



Institut National des Sciences Appliquées et de Technologie

UNIVERSITÉ DE CARTHAGE

Projet de Fin d'Études

Filière : **GL**

Développement d'une plateforme d'extraction de données automatisée pour l'alimentation d'un moteur de recherche spécialisé

Présenté par

Oussama TRABELSI

Encadrant INSAT : **Mme SFAXI Lilia**
Encadrant ENTREPRISE : **Mr BALMISSE Gilles**

Présenté le : **26/06/2019**

JURY

Mme. President Sarra BEN LAGHA (Président)
Mme. Rapporteur Sana HAMDI (Rapporteur)

Année Universitaire : 2018/2019

Dédicaces

Je dédie ce travail à

Mes parents qui ont beaucoup sacrifié pendant de longues années pour que je puisse être où je suis aujourd’hui.

Ma soeur et mon frère, qui ont toujours montré leur fierté et leur support.

Mes proches, qui m’ont toujours encouragé et souhaité du succès.

Mes amis, qui étaient toujours présents pour moi avec le support moral et l’encouragement desquels j’avais besoin.

A tous ceux qui m’ont aidé à évoluer sur un plan académique ou personnel. Je vous remercie !

Remerciements

Qu'**Allah** soit loué pour nous avoir guidé et donné la persévérance et la patience qui nous ont permis de réaliser ce travail.

Nous tenons à remercier tous les membres de **l'équipe Djiant** pour leurs efforts et leur contribution dans le bon déroulement de ce stage.

Nous remercions en particulier M. Gilles BALMISSE, M. Ma'amar BOUHERAOUA, M. Adam GORRAB et Mlle Nawres MAHFOUDH pour leur participation dans l'évaluation de ce travail et son évolution.

Nous tenons en plus à remercier tous **les enseignants de l'INSAT**. C'est grâce à leurs efforts que nous avons la formation nécessaire qui nous a permis d'accomplir ce travail.

Nous remercions en particulier **Mme Lilia SFAXI**, non seulement pour son engagement, sa disponibilité, ses conseils techniques et son esprit ouvert, mais surtout pour le soutien moral qu'elle nous a apporté tout au long de ce stage, et pour nous avoir encouragé à persister et à nous surpasser.

Enfin, merci à tous **les membres du jury** pour avoir accepté d'évaluer ce travail et de consacrer de leur temps pour nous aider à apprendre de cette expérience.

Table des Matières

Liste des Figures	vi
I Cadre Général Du Projet	3
1 Présentation de l'organisme d'accueil	3
2 Présentation du Projet	4
2.1 Idée globale du projet	4
2.2 Problématique	4
2.3 Solution Proposée	5
3 Méthodologie de travail	6
3.1 La méthodologie Kanban	6
3.2 Diagramme de Gantt	7
4 Cycle de développement	7
II Étude des besoins et Architecture	9
1 Identification des acteurs	9
2 Expression des besoins	10
2.1 Besoins fonctionnels	10
2.1.1 Data Factory	10
2.1.2 Matrix	11
2.1.3 Explorer	12
2.1.4 Crawler	12
2.1.5 Keyword Extractor	13
2.2 Besoins non fonctionnels	13
3 Architecture	15
III Première Release - Multi-Moteurs	17
1 Data Factory	18
1.1 Conception	18
1.2 Réalisation	21
2 Matrix	21
2.1 Conception	21
2.2 Technologies utilisées	24
2.3 Réalisation	26

3	Evaluation et perspectives	27
IV Deuxième Release – Crawling		28
1	Crawler	29
1.1	Itération 1 : Modèle de données et traitement asynchrone	29
1.1.1	Objectif	29
1.1.2	Conception	30
1.1.3	Choix technologiques	31
1.2	Itération 2 : Traitement synchrone et optimisation	32
1.2.1	Objectif	32
1.2.2	Architecture	32
1.2.3	Conception	33
1.2.4	Choix technologiques	36
1.3	Itération 3 : Monitoring et persistance des données	36
1.3.1	Objectif	36
1.3.2	Description	37
1.3.3	Choix technologiques	37
1.3.4	Réalisation	38
2	Keyword Extractor	40
2.0.1	Choix technologique : Spark	40
3	Evaluation et perspectives	41
V Troisième Release - Recherche, Approche déclarative et améliorations graphiques		42
1	Explorer	43
1.1	Iteration 1 : Interface de recherche	43
1.1.1	Objectif	43
1.1.2	Conception	43
1.1.3	Réalisation	45
1.2	Iteration 2 : Améliorations graphiques	48
1.2.1	Objectif	48
1.2.2	Réalisation	48
2	Data Factory	49
2.1	Iteration 2 : Synchronisation des données	49
2.1.1	Objectif	49

Table des Matières

2.1.2	Conception	49
2.1.3	Réalisation	50
2.2	Iteration 3 : Approche déclarative	51
2.2.1	Objectif	51
2.2.2	Conception	51
2.2.3	Réalisation	52
3	Insertion de données supplémentaires	54
3.1	Description	54
4	Evaluation et perspectives	55
Conclusion Générale et Perspectives		56
Bibliographie		57

Liste des Figures

I.1	Visualisation des tâches sur meistertask	6
I.2	Tâches Effectuées	7
I.3	Cycle de développement	7
II.1	Data Factory : diagramme de cas d'utilisation	10
II.2	Matrix : diagramme de cas d'utilisation	11
II.3	Explorer : diagramme de cas d'utilisation	12
II.4	Crawler : diagramme de cas d'utilisation	12
II.5	Keyword Extractor : diagramme de cas d'utilisation	13
II.6	Architecture	15
III.1	Périmètre du premier release	18
III.2	Première approche : une base de données par moteur	19
III.3	Deuxième approche : une base de données centrale	20
III.4	Data Factory : choix de moteur	21
III.5	Matrix : Région en tant que modèle séparé	22
III.6	Matrix : Région en tant qu'attribut	23
III.7	Matrix Web API	24
III.8	Écran Matrix	26
IV.1	Périmètre du deuxième release	29
IV.2	Crawler itération 1 : Diagramme de classe	30
IV.3	Crawler itération 2 : Architecture détaillée	33
IV.4	Diagramme de classe du "Master"	33
IV.5	Diagramme de classe du "Worker"	34
IV.6	Diagramme de séquence de l'opération de crawling	35
IV.7	Interface Kibana	39
V.1	Périmètre du troisième release	43
V.2	Diagramme de séquence d'une recherche	44
V.3	Explorer : choix de moteur	45
V.4	Explorer : choix du type d'objet	46
V.5	Explorer : Taper la requête	46
V.6	Explorer : Le mode d'affichage regroupé	47

Liste des Figures

V.7 Explorer : Le mode d'affichage mixte	47
V.8 Carte de la liste des moteurs	48
V.9 Diagramme de séquence de la synchronisation des données	49
V.10 Data Factory : Menu	50
V.11 Modèle de données	51
V.12 Ajouter un contact	52
V.13 Envoi du mail de l'interface de Data Factory	53
V.14 Vérification de l'envoi du mail	53
V.15 Vérification du contenu de test du mail	53
V.16 Interface du formulaire à remplir	54

Résumé

De nos jours, le Web présente un sujet très controversé.

D'une part, il est riche en informations sur toute sorte de sujet, grâce à la facilité de partage de données.

D'autre part, cette facilité de création de données rend parfois difficile la recherche d'informations fiables puisqu'il n'existe quasiment aucune forme de contrôle sur ce qui peut être publié et ce qui doit être supprimé ou du moins masqué.

Cette situation fait du Web un outil très utile pour les recherches à des fins générales. Toutefois, les spécialistes pourraient avoir trop de difficulté à trouver des informations suffisamment riches pour répondre à leurs besoins, parcequ'elles seraient dispersées sur Internet et parfois mal classées par les moteurs de recherche vu que la plupart des gens préfèrent accéder à des sites Web offrant des explications basiques sans rentrer dans des détails compliqués.

C'est pourquoi chez Djiant, nous avons pensé à organiser le Web en utilisant différents moteurs de recherche spécialisés où chaque moteur de recherche n'est concerné que par un domaine pour chaque région du monde. Cela offre des résultats plus précis puisque la recherche ne sera pas effectuée sur tout le Web, mais uniquement sur des données liées au domaine qui nous intéresse.

Djiant souhaite pouvoir indexer les données sous une forme structurée, ce qui signifie que les pages Web ne seront pas toutes mélangées et indexées uniquement par leur contenu, mais qu'elles seront associées à certains types de données telles que les entreprises, les formations, les offres d'emploi, etc... Cela permet aux utilisateurs d'afficher les résultats du moteur de recherche non seulement comme un ensemble de pages Web, mais également comme un ensemble d'objets de différents types pour lesquels les données seront extraites des pages Web auxquelles ils sont associés. Dans le cadre de ce projet, il nous a été initialement demandé de gérer la partie extraction de données afin de créer les différents types d'objets. Toutefois, en raison de certaines circonstances, la société a décidé de se concentrer sur un seul type d'objet, à savoir le type d'objet "Entreprise", et de terminer la chaîne de bout en bout en commençant par l'extraction des données jusqu'au module de recherche utilisé par les utilisateurs finaux.

Pour atteindre cet objectif, nous avons utilisé cinq modules distincts. Trois étant des applications Web permettant de la collecte de données, la recherche et la gestion des listes de moteurs. Les deux autres étant des services qui servent à collecter des pages Web, les stocker et y extraire automatiquement des mots-clés afin de mieux indexer les objets qui leur sont associés.

Abstract

Nowadays, the web presents a very controversial subject. On one hand, it's rich in information about every subject imaginable and that's thanks to the ease of sharing data.

On the other hand, this ease of data creation makes it sometimes hard to find reliable and usable information since there is almost no form of control on what is to be published and what should be removed or at least hidden.

This situation makes the web a very useful tool for general purpose searches, however, this may make it too difficult for specialists to find information that is rich enough for their needs because it would be scattered all over the internet and sometimes not very well ranked by search engines since most people prefer to access websites offering basic explanations without any complicated details.

This is why in Djiant, We thought about organizing the web using different specialized search engines where each search engine is concerned only by one domain for a certain region. This offers more precise query results due to the fact that the search won't be done across all the web, but only on data that is already relevant to the domain we are interested in.

Djiant aims to being able to index data in a structured form, meaning that web pages won't be all mixed and indexed by their contents only but they will be associated to certain data types such as companies, training courses, job offers, etc ... This allows the users to view the search engine results not only as a set of web pages, but as a set of objects of different types for which the data will be extracted from the web pages with which they are associated. Through this project, we were initially asked to handle the data extraction part in order to create the different object types. But, due to certain circumstances the company decided to focus on only one object type, which is the object type "Enterprise", and instead finish the whole process cycle starting with data extraction up until the search module that the final users will be using.

To achieve this goal, we used 5 separate modules where 3 are services in the form of web applications to handle data collection, search, and handling the engines lists, and the other 2 are services written in python to crawl web pages and automatically extract keywords from them in order to better index the objects associated to them.

Introduction Générale

Quand on parle des moteurs de recherche, on pense généralement aux entreprises géantes qui dominent le domaine et principalement Google, Bing et plus récemment DuckDuckGo. Malheureusement, les solutions existantes sont toutes basées sur la même idée avec des différences purement techniques, c'est l'idée d'avoir un moteur de recherche généraliste qui indexe le web, ou au moins une partie du web. L'entreprise Djiant vise à travers son projet à introduire l'idée de grille de moteurs de recherche spécialisés, c'est à dire, fournir un réseau mondial de moteurs de recherches où chaque moteur se spécialise dans une thématique spécifique dans une région bien déterminée. Évidemment, ce projet ne vise pas à remplacer les moteurs de recherche généralistes existants, mais plutôt à offrir une deuxième alternative qui cible les professionnels de chaque domaine et leur offre des recherches plus pertinentes et plus riches qu'une recherche globale sur tout le web, dont le contenu est généralement un mixte de données fiables et utiles et d'autres de sources inconnues ou peu crédibles.

Dans le but de contrôler la qualité des données, l'entreprise Djiant a proposé d'offrir une solution qui permet de collecter les informations à indexer dans les moteurs grâce à des indexeurs humains qui vont être assistés par l'ordinateur pour plus de productivité et de contrôle sur les sources des données. C'est dans ce cadre que notre projet ait lieu, afin d'améliorer l'opération de collecte de données ainsi que de terminer la chaîne de production de bout en bout : c'est à dire, dès l'extraction des données jusqu'à leur mise en production pour les recherches.

Dans le cadre de notre projet, le travail réalisé ne présente pas la totalité du produit, mais une preuve de concept qui présente, certes, la totalité du processus et ses différentes fonctionnalités, mais avec des limites au niveau des types d'objets à indexer. Dans notre travail nous allons nous limiter aux objets de type "entreprise".

Le présent rapport est formé de 5 chapitres qui détaillent le travail effectué lors de ce stage sur un niveau conceptuel ainsi qu'un niveau technique. Nous commençons avec un premier chapitre qui présente le contexte du projet, l'organisme d'accueil, la méthodologie de travail et le cycle de développement adopté. Nous passerons ensuite au deuxième chapitre où nous présentons l'architecture globale du projet ainsi que les besoins fonctionnels et non fonctionnels de ses différents modules. Les 3 chapitres suivants, seront consacrés aux différents releases du produit Djiant. Le troisième chapitre va donc énumérer les différentes tâches effectuées afin d'atteindre les buts de la première release qui vise à améliorer la solution existante très basique afin de pouvoir passer d'un modèle à un seul moteur de recherche spécialisé, à un produit

qui supporte toute une grille de moteurs. Le quatrième chapitre est consacré à la deuxième release au niveau de laquelle nous avons travaillé sur un module de “web crawling” qui permet de récupérer les pages web des sites des entreprises afin de les utiliser , éventuellement, dans l’extraction de mots-clés qui serviront à l’indexation de ces entreprises dans nos moteurs. Pour le cinquième et dernier chapitre, nous allons évoquer la partie recherche et mise en production des données collectées ainsi que quelques tâches de finalisation au niveau du module d’extraction de données.

Chapitre I

Cadre Général Du Projet

Plan

1	Présentation de l'organisme d'accueil	3
2	Présentation du Projet	4
2.1	Idée globale du projet	4
2.2	Problématique	4
2.3	Solution Proposée	5
3	Méthodologie de travail	6
3.1	La méthodologie Kanban	6
3.2	Diagramme de Gantt	7
4	Cycle de développement	7

Introduction

Dans ce chapitre nous présentons le cadre général du projet en commençant avec une présentation de l'organisme d'accueil puis la problématique posée ainsi que notre contribution dans la solution. Puis, nous allons passer à la présentation de la méthodologie adoptée ainsi que les tâches effectuées lors de ce stage sous forme de diagramme de Gantt. Et enfin, nous terminerons avec le cycle de développement que nous allons adopter lors de ce stage.

1 Présentation de l'organisme d'accueil

Djiant Africa & ME, est une succursale de Djiant qui est une société ambitieuse récemment formée par une équipe de 4 personnes en 2018. Malgré sa naissance récente, Djiant vise à être active sur un niveau international en planifiant la mise en place de succursales dans plusieurs pays. Mais pour le moment, ses efforts sont focalisés sur la mise en place de deux locaux :

- À Tunis, l'entreprise cherche à installer une équipe de développement informatique qui va être responsable de la création et la maintenance du produit.

- À soussse, l'entreprise cherche à installer un centre d'indexation qui serait responsable de la collecte et la validation des données.

2 Présentation du Projet

2.1 Idée globale du projet

Le projet Djiant vise à créer une grille de moteurs de recherche spécialisés qui permettent de faire des recherches dans des domaines spécifiques.

Ces moteurs de recherche vont contenir dans un premier temps une liste limitée de types d'objets telle que le type “Entreprise”.

De plus, ces moteurs vont être regroupés à la fois par région et par ce que nous allons appeler un “réseau” : par exemple le moteur IT-France est associé à la région “France” et fait partie du réseau des moteurs du IT.

Les réseaux sont eux-mêmes regroupés par des thématiques globales que nous allons appeler “domaines” : par exemple le réseau IT auquel appartient le moteur IT-France, fait partie du domaine Digital. La partie la plus critique pour le succès de ce projet sont les données, et c'est dans ce cadre que nous intervenons au niveau de la collecte de ces données à travers un module central “Data Factory” qui va être complémenté par plusieurs autres modules que nous allons préciser plus tard.

