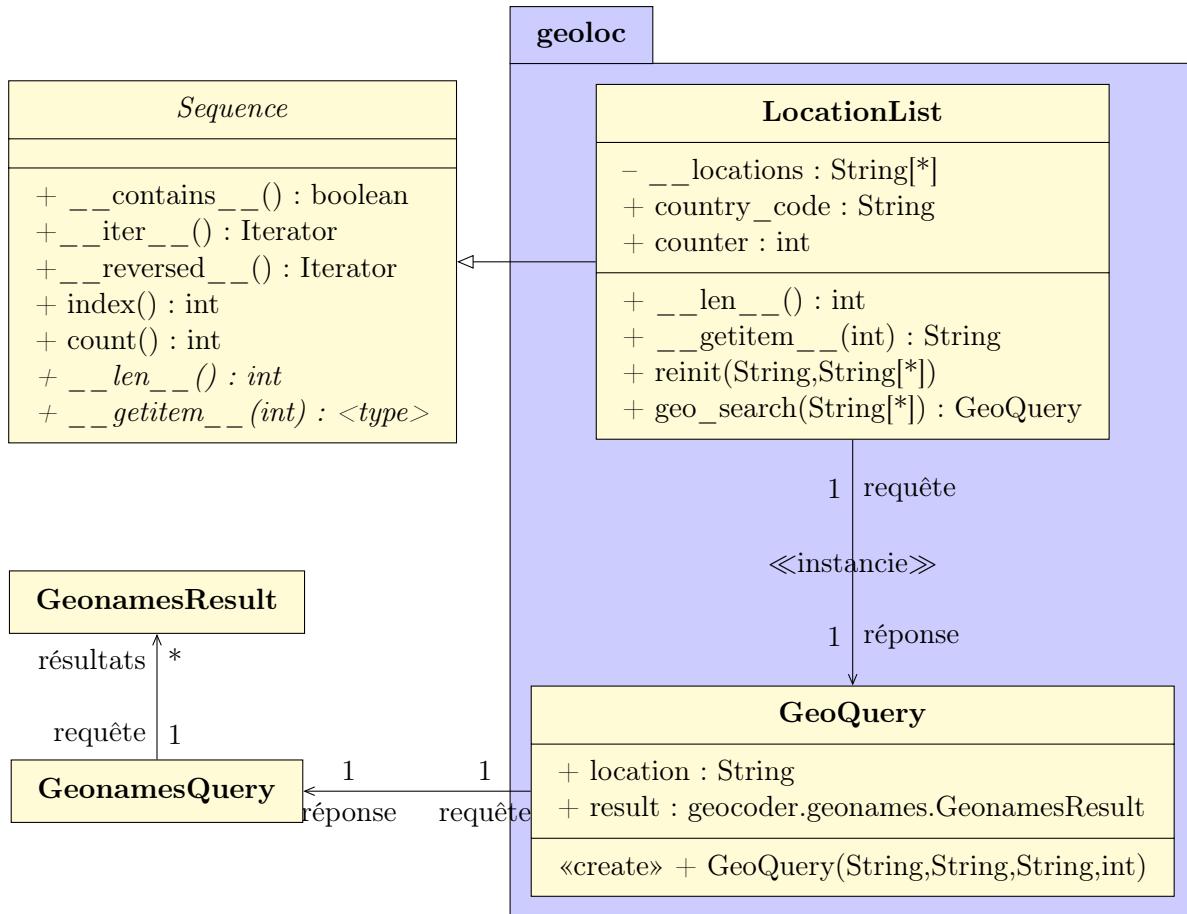


FIGURE 2.10 – Diagramme UML du module Geoloc. `LocationList` hérite de la *abstract base class* `Sequence`, qui fournit une interface pour listes. La méthode `geo_search` lance les recherches sur les toponymes de `__locations` en instanciant des objets de classe `GeoQuery`. En stockant le lieu recherché dans `location`, celle-ci permet à un résultat de requête du module Geocoder de se découpler de `LocationList`.

fonction du niveau de zoom. Ainsi, au fur et à mesure de l'accroissement du zoom, seules les *tiles* nécessaires à la vue courante sont chargées.

Leaflet permet de placer des balises (*markers*) sur la carte, avec des icônes personnalisées. Nous positionnons donc des balises aux coordonnées des images récupérées, et affectons comme icône à chacune la photo correspondant à ses coordonnées.

Un cas qui se présente régulièrement est celui d'une requête renvoyant plusieurs photos d'un même endroit. Les coordonnées étant les mêmes, sans intervention spéciale de notre part les images seraient superposées et seule la dernière serait visible. Nous avons réglé ce problème en utilisant la bibliothèque OverlappingMarkerSpiderfier⁹, qui disperse les photos autour d'un point central au clic de l'utilisateur.

9. <https://github.com/jawj/OverlappingMarkerSpiderfier> (Dernière consultation le 6 mai 2020)

2.6.6 Base de données dans le *cloud*

Comme nous l'avons expliqué à la section 2.5, nous stockons les documents correspondants aux liens Reddit dans une base de données. MongoDB Atlas propose un compte gratuit allouant 512 mégaoctets dans le nuage, ce qui nous permet d'accéder à nos données depuis n'importe quel ordinateur muni d'une connexion internet.

Les documents sont stockés dans une collection, l'équivalent d'une table dans le modèle relationnel. Mais dans une base de données orientée document, l'objet que l'on stocke n'est pas représenté sous forme de tuples répartis entre plusieurs tables, et il est inséré en bloc dans la collection. Un exemple de document est donné à la figure 2.11.

Pour s'assurer qu'un lien EarthPorn ne soit pas stocké plusieurs fois sous la forme de documents différents au même contenu, nous utilisons la fonctionnalité d'indexation de MongoDB. Les champs indexés sont en rouge dans l'exemple. Nous leur imposons une contrainte d'unicité : l'insertion d'un nouveau document dont la conjonction des valeurs des champs indexés n'est pas unique est rejetée.

La communication avec l'API de MongoDB est encore une fois facilitée par un *wrapper* Python, Pymongo¹⁰. Dans le but de s'abstraire un peu plus des détails de connexion, nous avons ajouté une couche supplémentaire avec notre module Mongo. En annexe, un diagramme modélisant les classes de ce module.

```
_id: ObjectId("5eb344848cbafcc4acb43824")
link: "https://www.reddit.com/r/EarthPorn/comments/dnw0kn/lake_chuzenji_..."

```

FIGURE 2.11 – Un document correspondant à un lien EarthPorn. `ObjectId` est un identifiant généré par MongoDB pour le champ `_id`, l'équivalent d'une clé primaire dans le modèle relationnel. Nous aurions pu créer notre propre objet `_id` en y stockant les valeurs des champs en rouge plutôt que de définir un index unique pour ces champs.

10. <https://pypi.org/project/pymongo/> (Dernière consultation le 6 mai 2020)

Partie 3

Analyse des Résultats

3.1 Les cycles de recherche sur GeoNames

Pour faire correspondre un lieu potentiel extrait du titre d'un lien Reddit avec un toponyme existant, Geoscape est entièrement dépendant de GeoNames. Nous avons expliqué dans la partie 2 les particularités du moteur de recherche de GeoNames, et le choix d'utiliser les fonctionnalités de recherche avancée pour pallier ces tendances.

Nous avons donc essayé différentes procédures de recherche, et les avons testé ; les résultats sont présentés à la figure 3.1. Les tests ont été effectués par un script parcourant 175 lieux extraits de titres et confirmés comme étant bons, en leur appliquant à chacun le cycle de recherche choisi. Le script donne 1 point pour un bon résultat en sortie de cycle (un bon résultat pouvant être vide, si le lieu n'est pas présent dans la base de données GeoNames), 0.5 pour un résultat « partiel » (l'aéroport plutôt que la ville, le chef-lieu plutôt que la province, la montagne voisine,...), et 0 point pour un mauvais résultat.