2.2 Problématique

De nos jours, les systèmes de tout organisme sont digitalisés afin d'améliorer la performance des employés et les assister à effectuer leurs tâches journalières. Malheureusement, pour le domaine d'indexation et collecte de données , en particulier la collecte des données des entreprises, on remarque un manque de services qui assistent les indexeurs à collecter ces données de façon automatique ou au moins semi-automatique.

C'est pour cette raison qu'une grande partie des services d'indexation font appel à des entreprises qui collectent , de façon complètement manuelle, des données en utilisant des moyens non conçus pour un tel besoin comme l'outil Microsoft Excel par exemple.

En effet, cette façon de faire n'est pas seulement peu productive, mais elle présente également un grand risque d'erreur lors de la saisie manuelle de centaines si ce n'est pas des milliers de fiches de données.

Dans ce cadre, l'entreprise Djiant présente une première version d'un service “Data Factory”

qui permet d’assister les indexeurs à extraire les données des entreprises afin de pouvoir, éventuellement, les exploiter pour des recherches effectuées par le public. Cependant, cette première version est très basique par rapport aux ambitions de l’entreprise et présente plusieurs limites :

- Les données collectées sont regroupées dans une même base sans avoir un découpage logique ou physique qui les organise par moteur.
- Les données collectées ne sont pas exploitables par le public.
- La collecte des données est faite de façon quasi manuelle avec une saisie manuelle de données à partir des sites web des entreprises.
- Une interface graphique basique qui manque les règles d’ergonomie les plus simples.

2.3 Solution Proposée

Afin de permettre à l’entreprise d’atteindre le potentiel qu’elle cherche avec cette solution, nous nous proposons dans ce stage d’améliorer la solution Data Factory dans sa version actuelle par :

- L’ajout de nouvelles fonctionnalités qui assurent de nouvelles méthodes de collecte de données.
- L’amélioration des fonctionnalités de saisie existantes en remplaçant les champs de saisie de base par des menus qui permettent d’interagir directement avec le site de l’entreprise et de sélectionner les données à insérer de façon plus simple et plus sûre.
- L’ajout de modules complémentaires qui couvrent le reste des besoins de l’entreprise telle que :
 - un module de recherche qui permet d’exposer les données collectées au public
 - un module de gestion de moteurs qui permet d’ajouter la possibilité de gérer les groupements possibles des données
 - un module d’extraction des mots-clés de façon automatique à partir des pages web des entreprises

Dans le reste de ce rapport, nous détaillons notre contribution dans chacune de ces tâches.

3 Méthodologie de travail

3.1 La méthodologie Kanban

La méthodologie Kanban se base sur l'approche *Lean* qui se base sur l'idée d'améliorer les processus de production afin de minimiser le gaspillage. Kanban partage cette même vision et l'assure à travers deux principes :

- Une visualisation très rapide des tâches à travers un flux constant : afin d'assurer cet aspect, nous avons utilisé l'outil meistertask[1] afin de visualiser la liste des tâches comme démontré par la figure I.1
- Assurer un produit de bonne qualité à moindre cout en matière de temps et de ressources : afin d'assurer cet aspect, Nous avons appliqué un concept de Kanban appelé TAF, qui consiste à limiter le travail à faire en cherchant un équilibre entre la compétence de l'équipe et les demandes des clients.

De plus, c'est la responsabilité des membres de l'équipe de se documenter et de définir les limites de TAF qu'elle peut supporter.

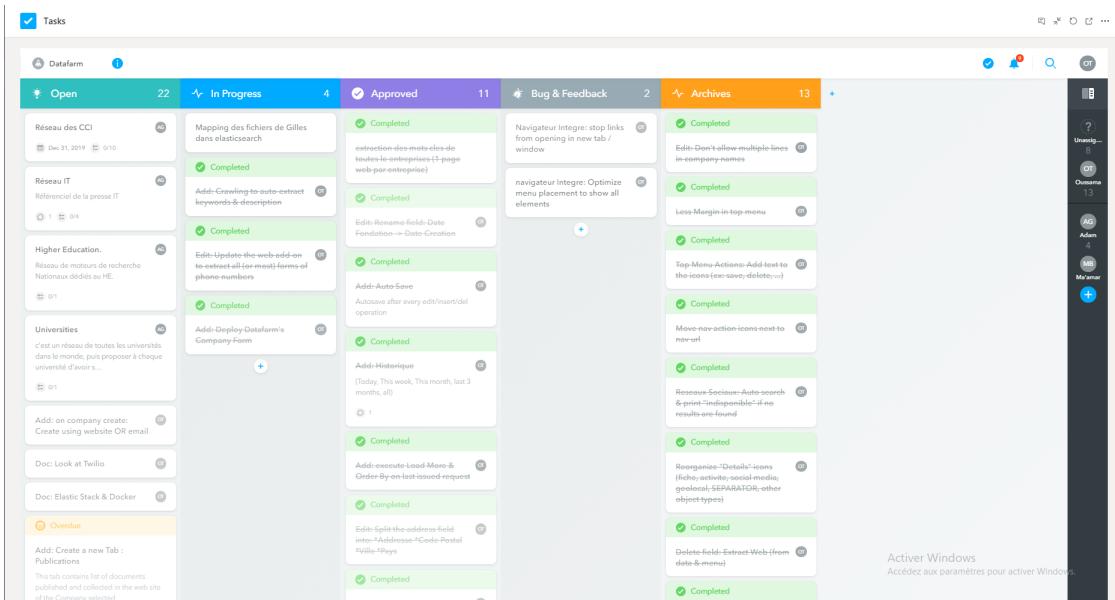


Figure I.1 – Visualisation des tâches sur meistertask

3.2 Diagramme de Gantt

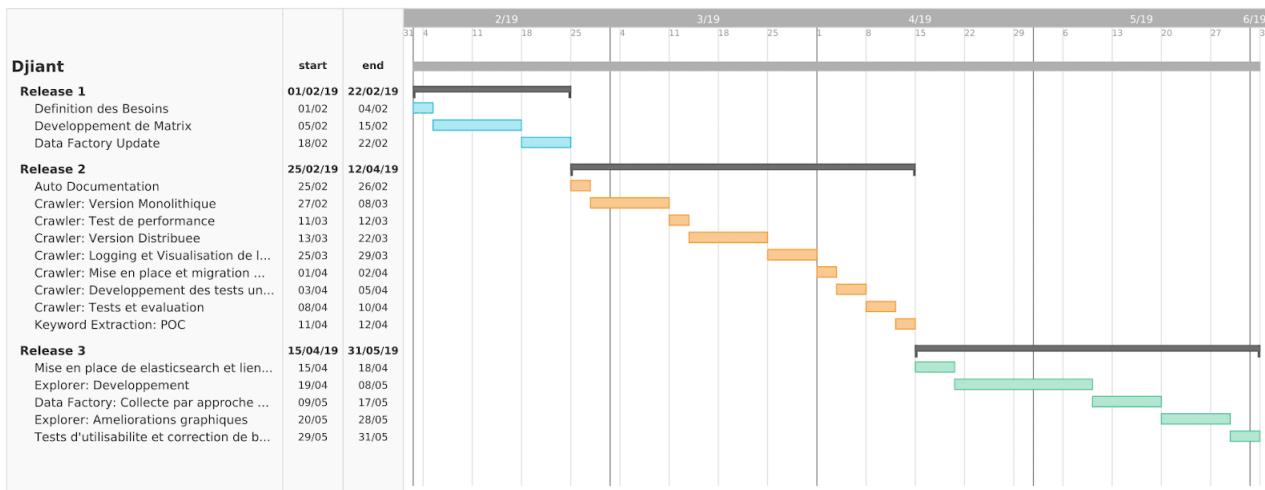


Figure I.2 – Tâches Effectuées

4 Cycle de développement

Au cours de ce stage, nous allons adopter un cycle de développement qui se compose de cinq étapes, dans le but d'arriver à un produit de bonne qualité sans devoir sacrifier l'agilité demandée par l'entreprise en matière de changements fréquents des besoins.

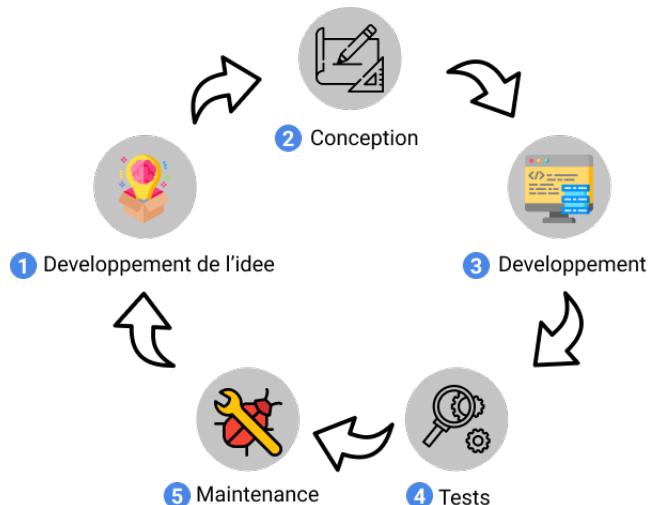


Figure I.3 – Cycle de développement

- Développement de l'idée : Avant de commencer les développements, une réunion est généralement consacrée à bien comprendre l'idée derrière le besoin exprimé qui va être plus tard développée en tâches détaillées par le développeur.
- Conception : Une fois l'idée bien décrite, nous passons à concevoir cette idée d'un point de vue technique.
- Développement : Lors de cette étape, nous implémentons la solution proposée par le développeur pour répondre au besoin exprimé au début du cycle en validant tout au long de cette étape l'avancement avec la personne responsable de la définition des besoins à travers des réunions régulières.
- Tests : Une fois les développements terminés, si le module développé présente une interface qui permet de communiquer avec un utilisateur, une série de tests est effectuée par un membre externe à l'équipe de développement afin de valider l'utilisabilité de l'application ainsi que détecter les bugs possibles. Sinon, si le module ne présente pas cette option, tel que le cas du module de crawling que nous allons détailler dans la suite de ce rapport, une série de tests unitaires est développée afin d'assurer le bon fonctionnement interne de l'application et l'absence de comportements inattendus.
- Maintenance : Cette étape est dédiée à la correction des bugs détectés dans l'étape précédente, aux modifications graphiques mineures et au refactoring si nécessaire.

Conclusion

Lors de ce chapitre, nous avons exprimé l'idée derrière ce projet, ainsi que notre contribution dans sa réalisation et la méthode de travail adoptée afin d'arriver à notre but. Dans le chapitre suivant, nous allons détailler encore plus les aspects techniques avec une étude des besoins ainsi qu'une architecture du système qui permet d'avoir une vision globale sur le travail réalisé.

Chapitre II

Étude des besoins et Architecture

Plan

1	Identification des acteurs	9
2	Expression des besoins	10
2.1	Besoins fonctionnels	10
2.2	Besoins non fonctionnels	13
3	Architecture	15

Introduction

Dans ce chapitre nous présentons une vision globale sur l'architecture du projet ainsi que les besoins auxquels répond chacun des modules.

1 Identification des acteurs

Dans le cadre de ce projet, on distingue deux acteurs principaux :

- Les "Data Managers" : ce sont les employés de l'entreprise qui vont utiliser les applications de back-office, principalement pour la gestion des données avec une tâche secondaire qui est la gestion des listes de moteurs.
- L'utilisateur : c'est le client final qui exécute des requêtes de recherche sur la partie front-office.

2 Expression des besoins

2.1 Besoins fonctionnels

Le projet est découpé en cinq grands modules :

2.1.1 Data Factory

Data Factory est une application web qui sert à extraire les données des entreprises afin de les exposer à un ou plusieurs moteurs de recherche spécialisés. La figure II.1 présente les besoins auxquels répond ce module sous forme de diagramme de cas d'utilisation.

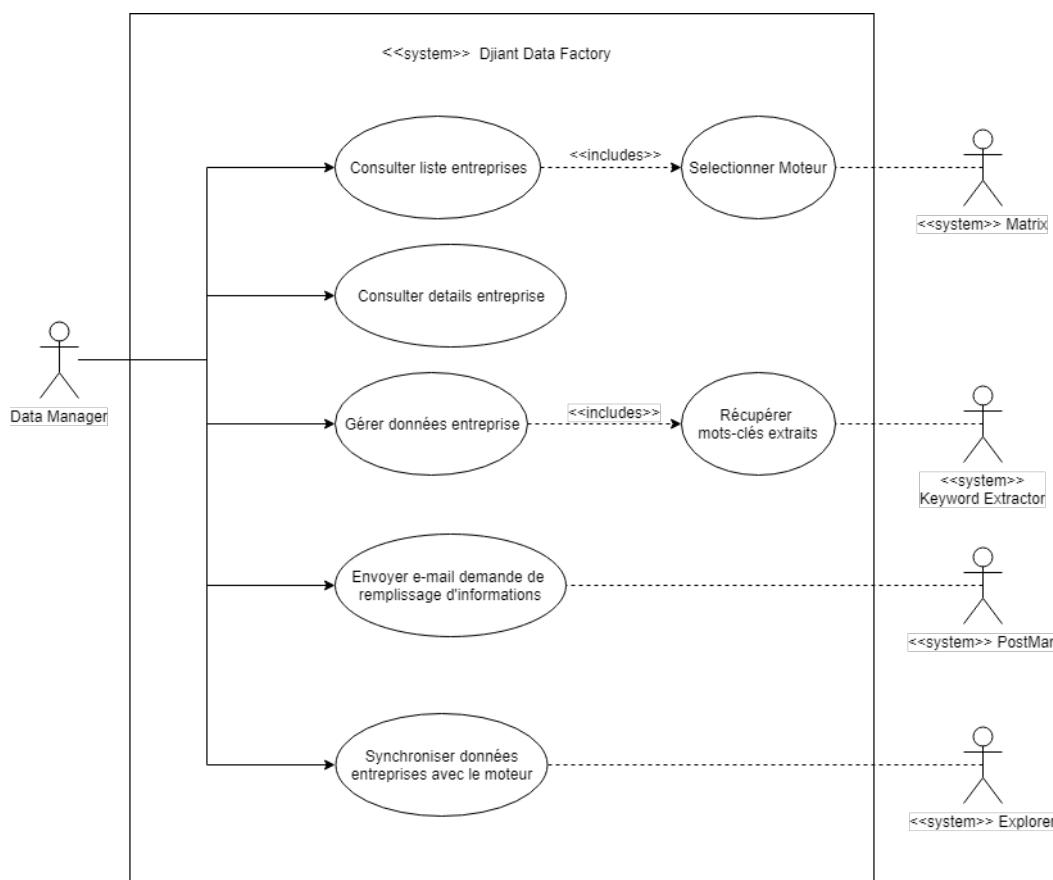


Figure II.1 – Data Factory : diagramme de cas d'utilisation

La collecte de ces données peut être effectuée de 3 manières :

- **Manuelle** : Un *Data Manager* utilise l'application pour sélectionner manuellement les données de l'entreprise à partir de son site web qui serait intégré dans l'interface de l'application.

II.2 Expression des besoins

- **Automatique** : Elle est effectuée par des scripts pour extraire automatiquement des données à partir des sites des entreprises, principalement des mots-clés.
- **Déclarative** : L'application doit permettre l'envoi d'emails pour inviter les entreprises à remplir des formulaires avec leurs données.

Une fois la collecte des données est effectuée, Data Factory doit permettre à son utilisateur d'envoyer les données collectées vers le moteur de recherche auquel elles appartiennent. Nous appellerons cette opération la “synchronisation de données”.

2.1.2 Matrix

Matrix est une application web qui référence la liste des domaines, des réseaux et des moteurs.

Ce module agit sur les données des moteurs de deux façons :

- Gestion de la liste des moteurs : Matrix offre une interface utilisateur qui permet d'ajouter, modifier ou supprimer un domaine, réseau, ou moteur.
- Référencement des moteurs : Matrix offre une api pour communiquer avec les autres modules de Djiant afin de permettre la récupération de l'arborescence formée par les domaines, réseaux et moteurs.

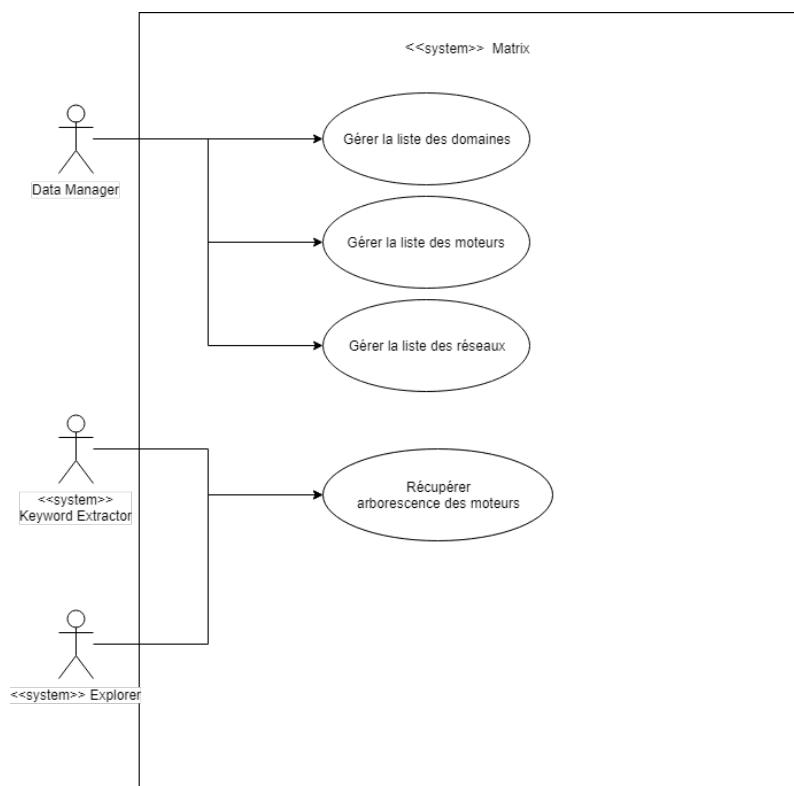


Figure II.2 – Matrix : diagramme de cas d'utilisation

2.1.3 Explorer

Explorer représente le front-office de la plateforme Djiant qui permet d'effectuer des requêtes sur les moteurs de recherche spécialisés.

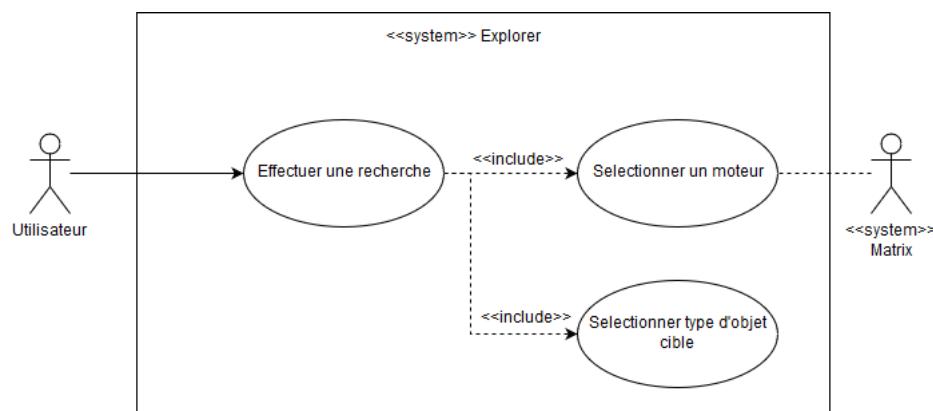


Figure II.3 – Explorer : diagramme de cas d'utilisation

2.1.4 Crawler

Crawler est un module qui permet la récupération des pages des sites web des différentes entreprises et leur stockage afin de les traiter ultérieurement par des scripts d'extraction d'information.

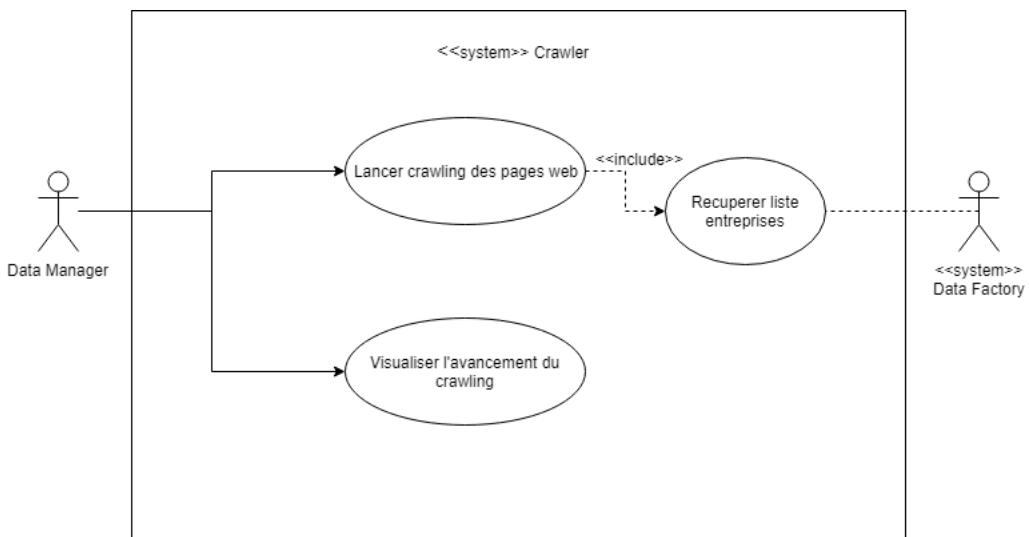


Figure II.4 – Crawler : diagramme de cas d'utilisation

2.1.5 Keyword Extractor

Keyword Extractor est un module qui sert à extraire des mots-clés à partir des pages web récupérées par le crawler et les exposer par une api web au module Data Factory.

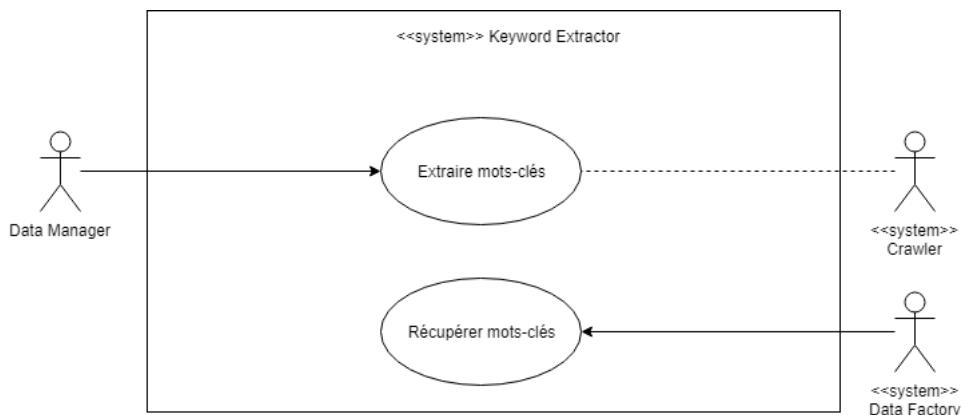


Figure II.5 – Keyword Extractor : diagramme de cas d'utilisation

2.2 Besoins non fonctionnels

Comme expliqué dans les sections précédentes, ce projet est formé par plusieurs modules et chacun d'eux présente ses propres besoins fonctionnels ainsi que non fonctionnels, nous allons donc présenter les différents besoins non fonctionnels sous forme de tableau qui regroupe les besoins et les modules qui leur sont associés.

	Data Factory	Matrix	Explorer	Crawler	Keyword Extractor
Sécurité	✓	✓	✓	✓	✓
Performance	✓		✓	✓	✓
Utilisabilité	✓	✓	✓		
Maintenabilité	✓	✓	✓	✓	✓

Tableau II.1 – Besoins non fonctionnels par module

- **Sécurité** : Ce besoin inclut tous les aspects de sécurité et non pas uniquement la sécurité applicative : la disponibilité du service, éliminer la possibilité de perte de données, la gestion d'accès aux services, etc ...
- **Performance** : La performance dénote, dans notre cas, un temps de réponse raisonnable. Tandis que ce critère est apprécié dans n'importe quelle application, il n'est pas critique pour tous les modules. Ce besoin a été assuré dans ce projet grâce à plusieurs

II.2 Expression des besoins

choix architecturaux et technologiques que nous allons détailler dans les chapitres suivants.

- **Utilisabilité** : L'utilisabilité est spécifique aux aspects graphiques et ergonomiques qui font qu'une application soit intuitive et facilement utilisable. Pour assurer cet aspect, nous sommes passés par plusieurs changements graphiques au cours de ce stage suite au retour d'information que nous avons eu durant les tests avec le reste de l'équipe ou les démos avec des parties externes à l'entreprise.
- **Maintenabilité** : La maintenabilité du code résultant de ce stage est l'élément le plus important pour deux raisons principales :
 - Au cours de ce stage, les changements de dernière minute étaient très fréquents, il fallait donc créer un code facilement modifiable afin de pouvoir suivre ce rythme.
 - Après la fin du stage, chaque module devrait être géré par au moins une personne, selon la complexité du module, afin de pouvoir assurer toutes les fonctionnalités futures proposées par l'entreprise. Il est donc absolument nécessaire que le code soit lisible et simple à debugger.

3 Architecture

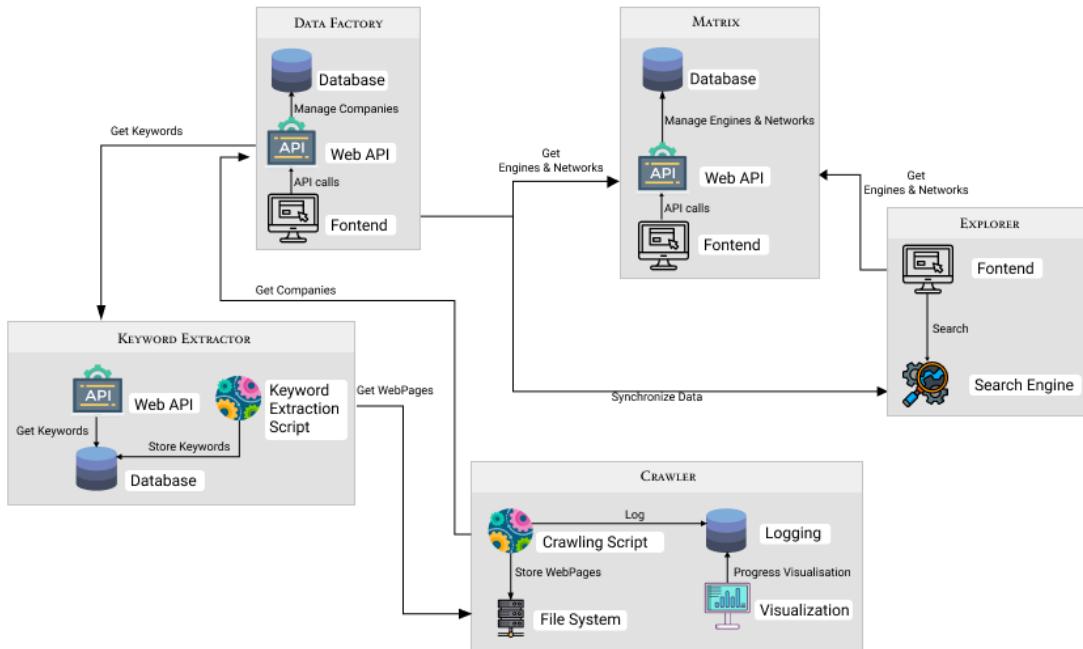


Figure II.6 – Architecture

Comme démontré par la figure II.6, ce projet se compose de cinq modules dans le but d'avoir une architecture qui respecte les principes de conception de base[2] tel que la séparation des préoccupations (Separation of Concerns).

En effet, chaque module représente une brique indépendante d'un point de vue technique, mais qui doit communiquer avec les autres afin d'apporter sa valeur ajoutée dans le produit final et c'est dans cette section que nous allons expliciter les différents liens entre les modules :

- Data Factory / Matrix et Explorer / Matrix : Les deux modules data factory et explorer doivent communiquer avec l'application Matrix afin de lister l'arborescence complète des moteurs Djiant (domaines, réseaux, régions et moteurs) vu que le choix du moteur à prendre en compte est la première étape dans ces deux modules.
- Data Factory / Explorer : Data Factory doit communiquer avec le module Explorer et plus précisément sa partie elasticsearch afin de pouvoir exporter les données collectées pour qu'elles soient sujets aux recherches effectuées par le public.
- Data Factory / Keyword Extractor : Data Factory communique avec l'api du keyword extractor afin de récupérer la liste de mots-clés suggérés pour chaque entreprise.

- Keyword Extractor / Crawler : Le keyword extractor doit pouvoir récupérer les données récupérée par le crawler.
- Crawler / Data Factory : Le module du crawler communique avec Data Factory afin de récupérer la liste des différentes entreprises listées afin de récupérer la liste de leurs sites web et de les parcourir

Conclusion

Dans ce chapitre, nous avons étudié les fonctionnalités offertes par chaque module du système global, ainsi que son architecture qui expose les liens entre les différents modules.

Dans les prochains chapitres nous allons détailler le travail réalisé au cours des trois releases.

Chapitre III

Première Release - Multi-Moteurs

Plan

1	Data Factory	18
1.1	Conception	18
1.2	Réalisation	21
2	Matrix	21
2.1	Conception	21
2.2	Technologies utilisées	24
2.3	Réalisation	26
3	Evaluation et perspectives	27

Introduction

Dans cette release nous préciserons les opérations nécessaires afin de permettre à la solution Djiant de gérer plusieurs moteurs de recherche spécialisés. Pour assurer ceci nous allons agir sur deux modules, Data Factory et Matrix.

La figure III.1 détermine le périmètre de cette release et sa contribution dans le résultat final du produit.

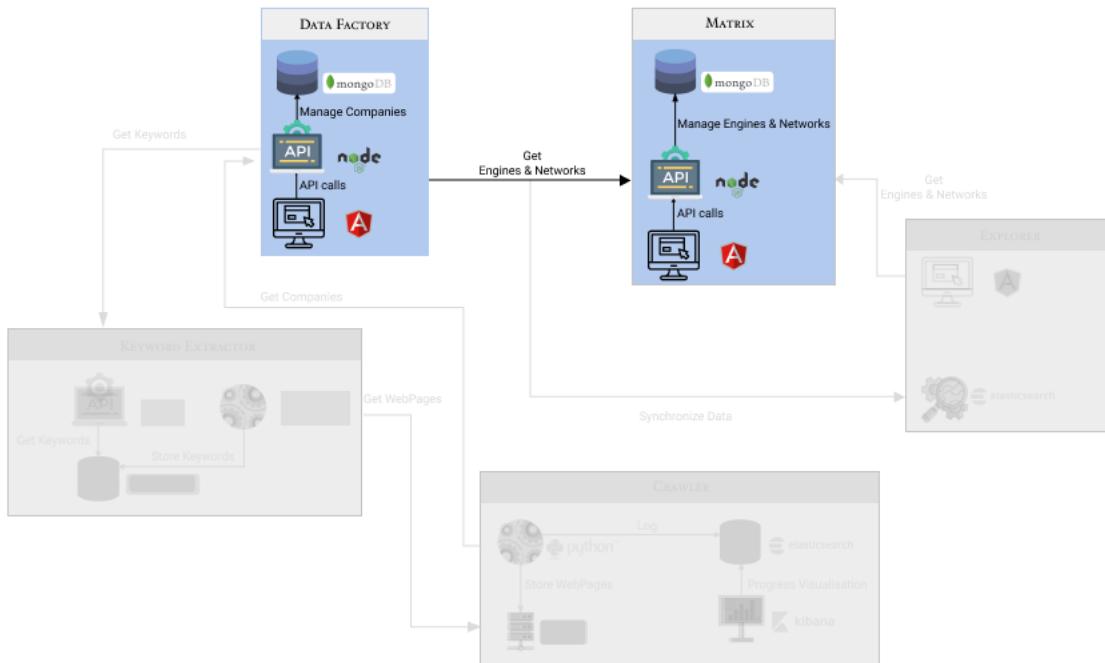


Figure III.1 – Périmètre du premier release

1 Data Factory

Nous présentons dans ce qui suit la première itération du module Data Factory. L'objectif de cette Itération est de trouver une solution qui assure le regroupement des entreprises par moteur tout en respectant les contraintes suivantes :

- Une Entreprise peut appartenir à un ou plusieurs moteurs.
- La cohérence des données des entreprises doit être gardée, c'est à dire, si nous mettons à jour les données d'une entreprise dans un moteur il faut que cette modification ait lieu au niveau de tous les moteurs auxquels elle appartient.

1.1 Conception

Pour les entreprises indexées par les moteurs Djiant, il est possible d'avoir des données qui sont présentées dans deux moteurs ou plus à la fois. Il fallait donc faire un choix entre deux alternatives :

1. Utiliser plusieurs bases de données, une par moteur.

Cette approche assure une meilleure disponibilité et une indépendance totale entre les différents moteurs vu la séparation physique entre les données.