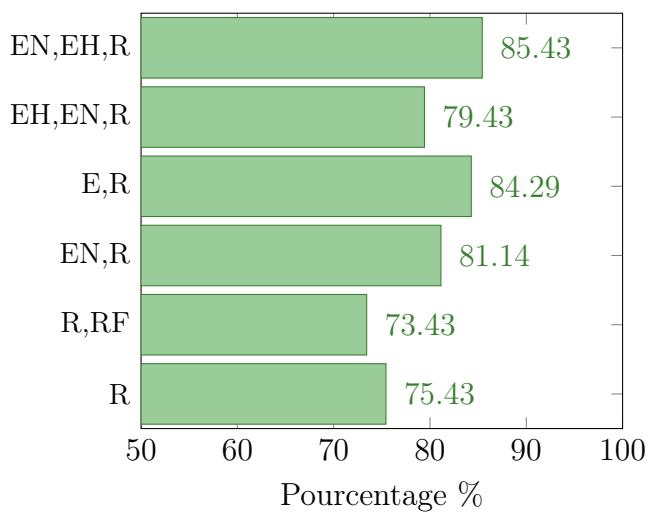


FIGURE 3.1 – Efficacité de différentes procédures de recherche sur GeoNames, d'après nos tests. **R** : recherche standard ; **RF** : recherche *fuzzy* ; **E** : recherche exacte ; **EN** : recherche exacte sur les ensembles naturels (montagnes,cours d'eau,...) ; **EH** : recherche exacte sur les ensembles humains (villes, provinces,...). Le pourcentage = $\sum \text{points} / 175 * 100$.

Nous devons ici justifier l'apparente inefficacité de nos procédures : au lieu de lancer plusieurs requêtes distinctes sur un même lieu, pourquoi ne pas lancer une requête standard, et ensuite parcourir les résultats en cherchant d'abord une correspondance exacte (ce qui équivaudrait aux cycles commençant par EX) ?

La réponse est double : d'une part, comme le montrent les exemples dans la table 2.3, le résultat exact peut se situer à un rang très éloigné du premier résultat, ce qui impliquerait de transférer le maximum de résultats autorisés par l'API (1000) à chaque requête. D'autre part, autre particularité de GeoNames, la recherche exacte ne l'est pas réellement : les noms « alternatifs » d'un lieu sont pris en compte, même s'ils ne correspondent pas exactement, et la recherche s'en retrouve améliorée.

Les résultats font apparaître que la recherche *fuzzy* est contre-productive : dans le cas de lieux qui ne sont pas sur GeoNames, un bon résultat est une réponse vide de la base de données, mais la recherche élargie trouve souvent un toponyme à l'orthographe similaire et le lieu associé est donné en réponse, même s'il est incorrect. Les cas où elle permet réellement de trouver un lieu dont le nom était mal orthographié dans le titre EarthPorn ne sont pas assez nombreux pour compenser ces erreurs.

Les deux cycles de recherche qui se détachent des autres sont EN,EH,R et E,R, avec respectivement 85.43 et 84.29% d'efficacité. En pratique, l'implémentation du premier cycle n'est pas plus coûteuse en nombre de requêtes : EN,EH se fait en une requête exacte, avec un parcours des au plus 10 résultats renvoyés jusqu'à trouver un lieu appartenant à EN, et le premier résultat sinon. Nous avons donc choisi ce cycle de recherche pour notre algorithme de recherche sur GeoNames.

3.2 Amélioration de Geoscape

3.2.1 Analyse des photos signalées

Nous avons expliqué comment l'utilisateur pouvait signaler les photos bien ou mal placées sur la carte. Ces dernières doivent être analysées : où se situe l'erreur, et comment y remédier ? Pour cela, nous avons mis en place un système de test permettant à des testeurs de les étudier.

Les testeurs doivent s'inscrire sur notre site pour accéder à la page qui leur est dédiée. L'inscription nous permet de les distinguer, et ainsi de répartir aléatoirement les liens EarthPorn à analyser. Soit T l'ensemble des testeurs, avec $|T| \geq 1$, alors le nombre de testeurs par lien est calculé par la fonction

$$nb_testeurs(T) = \begin{cases} \min(|T|, 9) & \text{si } |T| \text{ impair,} \\ \min(|T| - 1, 9) & \text{sinon.} \end{cases}$$

Le nombre impair de testeurs est justifié par notre protocole de test, qui est simple : les tests consistent à faire des choix binaires sur les mots du titre, et les résultats

définitifs d'un test sont déterminés par une sélection des choix majoritaires des testeurs. Ainsi, avec un nombre impair de testeurs nous pouvons faire émerger une majorité dans tous les cas de figure.

Augmenter leur nombre devrait augmenter la fiabilité des résultats, mais nous considérons qu'un maximum de neuf testeurs (nombre arbitraire) par lien suffit, et qu'au-delà le gain supposé en fiabilité ne compense pas le temps d'attente croissant avant l'obtention de tous les résultats. Car les testeurs sont après tout des volontaires qui réalisent leurs tests quand ils ont envie, sans contrainte de calendrier.