III.1 Data Factory

De plus, grâce à cette séparation, l'opération de sélection de données par moteurs est beaucoup plus performante puisque les données contenues dans la base appartiennent toutes au même moteur et donc aucune opération de filtrage n'est nécessaire.

En contrepartie cette façon de faire nécessite la duplication des données des entreprises, ce qui rend plus difficile leur mise à jour et touche à la consistance des données.

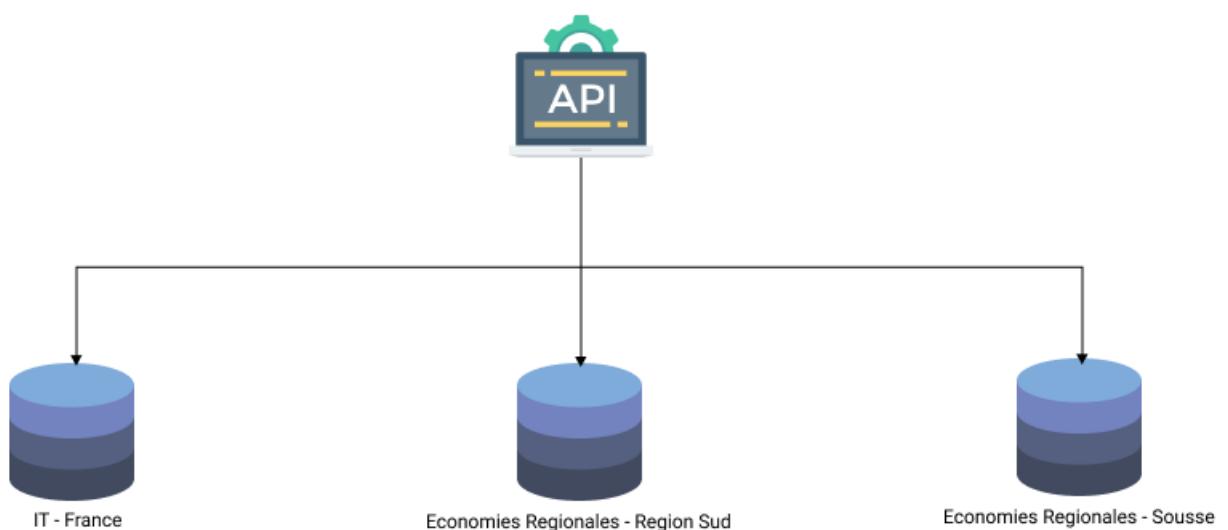


Figure III.2 – Première approche : une base de données par moteur

2. Utiliser une base centrale contenant une liste complète de toutes les entreprises qui contiennent, chacune, des références vers les moteurs dans lesquels elle devrait être présente.

Pour cette deuxième approche, la mise à jour des données d'une entreprise est une opération simple et atomique ce qui garantit la consistance des données et ne demande qu'une seule opération d'écriture, contrairement à la première approche. Pour l'opération de lecture par contre, afin de récupérer la liste d'entreprises appartenant à un moteur en particulier il faut filtrer les données par les identifiants des moteurs auxquels elles appartiennent ce qui peut être relativement lent pour de grosses masses de données.

III.1 Data Factory

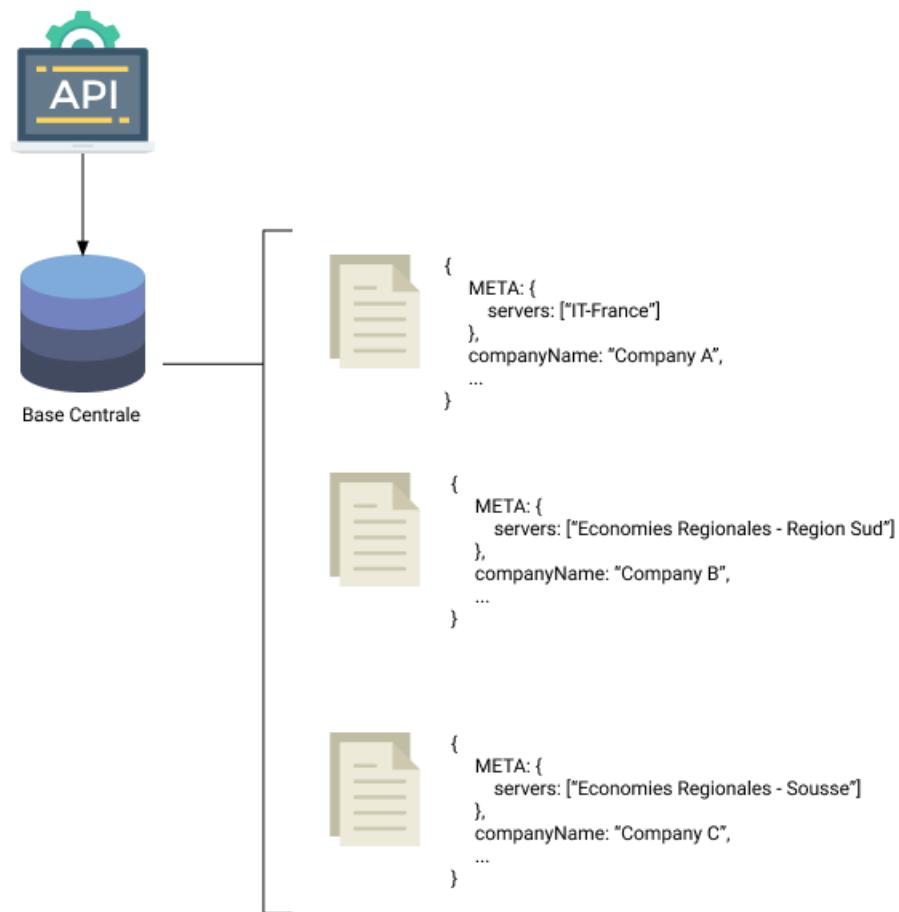


Figure III.3 – Deuxième approche : une base de données centrale

En tenant compte de l'importance de la consistance des données pour le module Data Factory ainsi que le nombre d'opérations d'écriture réalisées par l'application (1 opération par entreprise) qui surpasse énormément le nombre d'opérations de lecture (1 opération par page chargée), la première approche semble être non adaptée au besoin présenté.

De plus, le problème de disponibilité causé par l'utilisation d'une grosse base centrale est facilement gérable grâce aux bases *NOSql*[3].

Nous optons ainsi pour la deuxième approche.

1.2 Réalisation

Pour sélectionner le moteur pour lequel nous allons manipuler les données des entreprises, nous avons implémenté deux menus, un pour afficher l'arborescence des thématiques : domaines et réseaux, et l'autre pour afficher la distribution géographique des moteurs.

Par exemple, le moteur «Region Sud» appartient au réseau «Economies Regionales» du domaine «Local» sous la région France. En sélectionnant ce moteur, seules les entreprises qui lui sont associées sont chargées.

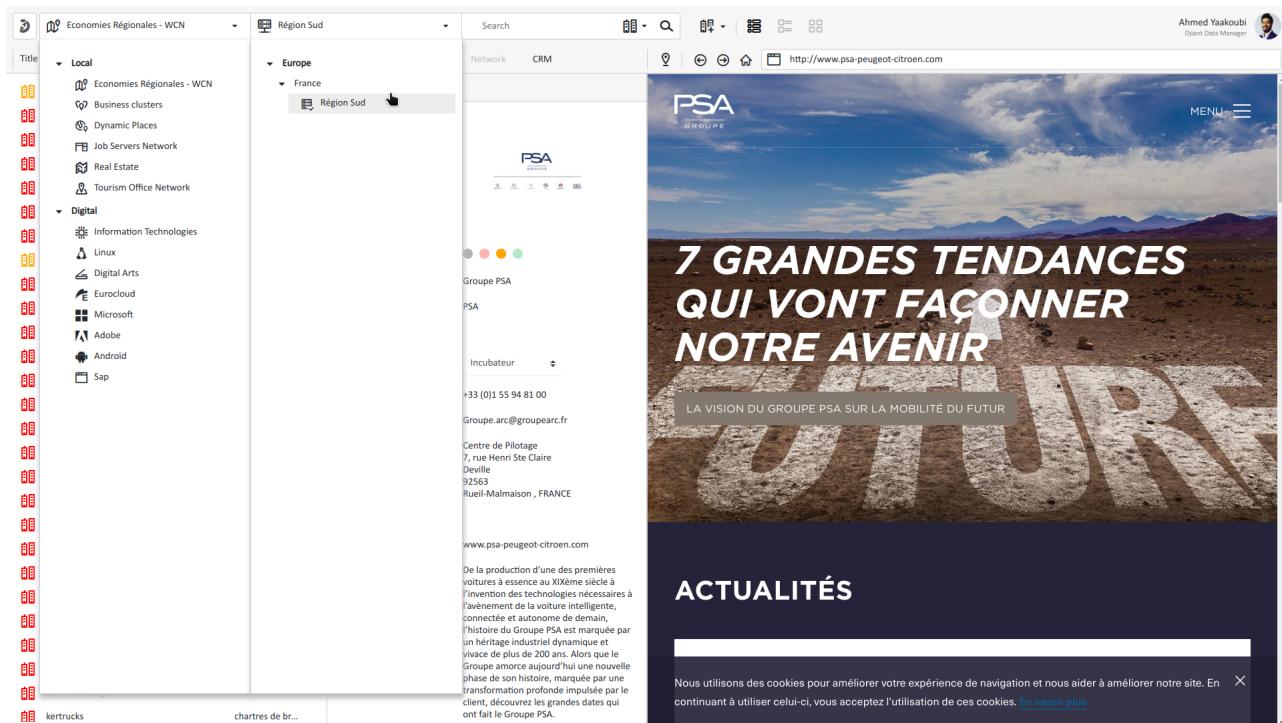


Figure III.4 – Data Factory : choix de moteur

2 Matrix

Nous présentons dans cette partie la première itération du module Matrix. Pour cette itération de matrix, nous allons faire en sorte qu'il permet de gérer l'arborescence des moteurs, c'est à dire la liste des domaines, réseaux et moteurs.

2.1 Conception

Au niveau de la base de données, trois modèles de données se présentent : Domain, Network et Engine.

III.2 Matrix

Ces modèles vérifient les conditions suivantes :

- Un domaine englobe un ou plusieurs réseaux, où les réseaux eux-même peuvent appartenir à plus qu'un seul domaine.
- Un réseau englobe un ou plusieurs moteurs, où chaque moteur peut appartenir à plus qu'un seul réseau
- Un moteur représente une et une seule région mais une région peut être représentée par plusieurs moteurs (un par réseau)

Pour exprimer la relation entre les moteurs et les régions nous avons eu à faire un choix entre deux façons de faire.

La première consiste à créer une arborescence commune entre tous les moteurs en concevant une région comme un objet identifié par le nom de cette région et contenant une ou plusieurs sous-régions ainsi qu'un ou plusieurs moteurs.

L'avantage de cette approche est le fait que le regroupement des moteurs par région est persisté dans la base, ce qui nous évite ainsi de faire un filtrage à chaque fois. Par contre, cet avantage est neutralisé par le traitement à faire afin de déterminer la liste des moteurs qui appartiennent à la fois au réseau et à la région sélectionnés.

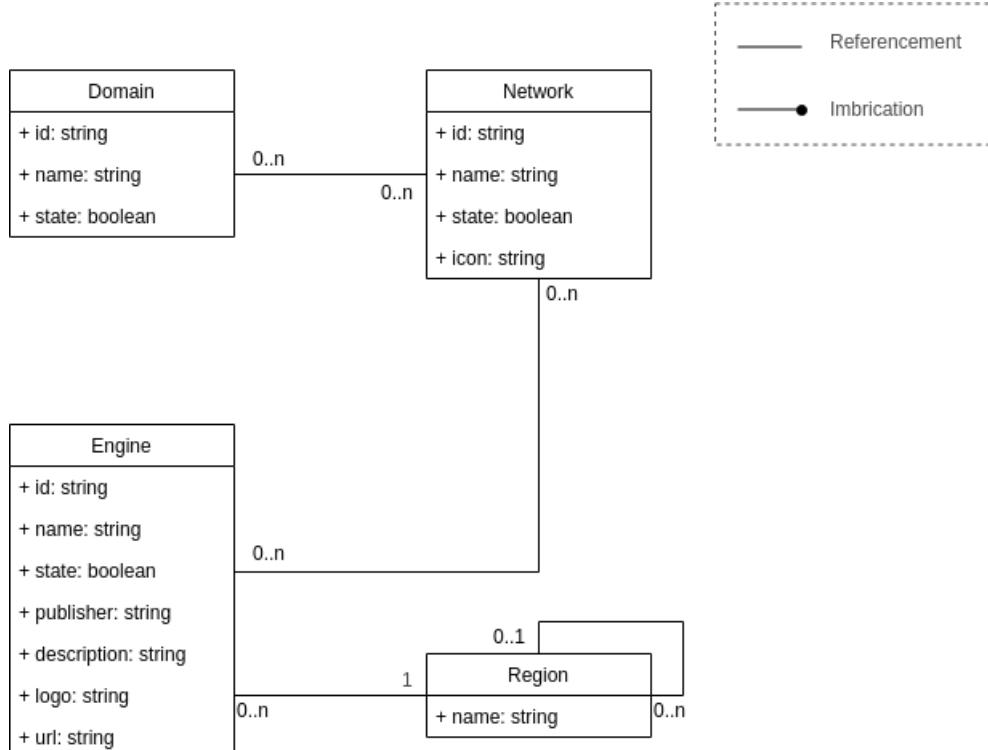


Figure III.5 – Matrix : Région en tant que modèle séparé

III.2 Matrix

La deuxième approche consiste à considérer la région à laquelle appartient le moteur comme un attribut. Par exemple, si un moteur est placé sous la region PACA, nous stockons au niveau du moteur le « chemin» vers cette région, soit «Europe/France/Paca».

En adoptant cette approche, nous avons besoin de faire un traitement pour regrouper les moteurs par région mais ceci offre une certaine flexibilité qui fait que chaque réseau de moteurs peut avoir sa propre arborescence.

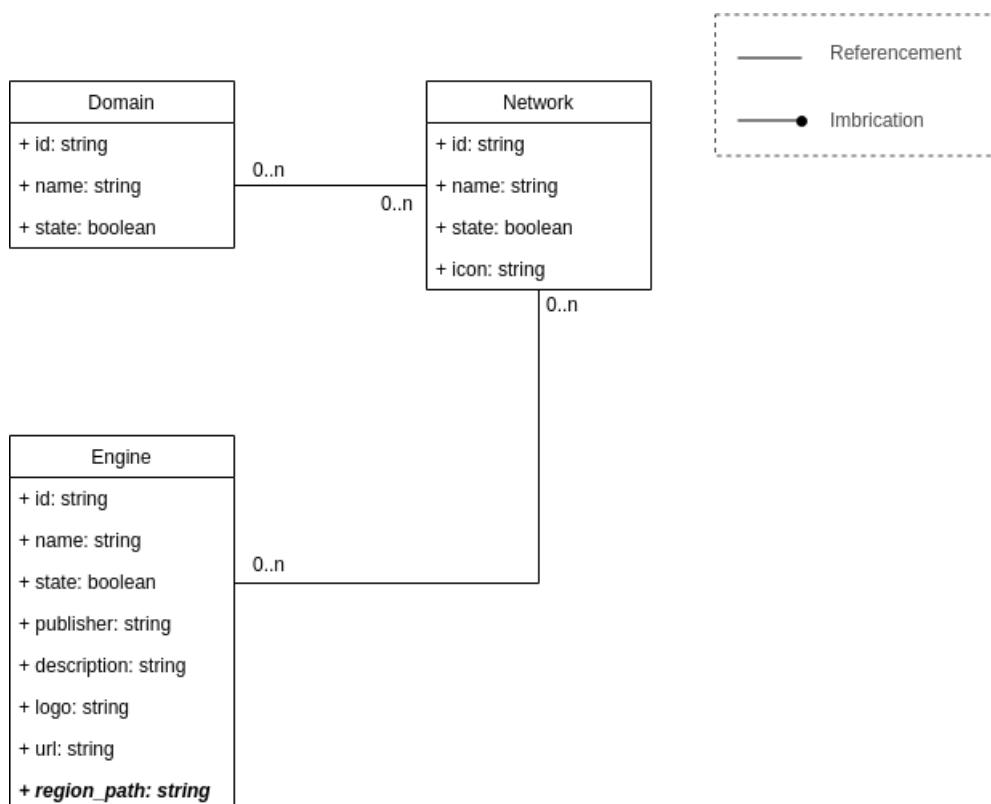


Figure III.6 – Matrix : Région en tant qu’attribut

En discutant des différents scénarios possibles,nous nous sommes rendu compte que pour certains réseaux les moteurs sont au niveaux des pays, c'est à dire l'arborescence se termine au deuxième niveau : continents puis directement les moteurs.