Lorsqu'un testeur se connecte et accède à la page de tests, dont l'interface est visible à la figure 3.2, une requête est automatiquement lancée pour compter le nombre de liens EarthPorn qu'il doit traiter et le lui signaler.



FIGURE 3.2 – L'entrée dans la zone de tests.

Le principal test consiste à sélectionner un ou plusieurs mots du titre désignant le lieu où la photo a été prise, ou potentiellement les lieux, si le titre en contient plusieurs et que le testeur n'en privilégie aucun. Ces lieux sont ensuite recherchés sur GeoNames, et jusqu'à trois résultats sont affichés pour chacun d'entre eux. Dans ce second test, l'utilisateur désigne alors le résultat de GeoNames qui correspond le mieux au(x) lieu(x) qu'il a identifié. Ceci nous permet en particulier de savoir si l'erreur de positionnement est due à notre algorithme de recherche et, *in fine*, à une erreur d'étiquetage de TreeTagger, ou si le lieu n'est pas présent sur GeoNames.

Mais une possibilité plus grave est qu'il n'y ait tout simplement pas d'indication géographique dans le titre du lien EarthPorn (nous avons expliqué à la section 2.3 que dans les cas où la recherche GeoNames est infructueuse, nous utilisons les coordonnées du pays pour l'affichage). Un troisième test traite de ce cas : l'utilisateur indique si le lieu est

présent dans le titre. Si sa réponse est négative, nous savons qu'il n'y a rien à faire. Nous ne pourrons jamais correctement positionner la photo, puisque tout le fonctionnement de Geoscape se base sur l'analyse du titre de l'image.

3.2.2 Traitement des résultats

Lorsque tous les testeurs assigné à l'examen d'un lien ont réalisé leurs tests, un résultat final est établi à partir des résultats individuels et stocké dans une collection de la base de données. L'analyse de ce document par un script de traitement peut conduire à la création d'une nouvelle règle pour l'identification des toponymes dans un titre de lien EarthPorn.

Notre traitement se base sur la comparaison des différentes listes que nous avons construites jusqu'ici, comme l'illustre la table 3.1 : la liste d'étiquettes de TreeTagger, la liste de mots sélectionné par notre algorithme, qui se base sur le regroupement de noms propres voisins (*cf* section 2.2.2), et la liste des mots sélectionnés comme étant des lieux par les testeurs.

Titre	...	zoom	in	for	details	.	Lago	di	Carezza	,	Italy
Tags	...	VVB	AVP	PRP	NN2	SENT	NP0	NP0	NN1	PUN	CTY
Algo	...	0	0	0	0	0	Lago	di	0	0	0
Test	...	False	False	False	False	False	True	True	True	False	False

TABLE 3.1 – Comparaison de listes. Le lieu sélectionné par le testeur est « Lago di Carezza », mais « Carezza » n'a pas été étiqueté comme étant un nom propre, et donc le toponyme qui a été extrait du titre se réduit à « Lago di ». Une anomalie est détectée dans la colonne en rouge : la valeur booléenne de 0 n'est pas égale à True.

A partir du mot ou de la suite de mots correspondant à une anomalie tel qu'elle est définie dans la table 3.1, nous opérons toujours par le principe du voisinage : nous considérons les mots voisins sélectionnés par le testeur. Cela donne la règle de la figure 3.3.

```
country: Italy,
expr: Lago di,
pos: [NPO, NPO],
take: 1
```

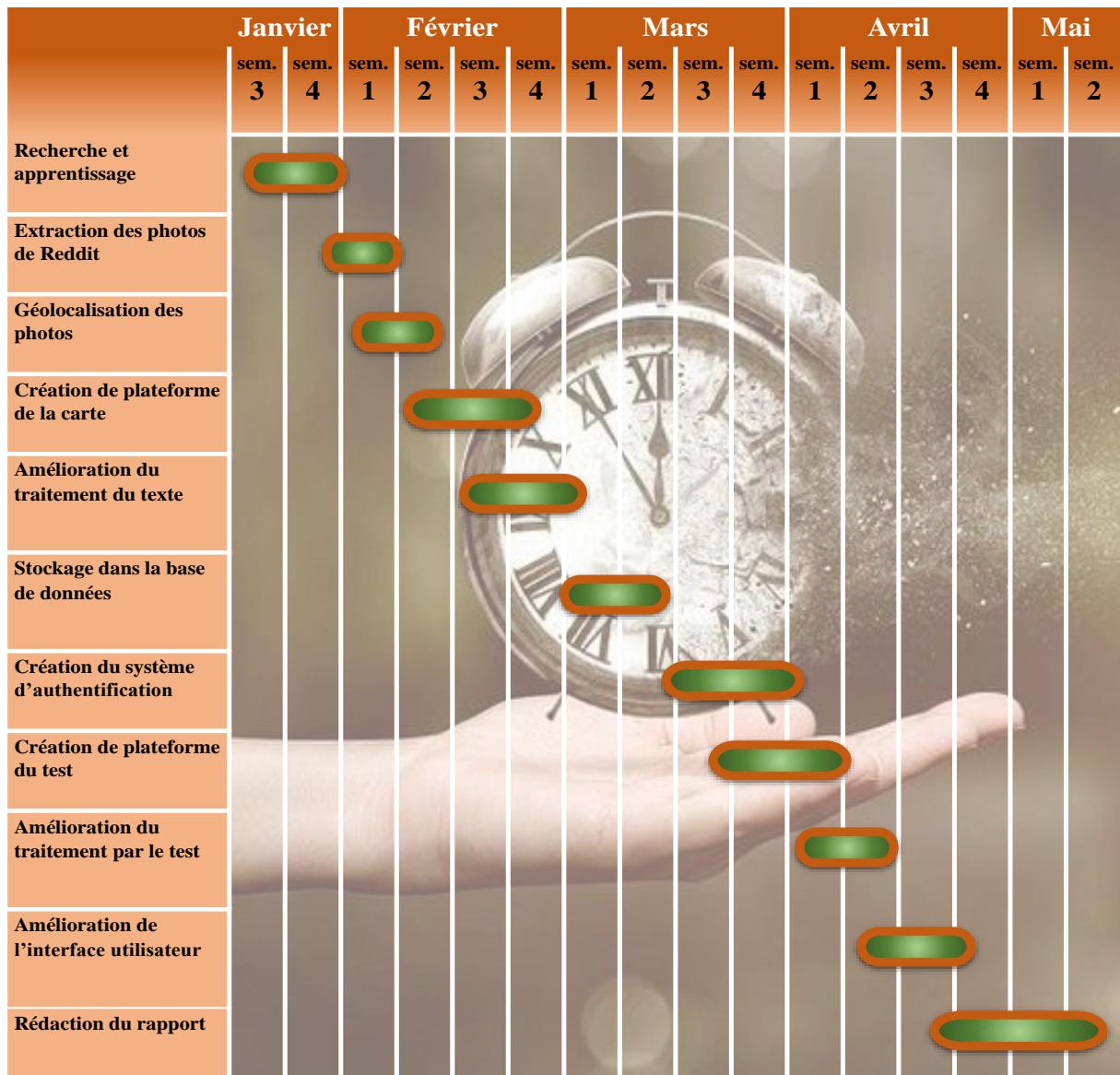
FIGURE 3.3 – La règle créée à partir de l'exemple précédent. L'expression « Lago di » rencontré dans un titre lors d'un *scraping* par notre algorithme déclenchera une capture du mot à sa droite. Le pays dans cette règle est un moyen simplifié d'identifier la langue.

Le script crée de la même façon des règles qui capturent à gauche ou des deux côtés. Si un mot a été sélectionné en trop par notre algorithme, le traitement génère une règle de suppression. Ainsi nous avons mis en place un système assez simple mais automatisé de création de règles.

Partie 4

Gestion du projet

4.1 Diagramme de Gantt



4.2 Organisation du travail

Nous avons réparti notre travail selon la division client-serveur de l'application : d'un côté les modules Python, de l'autre les pages web et la programmation dynamique en JavaScript. Nous avons commencé par une « exploration » des différents aspects du projet, en partant du *scraping* et en cheminant jusqu'à l'affichage, ce qui nous a permis de réaliser le cœur de Geoscape. Nous avons ensuite mis en place la base de données, avant de nous lancer dans l'implémentation des signalements de l'utilisateur et du système de test qui en découle.

Pour nous organiser dans le temps, nous avons utilisé l'application Trello¹, un outil de gestion de projet en ligne très visuel et simple d'utilisation. La rédaction du rapport s'est faite sur le site Overleaf², qui permet de rédiger de manière collaborative des documents L^AT_EX.