Pour d'autres réseaux, nous pouvons avoir des moteurs qui représentent plutôt des villes et donc nous avons des arborescences à 3 niveaux : continents, pays et puis les moteurs.

Nous avons donc opté pour la deuxième solution pour sa flexibilité vu que d'un point de vue de performance les deux solutions sont pratiquement équivalentes.

Au niveau de l' API web, nous allons concevoir une structure qui va être adoptée pour toutes les applications backend.

Puisque la plupart des opérations effectuées par cette API vont être des opérations CRUD, nous avons tiré profit du principe d'héritage pour définir une classe de base "GenericController" qui décrit les opérations CRUD traditionnelles afin de centraliser la définition de ces processus et donc simplifier leur maintenance et mise à jour ainsi que diminuer la durée nécessaire de développement des nouveaux modules.

En effet, il suffit de définir la structure du modèle de données pour avoir une API qui permet de le manipuler.

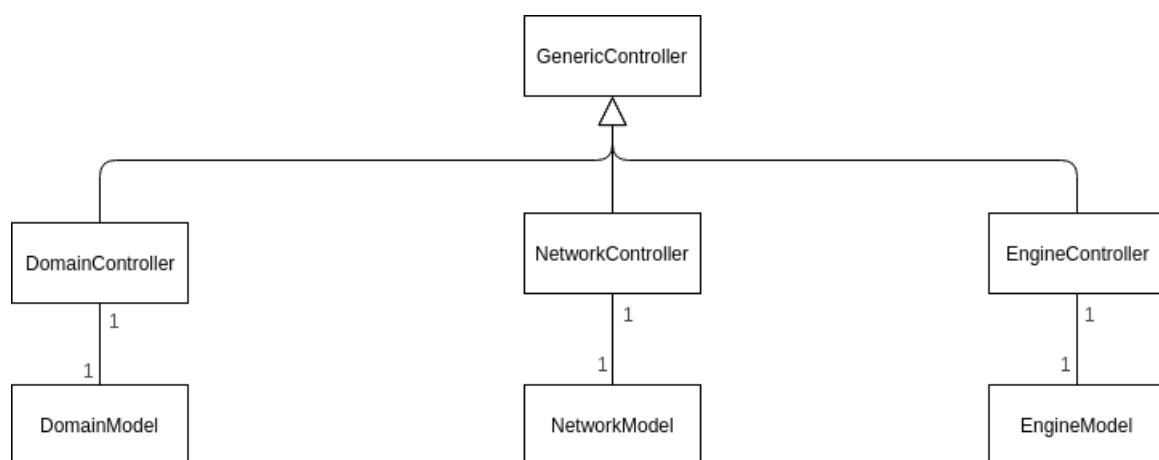


Figure III.7 – Matrix Web API

2.2 Technologies utilisées

— NodeJs :

NodeJs^[4] est un environnement d'exécution open-source pour le langage JavaScript.

Il est basé sur le principe de programmation évènementielle (*event-driven programming*), c'est à dire que NodeJs utilise un seul thread qui exécute un *event loop*^[5] où les opérations de lecture/écriture ne sont pas bloquantes ce qui permet de gérer des milliers de connexions concurrentes de façon efficace puisque ça nous évite le coût supplémentaire de changement de contexte si nous avions utilisé plusieurs threads. L'aspect non bloquant de la gestion des opérations de lecture/écriture est assuré par la notion de *callback*. Un *callback* est simplement une fonction qui serait passée en paramètre à une autre fonction afin de définir, de façon dynamique, un traitement effectué par cette dernière. En effet, quand une opération d'input/Output est terminée, l'*event loop* fait appel à un *callback* défini par l'utilisateur afin de consommer le résultat de cette opération.

Ce côté de NodeJs assure une performance très élevée pour les scripts où les opérations de lecture/écriture sont dominantes, mais à l'effet opposé pour les traitements riches en calculs complexes qui vont détenir l'event loop et donc bloquer la réception de toute nouvelle connexion. Vu que la majorité des opérations effectuées dans ce module sont des opérations d'interaction avec des bases de données et des requêtes HTTP, NodeJs est parfaitement adapté pour notre besoin.

— **Mongodb :**

MongoDB[6] est une base de données NOSql flexible qui persiste les données sous forme de documents JSON. Ces documents peuvent avoir des structures différentes ou qui évoluent au cours du temps. MongoDB est à la base une base de données distribuée ce qui offre une haute disponibilité ainsi que la possibilité de la mise à l'échelle verticale, c'est à dire le fait de pouvoir faire évoluer la base en taille et en performance en ajoutant de nouvelles machines par opposition à l'utilisation d'une seule machine centrale que nous faisons évoluer en ajoutant des composants (CPU, RAM, HDD, etc...). De plus, MongoDB utilise un mécanisme de réPLICATION qui assure encore plus de disponibilité pour les données fortement utilisées ainsi qu'un certain niveau de résistance aux pannes puisqu'une même donnée peut être présente dans plusieurs machines.

— **Angular :**

Angular[7] est un framework de développement d'interfaces web open-source maintenu par Google. Les éléments de base d'un projet Angular sont les composants, qui représentent des unités autonomes qui peuvent communiquer par plusieurs méthodes :

- Par passage de référence d'un composant à l'autre
- Par l'intermédiaire du composant parent
- A travers des services

Grâce à son architecture basée sur des composants, les projets Angular présentent un haut niveau de qualité de code ce qui assure plusieurs avantages :

- **Réutilisabilité** : L'autonomie des composants permet d'externaliser les parties semblables de l'application dans des composants auxquels nous pouvons faire appel dans plusieurs modules.
- **Lisibilité** : En organisant l'application en composants et services où chaque service représente une fonctionnalité et chaque composant un élément graphique de la page, il devient très facile de lire le code pour les nouveaux membres ou dans le but de l'évaluer.
- **Maintenabilité** : Le couplage faible entre les différents composants rend très

III.2 Matrix

facile le remplacement de ces composants ou la mise à jour des composants graphiques.

De plus, certains aspects techniques du framework, tel que l'injection de dépendance, assure une meilleure performance puisque les services ne sont pas dupliqués dans chaque composant mais sont référencés et consommés comme des ressources externes ce qui permet aux services et composants de s'exécuter en parallèle.

2.3 Réalisation

Dans cette partie nous présentons les interfaces créées qui illustrent le résultat du travail effectué au cours de cette itération.

The screenshot shows the 'Djiant Matrix' application interface. On the left, there's a sidebar with sections for 'Local' and 'Digital'. Under 'Local', there are icons for 'Add Network', 'Economies Régionales - WCN', 'Business clusters', 'Dynamic Places', 'Job Servers Network', 'Real Estate', and 'Tourism Office Network'. Under 'Digital', there are icons for 'Add Network', 'Information Technologies', 'Linux', 'Digital Arts', 'Eurocloud', 'Microsoft', 'Adobe', 'Android', and 'Sap'. The main area has two tabs: 'Domains & Networks' and 'Engines'. The 'Engines' tab is selected, showing a list of regions: Liechtenstein, Lettonie, Italie, Islande, Irlande, Hongrie, Grèce, France, Auvergne-Rhône-Alpes, Bourgogne-Franche-Comté, Centre-Val de Loire, Corse, Hauts-de-France, Grand Est, Île-de-France, Occitanie, Nouvelle-Aquitaine, Pays de la Loire, Finlande, Estonie, Espagne, Danemark, and Croatie. A specific entry for 'Région Sud' is highlighted with a blue background. To the right, there's a 'Details' panel for 'Région Sud' with fields for '_id', 'Name', 'URL', 'Publisher', 'Description', 'Logo', 'Background', 'State', 'Zoom', 'Longitude', 'Latitude', and 'Networks'. The 'Networks' field contains 'Local / Economies Régionales - WCN'. There are also 'Add Network' and 'Del' buttons. At the top right, there's a user profile for 'Olfa Rahmen'.

Figure III.8 – Écran Matrix

La figure III.8 présente l'interface de l'application matrix qui se présente sous forme de trois colonnes. La première colonne permet de consulter la liste des domaines et réseaux ainsi que l'ajout de nouvelles entrées du même type. La deuxième colonne permet de consulter la liste des moteurs appartenant au réseau sélectionné, regroupés par région. Elle permet aussi d'insérer un nouveau moteur. La troisième colonne permet de modifier les données de l'objet sélectionné : domaine, réseau ou moteur.

3 Evaluation et perspectives

Le travail réalisé sur les modules Data Factory est Matrix lors de cette release, peut être sujet à plusieurs améliorations.

Par exemple, au niveau du module Data Factory nous pouvons ajouter une fonctionnalité de gestion des permissions qui permet de personnaliser la liste des moteurs accessibles à chaque utilisateur selon les permissions qui lui sont accordées. Ceci, non seulement garantit une meilleure sécurité au niveau de l'application, mais élimine également les éléments non importants pour les Data Managers et par conséquent leur simplifie l'interface même si le nombre de moteurs augmente.

Au niveau de matrix, l'accès à l'arborescence des domaines, réseaux et moteurs est relativement lent. Une amélioration possible est l'utilisation d'une base "in-memory" tel que *Redis*[8], pour la mise en cache de cette arborescence qui est accédée très souvent en lecture mais mise à jour beaucoup moins fréquemment.

Conclusion

Ce chapitre nous a permis d'expliquer les choix que nous avons fait lors du développement d'une première version du module Matrix ainsi que notre première intervention sur le module Data Factory. Pour le chapitre suivant, nous allons nous intéresser à la collecte des pages web à partir des sites des entreprises et l'utilisation de ces pages dans le but d'extraire des mots-clés qui permettent de mieux indexer les entreprises gérées par Data Factory.

Chapitre IV

Deuxième Release – Crawling

Plan

1	Crawler	29
1.1	Itération 1 : Modèle de données et traitement asynchrone	29
1.2	Itération 2 : Traitement synchrone et optimisation	32
1.3	Itération 3 : Monitoring et persistance des données	36
2	Keyword Extractor	40
3	Evaluation et perspectives	41

Introduction

Cette release est dédiée à l'utilisation des pages web des entreprises afin d'extraire plus d'informations de façon automatique, principalement les mots-clés.

Dans un premier temps nous allons discuter du crawler qui va évoluer au cours de plusieurs itérations vers un service dont le rôle est ,uniquement, d'extraire les données brutes, qui sont des documents JSON englobant le code html des pages web ainsi que d'autres attributs qui vont servir de métadonnées tels que le timestamp de l'opération d'extraction, le site source, etc

Puis, nous allons procéder à présenter une première version du Keyword Extractor qui va servir de "proof of concept" avec un traitement relativement simple dedans afin d'avoir un MVP (Minimal Viable Product).

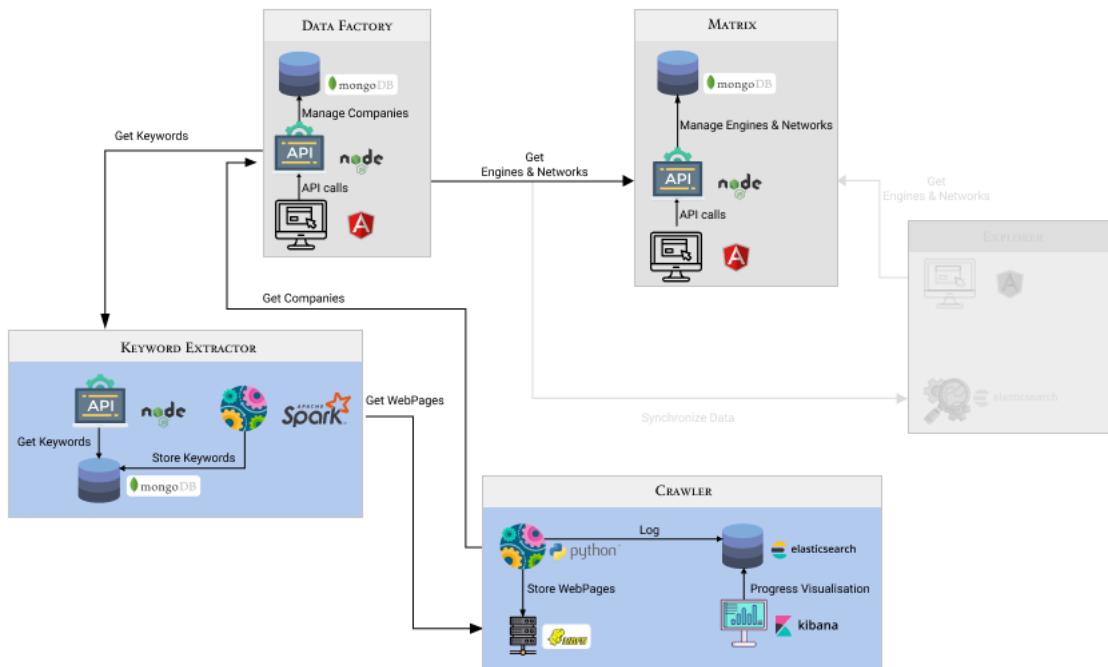


Figure IV.1 – Périmètre du deuxième release

1 Crawler

1.1 Itération 1 : Modèle de données et traitement asynchrone

1.1.1 Objectif

Pour cette première itération au niveau du module de crawling, nous visons à créer une version simpliste qui permet d'extraire les mots-clés à partir des pages web des entreprises afin d'indexer ces pages ainsi que pour indexer les entreprises elles-même pour des recherches futures.

1.1.2 Conception

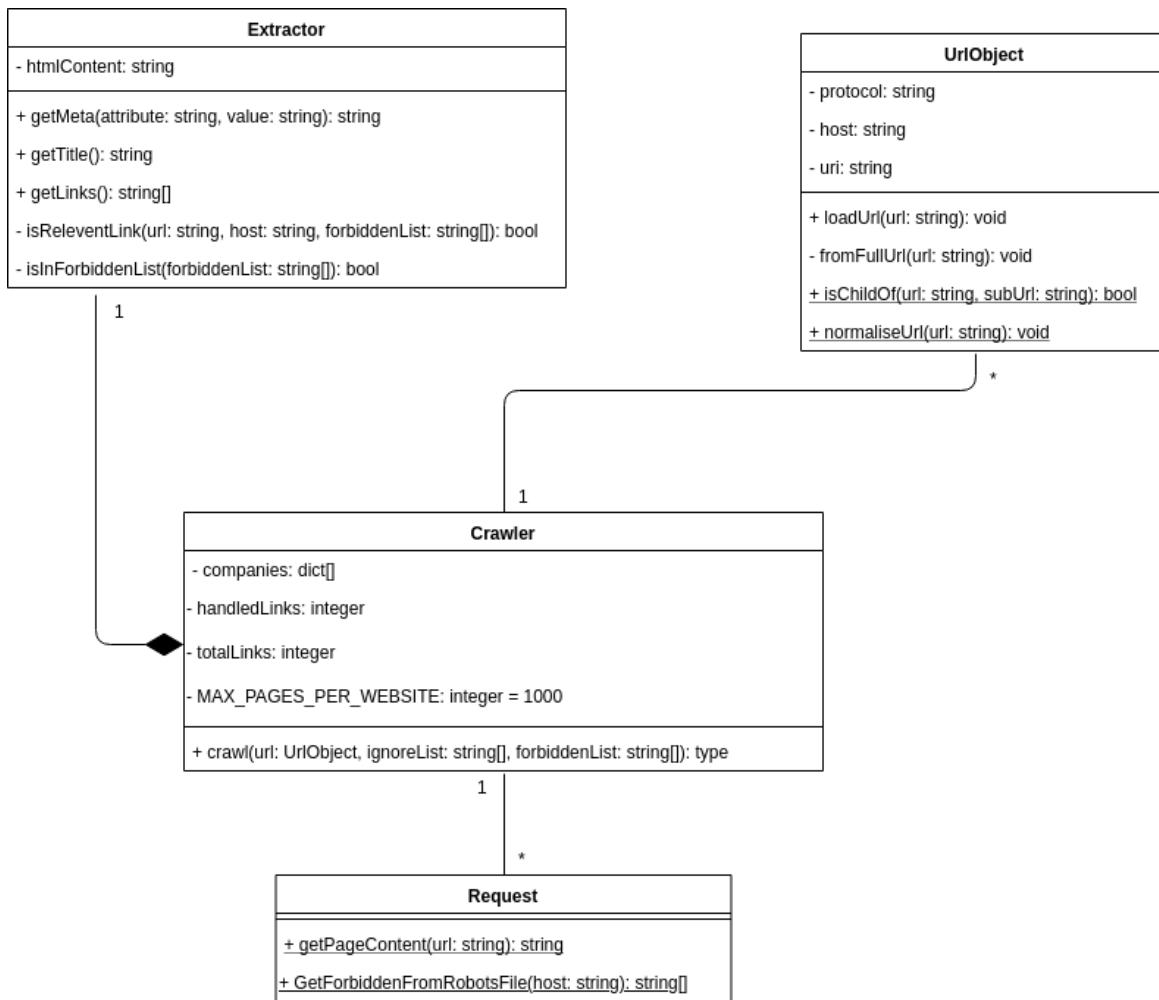


Figure IV.2 – Crawler itération 1 : Diagramme de classe

Comme présenté dans la figure IV.2, le module Crawler est composé de 4 classes :

- Request : Le rôle de cette classe est d'offrir des fonctionnalités qui permettent la récupération des pages web, ainsi que la liste des chemins où le crawling est interdit par le site qui existe dans le fichier «robots.txt» sous la racine du site.
- Extractor : Le rôle de cette classe est d'offrir les fonctionnalités d'extraction de données à partir du contenu html qui a été récupéré.
- UrlObject : Cette classe représente une couche d'abstraction qui simplifie la manipulation des URLs de formats différents afin d'avoir un code simple à comprendre et facilement maintenable.
- Crawler : Elle représente la classe principale qui fait appel aux fonctionnalités offertes

par les autres classes du module afin d'enchaîner les opérations de récupération des pages web pour chaque site ainsi que d'assurer certaines contraintes tel que le nombre maximal de pages traitées.

Elle permet aussi de suivre l'avancement du script.

1.1.3 Choix technologiques

Pour le module de crawling le choix technologique n'était pas aussi évident que pour les autres modules du projet Djiant.

En effet, la première réflexion que nous avons eu était d'utiliser le framework NodeJS[4] pour sa performance élevée avec les opérations de lecture/écriture qui sont les opérations les plus présentes dans ce module.

En testant cette idée, nous avons remarqué que malgré la performance offerte par ce framework, l'aspect asynchrone rend très difficile la détection de liens dupliqués, c'est à dire que si certains liens vers les pages web sont présents dans plus qu'une seule page, il serait très difficile de détecter cette duplication après leur extraction puisque ces pages web vont être traitées en parallèle.

Pour remédier à ce problème, nous avons essayé d'implémenter des solutions de synchronisation entre les différents callbacks, mais nous avons remarqué que cette approche non seulement élimine presque complètement l'avantage de la performance élevée mais aussi complique énormément le code.

Nous avons ainsi déduit que NodeJS n'est pas le choix le plus approprié à notre besoin et nous avons opté vers le choix d'un langage qui soit synchrone, ce qui nous amène à la seconde itération de ce module.

Grâce à son processus d'apprentissage relativement simple, sa large communauté et la multitude de bibliothèques qui lui sont associées, nous avons choisi d'utiliser le langage Python pour ce module.

1.2 Itération 2 : Traitement synchrone et optimisation

1.2.1 Objectif

Au niveau de cette deuxième itération, nous avons visé à remédier aux problèmes que nous avons identifié lors de l'itération précédente. Pour ce faire, nous avons pensé à une solution architecturale pour essayer d'améliorer les performances en terme de temps d'exécution nécessaire pour extraire les mot-clés des pages web de quelques milliers d'entreprises (7000 à-peu-près).

1.2.2 Architecture

Après avoir examiné les résultats de la première itération du script de crawling nous avons déduit deux contraintes à respecter :

- Il faut implémenter une solution de parallélisme afin d'améliorer la performance et avoir ainsi un temps d'exécution acceptable.
- Il n'est pas possible de paralléliser la récupération de pages appartenant au même site web.

Nous en déduisons ainsi qu'il faut paralléliser le traitement au niveau des sites web.

La récupération des pages web appartenant à un même site est toujours séquentielle mais nous allons exécuter l'opération de crawl sur plusieurs sites indépendants en parallèle. Il faut donc gérer les accès concurrents à la liste des sites web.

Afin d'assurer ces contraintes nous avons implémenté une architecture master-slave.

Le master a pour rôle de synchroniser les accès à la liste des sites web ainsi que de gérer certaines configurations des «slaves» tels que le nombre de sous-processus lancés pour chaque slave.

Un slave, que nous allons appeler worker dans notre cas, a pour rôle d'assurer le crawling d'un ou plusieurs sites web sur une machine à travers plusieurs processus lancés en parallèle, où chaque processus récupère les pages web d'un seul site à la fois.

En effet, au lieu de créer directement plusieurs instances indépendantes du script de crawling, par machine , nous avons opté vers l'utilisation d'une instance unique par machine ,que nous avons appelé worker, qui gère un groupe de processus indépendants qui effectuent le traitement parallélisé.

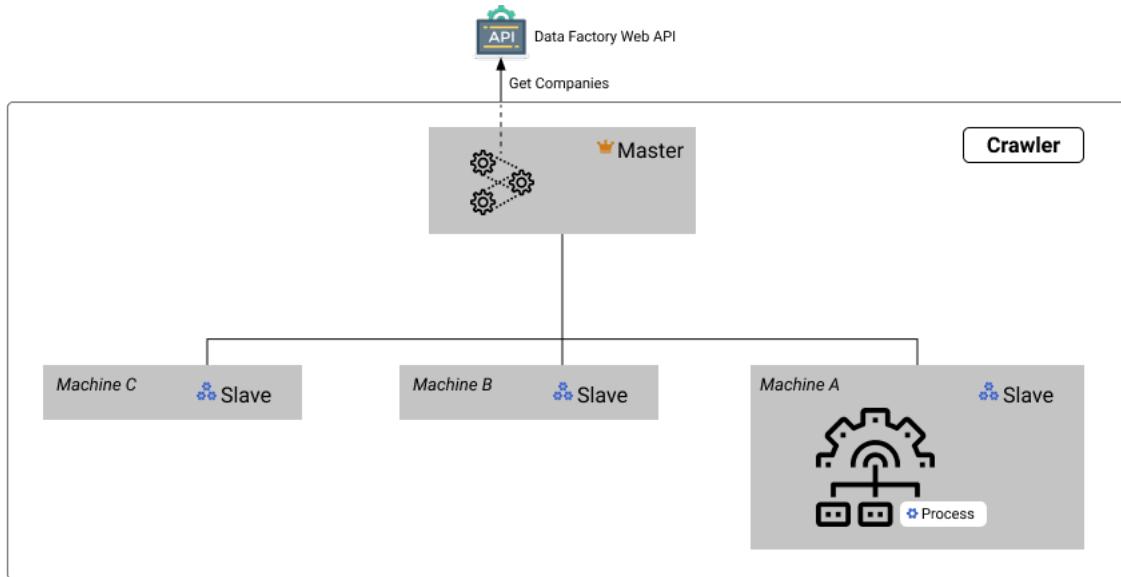


Figure IV.3 – Crawler itération 2 : Architecture détaillée

Cette approche présente deux avantages majeurs :

- Minimiser le nombre de connexions avec le master : nous avons une connexion par machine au lieu d'avoir une connexion par processus.
- Permettre la gestion facile et dynamique du nombre de processus lancés pour chaque machine, à travers la communication entre le Master et le Worker.

1.2.3 Conception

Dans cette partie, nous allons expliciter les classes utilisées et les liens entre elles dans le "Master" ainsi que le "Worker".

Master

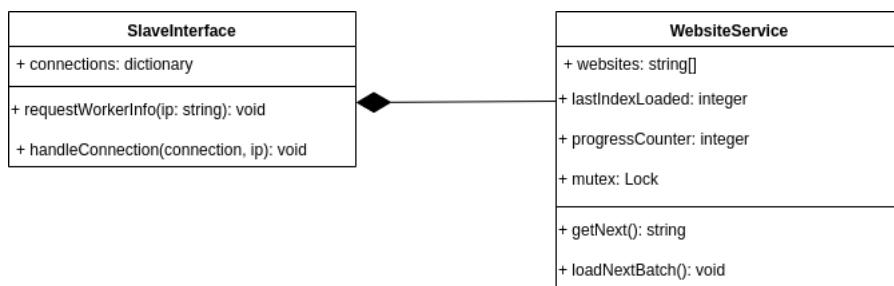


Figure IV.4 – Diagramme de classe du "Master"

Au niveau du "Master", nous avons deux classes :

- SlaveInterface : Elle joue le rôle d'une couche d'abstraction qui offre des fonctionnalités permettant de communiquer avec les workers
- WebsiteService : Elle permet d'interagir avec la liste des sites web.

Worker

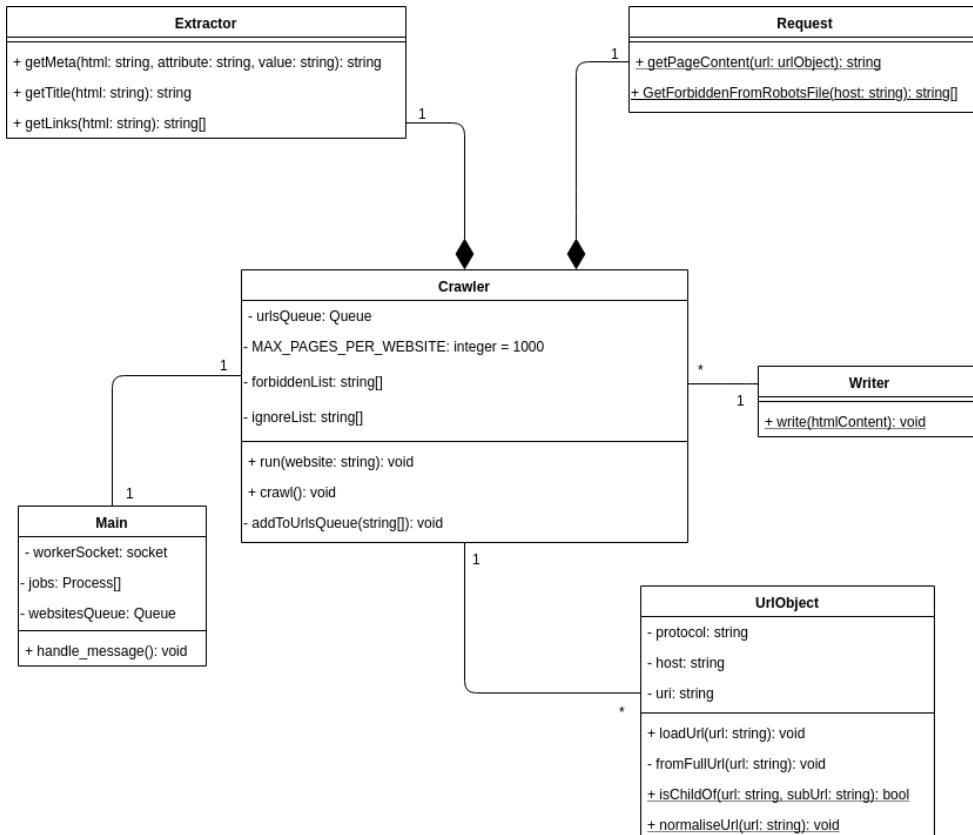


Figure IV.5 – Diagramme de classe du "Worker"

Au niveau du “Worker”, les classes Extractor, Crawler, Request et UrlObject sont identiques, d'un point de vue conceptuel, à celles de la première itération . Au niveau du traitement interne par contre, nous avons opté vers la séparation entre la récupération des pages et l'extraction des informations.

En effet, au cours de l'itération précédente nous avons remarqué qu'à chaque modification de la logique d'extraction des informations, nous sommes obligés de récupérer de nouveau toutes les pages web, ce qui est coûteux en terme de temps d'exécution.

Ceci a donné naissance à la classe Writer, dont le but est de présenter une couche d'abstraction pour la persistance des données indépendamment de la technologie utilisée.

De plus, par rapport à l'itération précédente, nous avons ajouté la classe “Main” où nous avons

implémenté la logique de parallélisation de ce traitement.

La figure IV.6 détaille les interactions entre les différentes classes afin d'assurer les fonctionnalités précédemment mentionnées.

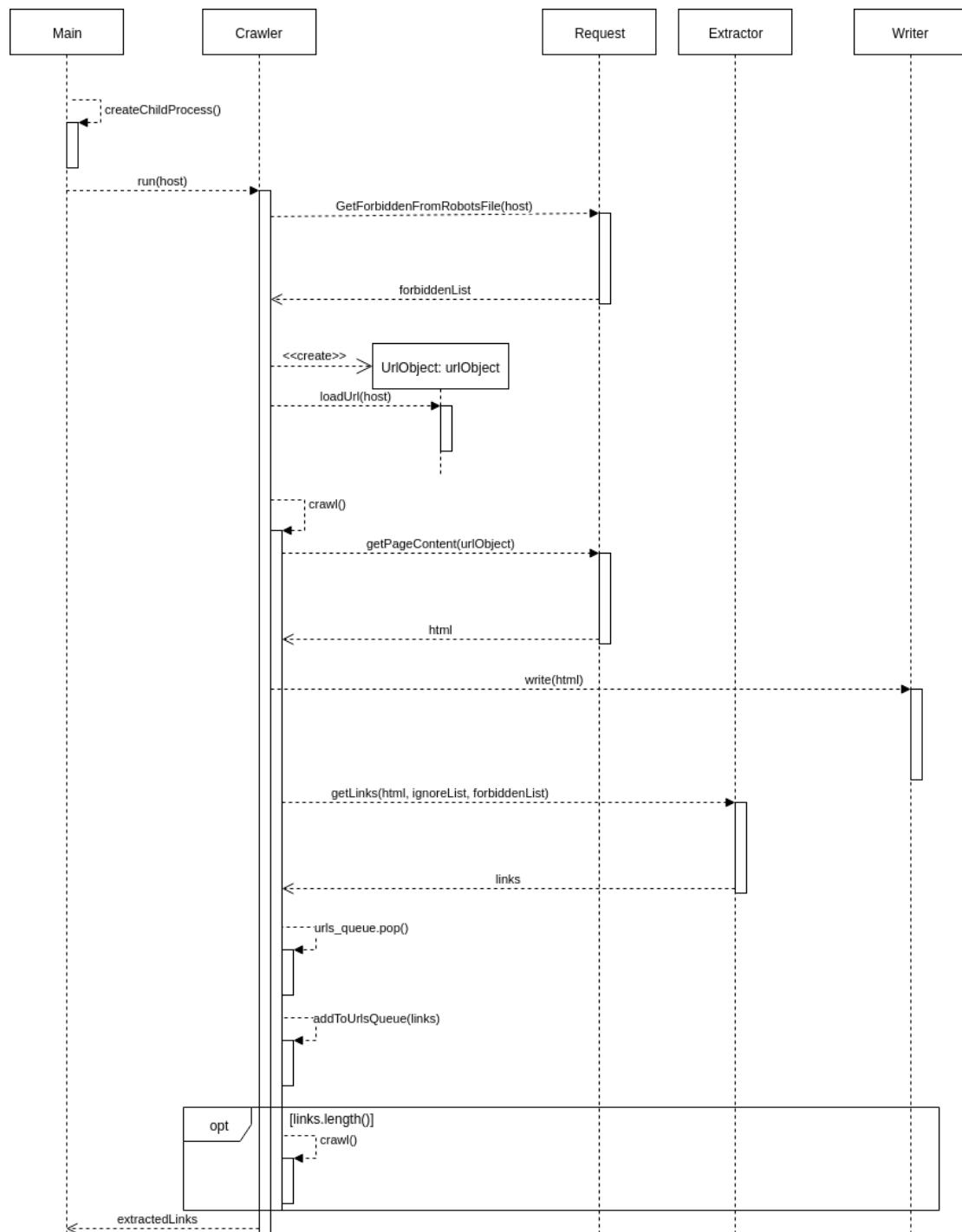


Figure IV.6 – Diagramme de séquence de l'opération de crawling

1.2.4 Choix technologiques

Afin de paralléliser le traitement au niveau du Crawler, nous avons eu le choix entre deux librairies python[9] :

- **Multithreading** : Elle permet l'exécution d'un traitement dans un nouveau thread.