4.3 Réunion avec l'encadrant

Tout au long de ce projet, nous étions guidés par notre encadrant. Des réunions régulières avec M. Poncelet ont eu lieu afin d'assurer le suivi de notre avancement. Lors de la période de confinement suite à l'épidémie de Covid-19, les réunions se sont déplacées sur l'application Discord³. À la fin de chaque rencontre, nous établissions un compte-rendu résumant les points abordés, les décisions prises ainsi que la date prévue pour la réunion suivante.

-
1. <https://trello.com/> (Dernière consultation le 6 mai 2020)
 2. <https://www.overleaf.com/> (Dernière consultation le 6 mai 2020)
 3. <https://discordapp.com/>

Conclusion

Dans ce rapport nous avons présenté Geoscape, une application web qui permet d'afficher des photos de magnifiques paysages naturels sur une carte interactive. Le dépôt GitHub du projet est à l'adresse <https://github.com/MarwanMashra/Reddit/>.

Ce projet s'est caractérisé par la diversité des langages, des bibliothèques et des API : d'une part la programmation web, faite d'événements et de requêtes asynchrones, avec le JavaScript, sa fameuse bibliothèque JQuery et Leaflet pour créer notre carte ; d'autre part, le langage Python et les nombreux *wrappers* qu'il propose pour les sites avec lesquels nous devions communiquer.

La boucle centrale de Geoscape, de la requête envoyée par l'utilisateur jusqu'à l'affichage des images sur la carte générée pour lui, fonctionne parfaitement, et réalise notre objectif de fusion de données. Cela nous a beaucoup appris sur la programmation web, quasiment pas abordée dans nos études jusqu'ici. De plus, avoir dû combiner différents langages et interfaçer avec différentes API a développé notre adaptabilité, une compétence clé pour tout informaticien.

La deuxième partie du projet, le cycle d'amélioration de Geoscape par les tests, a également été implémentée, mais n'est pas aussi aboutie. Nous avons été tellement plongée dans l'implémentation que nous n'avons jamais vraiment pu réaliser d'importantes séries de tests et en dégager des résultats significatifs. C'est aussi le domaine où la programmation ne suffit plus, et des ressources plus théoriques en traitement du langage sont nécessaires. Nous en sommes donc resté à des tests relativement sommaires, avec création de règles simples basées sur le voisinage.

En conséquence, si on peut bien sûr continuer à améliorer l'interface utilisateur, peaufiner l'esthétique et proposer plus d'options, c'est surtout après un travail approfondi de récolte et d'analyse des résultats de l'application que Geoscape pourra proposer un service de qualité. Cela nous fait prendre conscience du rôle majeur joué par la recherche d'information et le traitement du langage dans le développement d'applications et de services web performants.

Le développement de cette application a été un processus particulièrement enrichissant pour nous. Pour le mener à bien nous avons mis à profit les compétences acquises au cours de nos deux années d'études d'informatique. Nos cours de programmation (HLIN202, HLIN302, HLIN303), de modélisation (HLIN406) et de systèmes d'information (HLIN304) ont été particulièrement utiles. Utiles aussi les conseils avisés et les encouragements de notre encadrant, M. Poncelet, que nous remercions ici.

Bibliographie

SCHMID, HELMUT, (1994). Probabilistic Part-of-Speech Tagging Using Decision Trees. In *Proceedings of the International Conference on New Methods in Language Processing*.

<https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/tree-tagger1.pdf>
Dernière consultation le 6 mai 2020.