Les threads sont distingués par leur aspect léger puisqu'ils partagent le même processus et le même emplacement mémoire sur la partie *heap* alors qu'ils exécutent chacun un code différent sur un espace mémoire *stack* dédié. A cause de ce partage de mémoire *heap* si une corruption de mémoire a lieu dans un thread, il est possible que ce problème va affecter les autres threads ainsi que le programme principal.

Les threads sont donc rapides à créer mais susceptibles au GIL (Global Interpreter Lock) qui interdit aux threads de s'exécuter simultanément pour un même programme, ils n'offrent donc qu'un parallélisme simulé.

Cette librairie présente d'autres limites tel que l'absence de la possibilité d'arrêter l'exécution d'un thread de l'extérieur.

- **Multiprocessing** : Elle permet de lancer des processus complètement indépendants, c'est à dire des zones mémoires et des CPUs séparés, ceci permet donc d'éviter les limitation du GIL et d'assurer un vrai parallélisme.

De plus cette librairie permet de gérer les processus facilement : les lancer ainsi que les arrêter.

Nous avons donc choisi d'utiliser le module Multiprocessing pour un meilleur contrôle sur les processus fils, un vrai parallélisme d'exécution du code et une meilleure résistance aux pannes assurée par la séparation totale entre ces processus.

1.3 Itération 3 : Monitoring et persistance des données

1.3.1 Objectif

Lors de cette itération nous allons agir sur trois points :

- Créer un mécanisme de log afin de permettre à la fois, le tracking de l'état d'avancement du crawler, ainsi que de continuer l'exécution du crawler a partir du dernier point d'arrêt du programme (en cas d'erreur par exemple).
- Pouvoir visualiser l'état d'avancement du crawler en temps réel
- Utiliser HDFS[10] pour persister les données au lieu du système de fichier local de la machine (ext4 dans notre cas).

1.3.2 Description

Afin de pouvoir s'assurer du bon fonctionnement du module et de pouvoir suivre l'avancement de son exécution au cours du temps, nous avons ajouté un segment de gestion de log. Les messages du log vont être utilisés pour exprimer deux “variables” :

- L'état de l'opération de crawling pour un site : en cours - achevé
- L'état de l'opération de crawling pour une page : en cours - achevé - erreur

Ces messages log ne vont pas être utilisés uniquement pour le débogage si un problème technique arrive, mais aussi pour visualiser l'avancement de l'opération de récupération des pages web ainsi que pour pouvoir reprendre le crawling de son point d'arrêt si son exécution est interrompue avant la fin de la récupération de toutes les pages.

En effet, pour la partie visualisation, il suffit de faire des calculs relativement simples afin d'extraire les informations nécessaires, tel que de nombre de pages générées, le nombre de sites générées, le nombre d'erreurs, la performance au cours du temps (en pages par unité de temps), etc . . .

Pour la restitution du dernier état du Crawler, nous utilisons les messages log afin de créer deux listes :

- La liste des pages web à ignorer : celles qui sont déjà traitées et donc leur état dans le log est à “achevé” ou “erreur”
- La liste des pages web à traiter : celles qui ont l'état dans le log à “en cours”.

En créant ces deux listes pour chaque site qui a l'état mis à “En cours”, nous pouvons récupérer les valeurs des variables clés qui représentent l'état global du crawler avant l'interruption pour qu'il puisse continuer son exécution sans devoir tout refaire.

1.3.3 Choix technologiques

Log et visualisation

Vu que les messages du log vont être sujet à différents traitements afin de restituer l'état du Crawler et calculer les différentes mesures pour le suivi d'avancement, l'utilisation des systèmes traditionnels avec des fichiers textes persistés directement sur le système de fichier va rendre l'exploitation de ces messages trop difficile et coûteuse.

Il faut ainsi utiliser une technologie qui permet de gérer une grosse masse de données de façon efficace et qui facilite l'exploitation des données pour visualiser le résultat de certaines opérations de calcul et d'agrégation.

Ces spécifications nous ont fait penser à Elasticsearch et Kibana [11].

Persistence des données

Le premier facteur à prendre en compte lors du choix de la technologie à utiliser est le support de données de grande taille.

En effet, avec les données que nous allons utiliser pour cette première étape de développement du produit qui va servir principalement pour des démos d'un seul moteur spécialisé nous allons utiliser une base de données de 7000 entreprises et donc 7000 sites web.

En testant sur une portion de 200 sites web, la taille des données produites a dépassé 10GB de stockage, et donc nous avons estimé un total de 350 GB pour les pages d'un seul moteur.

Il est donc impératif que la solution choisie soit facile à faire évoluer en termes de stockage.

De plus, la haute performance au niveau de la sauvegarde des données est un facteur clé dans le choix de la technologie à adopter.

Enfin, il faut s'assurer que les données ne soient pas perdues à cause des pannes possibles et donc la solution choisie doit être résistante aux pannes.

En prenant tous ces facteurs en compte, nous avons choisi d'utiliser HDFS[10], pour le stockage des pages récupérées.

En effet, grâce à l'aspect distribué de HDFS nous gagnons à la fois au niveau de la facilité de faire évoluer le système global et au niveau de la performance en écriture.

De plus, le système de duplication des données stockées sur HDFS assure un excellent niveau de résistance aux pannes, puisque même si une machine tombe en panne nous avons des copies des données qu'elle contienne distribuées sur le reste du cluster.

1.3.4 Réalisation

La figure IV.7 présente le tableau de bord généré par Kibana afin de visualiser les différentes mesures qui expriment l'état d'avancement du Crawler. Les informations exposées par ce tableau de bord sont :

- Le nombre de pages web selon l'état : en attente de récupération, récupérées, erreur.
- Le nombre de sites pour lesquels toutes les pages sont récupérées.
- Le nombre de pages par site.
- Le nombre de pages récupérées au cours du temps.

IV.1 Crawler

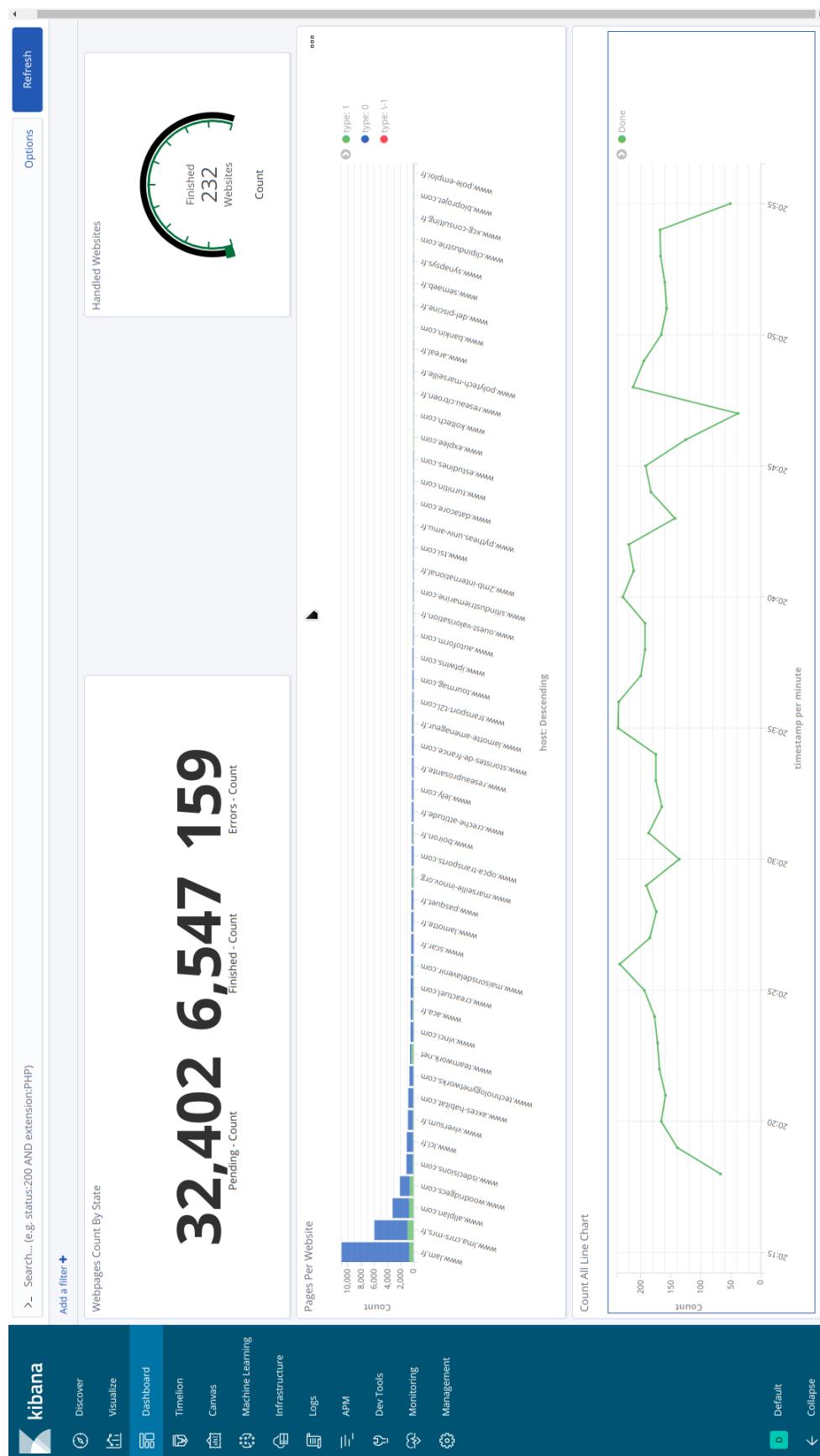


Figure IV.7 – Interface Kibana

2 Keyword Extractor

Cette première itération du Keyword Extractor va servir de proof of concept pour valider l'architecture que nous avons mis en place. Cette première version va se baser sur l'utilisation des mots-clés contenus dans les balises metas des pages web pour les indexer.

2.0.1 Choix technologique : Spark

Spark[12] est défini comme un moteur de traitement de données distribué polyvalent qui permet de communiquer avec plusieurs systèmes de stockage tel que HDFS, HBase, S3 de Amazon, etc...

Spark présente plusieurs avantages, pour notre besoin les plus pertinents sont :

- **Machine learning** : Spark est équipé de plusieurs bibliothèques dont MLlib[13], qui est une bibliothèque de machine learning intégrée. Pour cette release le but est toujours de créer un POC simple, mais la version finale du keyword extractor doit présenter un élément d'intelligence afin de garantir des résultats plus pertinents et donc une bibliothèque de machine learning est essentielle.
- **Performance élevée de traitement** : Ceci est dû à deux facteurs.
 - La possibilité de charger les données dans la mémoire.
 - Minimiser le nombre d'opérations de lecture/écriture sur le disque.
- **Tolérance aux fautes** : Les Resilient Distributed Datasets (RDD) sont la structure de données de base dans spark.
Une dataset RDD[14] est subdivisée en partitions logiques qui peuvent être distribuées sur plusieurs noeuds pour les traitements qui sont généralement les mêmes noeuds utilisés par HDFS ,dans notre cas, pour le stockage des données. Spark hérite donc sa tolérance aux fautes de celle de HDFS.
- **Facilité d'utilisation** : Spark offre plusieurs opérations de haut niveau qui assurent beaucoup de simplifications pour le développement de solutions parallèles.

3 Evaluation et perspectives

Au niveau du module “Crawler”, la performance est très acceptable en terme de temps d’exécution. En 15 minutes nous avons pu extraire les pages web de 120 entreprises en utilisant une seule machine contenant un Worker qui lance l’exécution de 10 processus sur cette machine, ce qui indique que dans le cas d’utilisation d’un cluster de trois machines identiques à celle que nous avons utilisé pour nos tests, cette opération peut se faire sur la totalité des 7000 entreprises sauvegardées dans la base dans une durée de 15 heures à-peu-près, ce qui représente une grande amélioration par rapport à la première version du Crawler.

De plus, nous avons utilisé une série de tests unitaires, en utilisant la bibliothèque “unittest” de python, afin de vérifier le bon fonctionnement des différentes classes qui composent la partie “Worker” du module “Crawler” vu que cette partie est la partie la plus complexe et la plus critique du module.

Par contre, nous pouvons améliorer ce module en introduisant une interface web qui permet de communiquer avec les “Workers” à travers le noeud “Master” afin de :

- Pouvoir consulter leurs informations de base : nombre de processus actifs par “Worker”, les états des “Workers”, etc ...
- Pouvoir les gérer à chaud en ajoutant de nouveaux processus, en arrêtant l’exécution de certains processus, etc ...

Au niveau du module “Keyword Extractor”, l’amélioration la plus évidente et la plus essentielle est d’ajouter un degré d’intelligence à l’extraction des mots-clés en implémentant des modèles de machine learning par exemple.

Conclusion

Au cours de ce chapitre, nous avons discuté des modules qui interagissent avec le web afin d’enrichir notre système avec des informations qui peuvent aider avec la pertinence des résultats de recherche, et c’est dans le chapitre suivant que nous allons introduire l’aspect recherche dans notre système.

Chapitre V

Troisième Release - Recherche, Approche déclarative et améliorations graphiques

Plan

1	Explorer	43
1.1	Iteration 1 : Interface de recherche	43
1.2	Iteration 2 : Améliorations graphiques	48
2	Data Factory	49
2.1	Iteration 2 : Synchronisation des données	49
2.2	Iteration 3 : Approche déclarative	51
3	Insertion de données supplémentaires	54
3.1	Description	54
4	Evaluation et perspectives	55

Introduction

Cette dernière release englobe plusieurs aspects de la plateforme Djiant.

D'une part, cette release englobe le processus de l'exploitation des données qui se résume à leur exposition pour être sujet à des recherches à travers le module "Explorer", et ceci en mettant en place un moteur Elasticsearch ainsi que l'interface qui lui est associée et le développement du lien entre Data Factory et la base Elasticsearch.

D'autre part cela inclut des changements graphiques proposés par l'entreprise afin d'améliorer l'aspect UX ainsi que le développement d'une dernière fonctionnalité clé pour le module Data Factory qui permet la collecte de données de façon déclarative.

La figure V.1 représente l'architecture résultante de cette release.

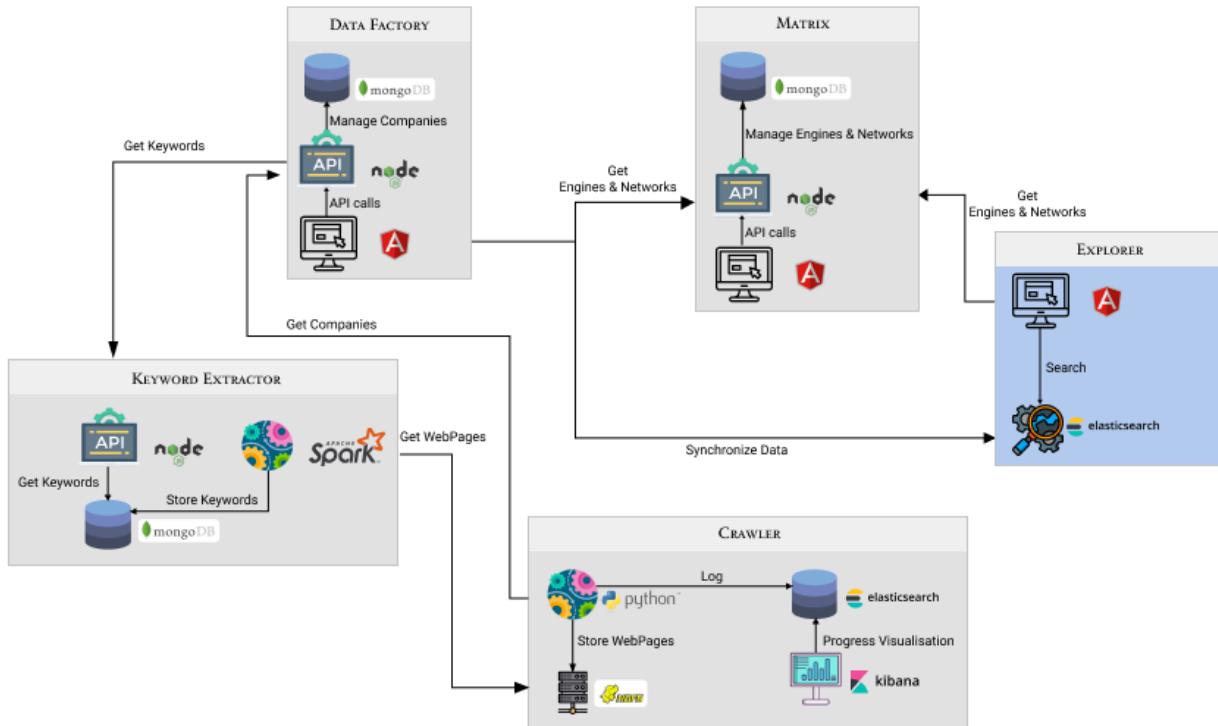


Figure V.1 – Périmètre du troisième release

1 Explorer

1.1 Iteration 1 : Interface de recherche

1.1.1 Objectif

Pour cette première itération, nous cherchons à développer les bases du module Explorer qui sont le moteur de recherche, sous forme d'une base Elasticsearch, et une interface Web.