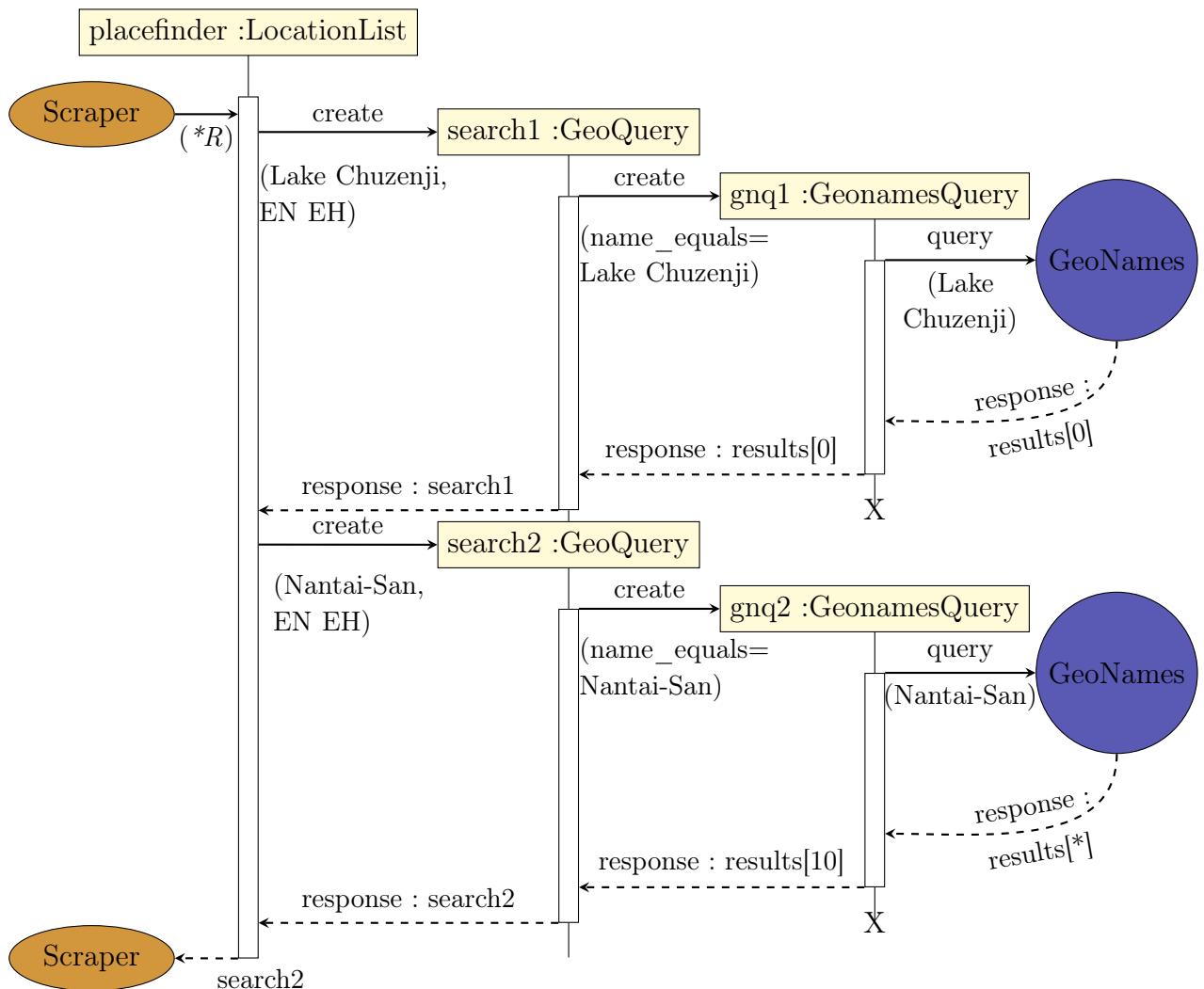
CHERNIAK, EUGENE, (1997). Statistical Techniques for Natural Language Parsing. In *AI Magazine*. cs.brown.edu/people/echarnia/papers/aimag97.ps
Dernière consultation le 6 mai 2020.

JURAFSKY, DANIEL & MARTIN, JAMES H., (2019). Speech and Language Processing, 3rd Edition Draft. <https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf>
Dernière consultation le 6 mai 2020.

Annexes

Annexe A

Diagramme de séquence de la recherche sur GeoNames. $*R$ est l'ensemble des types de recherche à effectuer. Le premier élément de la liste de `placefinder` est passé au constructeur de `search1`, qui le stocke dans son attribut `location` puis le passe au constructeur de `gnq1`. Celui-ci effectue la recherche sur GeoNames, mais la recherche est infructueuse et une liste vide est renvoyée. `search1` est donc construit avec l'attribut `result` vide, et `placefinder` déclenche alors la construction d'une nouvelle instance avec le deuxième élément de la liste. Un résultat est trouvé, et `search2` est retourné au `scraper`.



Annexe B

Modélisation UML du module Mongo. La classe Mongo contient des méthodes statiques effectuant des vérifications sur les collections de notre base de données. Trois classes héritent de cette super classe : MongoSave pour l'insertion de documents, MongoLoad pour la récupération, et MongoUpd pour les mises à jour. Les types Document, Query, Projection et Update sont des dictionnaires.