1.1.2 Conception

Pour le cas du module Explorer, la base de données ne demande aucune conception puisqu'elle va être utilisée simplement pour indexer les données indépendantes les unes des autres, sans besoin de créer des relations entre les différents types.

Par contre, nous allons présenter les interactions entre les parties frontend et backend du module Explorer ainsi qu'avec les différents modules de Djiant à travers un diagramme de séquence système qui illustre les étapes d'exécution d'un scénario de recherche de base.

V.1 Explorer

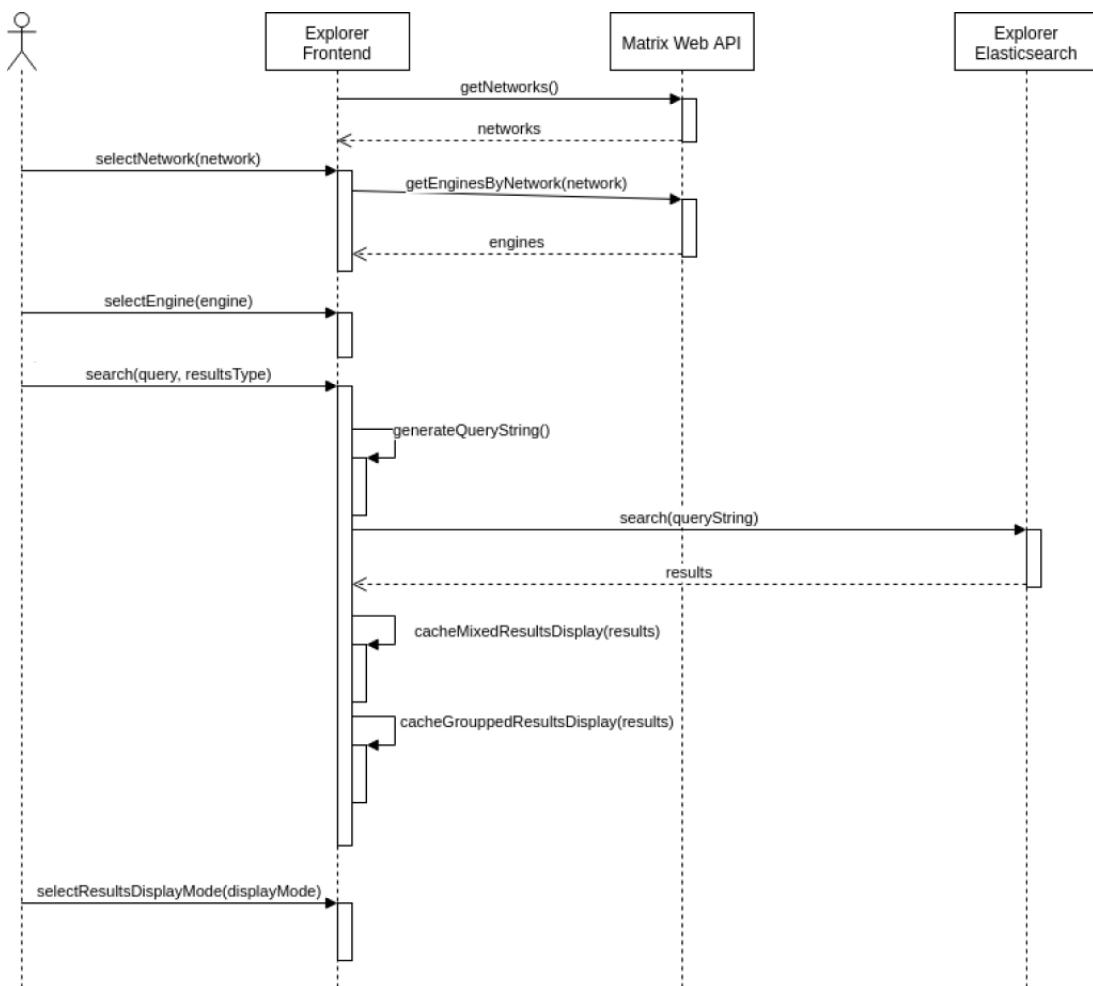


Figure V.2 – Diagramme de séquence d'une recherche

Comme exprimé par la figure V.2, l'utilisateur commence par choisir le moteur de recherche qui lui convient, puis tape la requête pour laquelle il cherche des résultats.

A ce moment la partie frontend d'Explorer traduit cette requête en un format accepté par Elasticsearch et envoie la requête vers ce dernier.

Une fois les résultats récupérés, un traitement relativement long est effectué afin de générer deux modes d'affichage des résultats :

- Le mode mixte : les résultats sont mélangés et ordonnés par leur pertinence par rapport à la requête saisie.
- Le mode regroupé : les résultats sont regroupés par type d'objet.

Afin d'éviter de refaire ce traitement à chaque changement du mode d'affichage, nous sauvegardons les résultats de ces deux opérations dans le *localStorage* du navigateur.

1.1.3 Réalisation

Dans cette partie nous allons présenter l'implémentation des étapes décrites par le diagramme de séquence système précédent.

Le but est de faire une recherche sur tous les types d'objets dans le moteur IT-France, puis afficher les résultats dans les deux modes d'affichage précédemment mentionnés, à savoir le mode mixte et le mode regroupé.

La première étape est de choisir le moteur dans lequel nous allons effectuer la recherche.

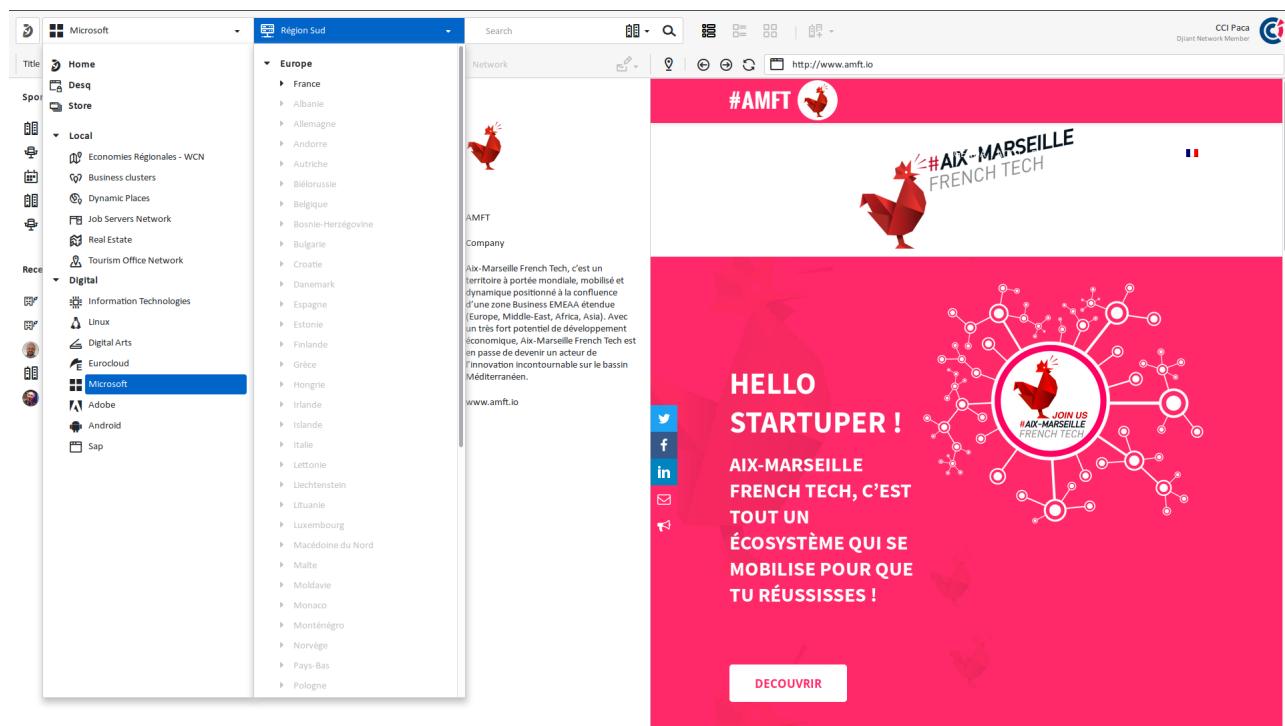


Figure V.3 – Explorer : choix de moteur

V.1 Explorer

La deuxième est de sélectionner le type d'objet pour lequel nous cherchons des résultats.

The screenshot shows the Le Monde Informatique search interface. A search bar at the top contains the text "Information Technologies /France". Below it, a dropdown menu is open, showing "Type" selected. Other options like "Details", "Publications", "Network", and "Me" are visible. To the right of the dropdown, there's a detailed profile for "Gilles BALMISSE" from "Djant". The profile includes a photo, basic information (Name, Type, Post, Company, City), a "Mini Bio" section, and a "Website". On the far right, there's a sidebar with various news articles and a "Who gefunded" section.

Figure V.4 – Explorer : choix du type d'objet

La troisième est de saisir la requête de recherche

The screenshot shows the ORSYS formation search interface. A search bar at the top contains the text "ITIL". Below it, a dropdown menu is open, showing "Type" selected. Other options like "Details", "Publications", "Network", and "Me" are visible. To the right of the dropdown, there's a detailed profile for "Formation Découverte d'ITIL® 2011". The profile includes a photo, basic information (Title, Type, Duration, Source, Description), and a "Programme" section. On the far right, there's a sidebar with various course details and a "Pour vous inscrire" section.

Figure V.5 – Explorer : Taper la requête

V.1 Explorer

Enfin, nous pouvons basculer entre les deux modes d'affichage des résultats.

The screenshot illustrates the 'Explorer' mode, which allows users to switch between two display modes: grouped and mixed. On the left, a sidebar displays a list of search results for 'ITIL'. On the right, a detailed view of a specific course titled 'Formation Découverte d'ITIL® 2011' is shown, including its title, type, duration, source, and a detailed description. The course description highlights ITIL® as a set of best practices for service management and promises to help understand the philosophy and objectives of ITIL® 2011 through five phases of a service's life cycle.

Figure V.6 – Explorer : Le mode d'affichage regroupé

The screenshot illustrates the 'Explorer' mode in 'mixed' view. On the left, the sidebar shows a list of search results for 'ITIL', with 'alcyor' selected. On the right, a detailed view of the company 'Alcyor' is displayed. The company profile includes its name, type (Company), activity (Services de conception de systèmes informatiques), description (Société de Conseil et Expertise en Système d'informations de la région lyonnaise (Iyon, Rhône/69). Nos interventions sont organisées autour de 3 axes majeurs qui sont Assistance à maîtrise d'œuvre (MOE) ou maîtrise d'ouvrage (MOA) et projets, Processus et Méthodes, Expertise Technique.), address (42 rue de l'université 69007 Lyon, france), website (www.alcyor.fr), and keywords related to ITIL, MOA, and project management. The Alcyor logo is prominently displayed at the top of the page.

Figure V.7 – Explorer : Le mode d'affichage mixte

1.2 Iteration 2 : Améliorations graphiques

1.2.1 Objectif

Grâce au retour d'information que nous avons eu après les démos qui ont été faites, nous avons eu des idées pour améliorer l'expérience des utilisateurs au niveau du module de recherche «Explorer».

Certains changements sont des modifications graphiques simples afin de rendre l'interface plus intuitive, mais l'élément le plus important c'est l'implémentation d'une carte qui rend le choix du moteur à utiliser plus intuitif aux utilisateurs.

1.2.2 Réalisation

Afin de rendre l'utilisation de l'application plus intuitive et de permettre aux utilisateurs de mieux visualiser l'idée de la grille de moteur de recherche spécialisés, nous avons remplacé la liste simple des moteurs présents dans chaque réseau par une carte où chaque moteur est représenté par un marqueur placé sur la région à laquelle il est associé.

De plus, en cliquant sur l'un des marqueurs sur la carte, ça permet de consulter les informations de l'éditeur du moteur : c'est une personne physique ou morale qui demande à Djiant de lui fournir un moteur qui lui serait associé et sur le contenu duquel elle aura un contrôle absolu.

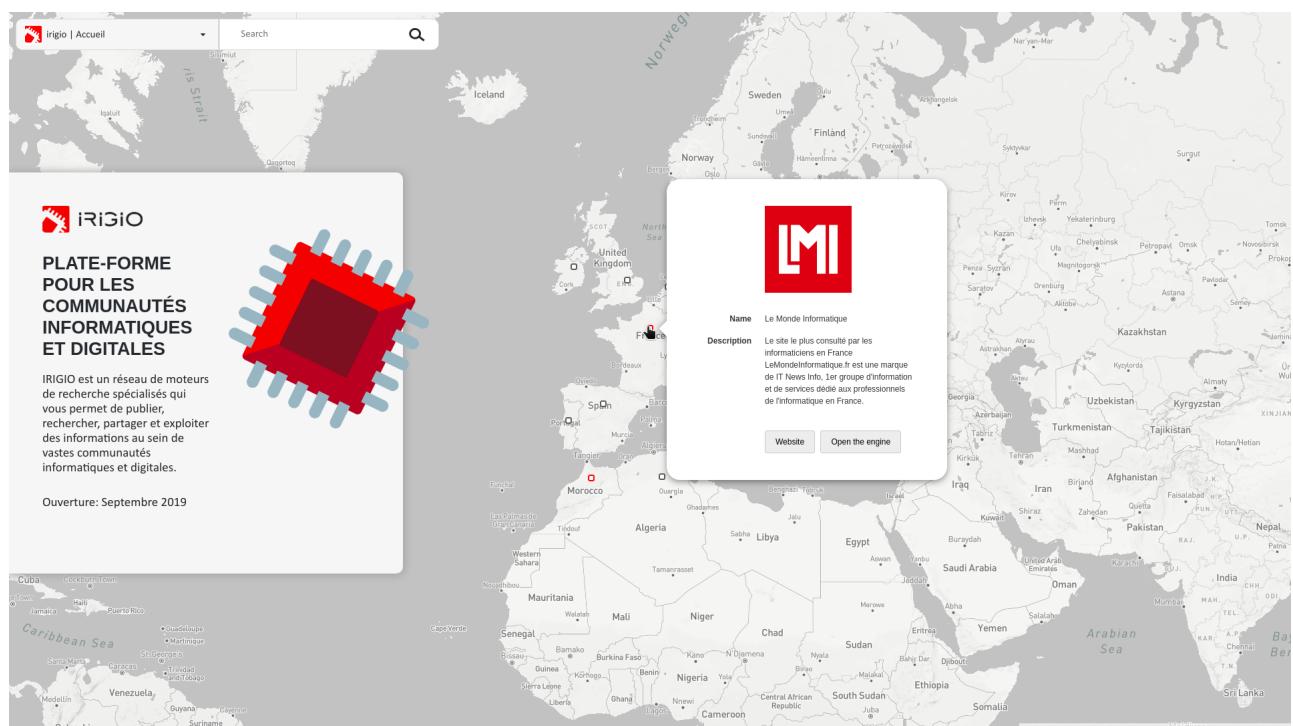


Figure V.8 – Carte de la liste des moteurs

Avec cette itération, nous avons clôturé les développements pour le module Explorer et nous nous sommes concentrés sur le module Data Factory afin de le finaliser à travers deux itérations supplémentaires.

2 Data Factory

2.1 Iteration 2 : Synchronisation des données

2.1.1 Objectif

Au niveau de Data Factory l'objectif principal est d'établir le lien entre les données extraites et les moteurs de recherche, c'est à dire, l'envoi des données à partir de la base de données du module Data Factory vers Elasticsearch afin de les indexer pour les opérations de recherche.

2.1.2 Conception

Dans cette partie, nous présentons les interactions entre les classes de l'API web, qui permettent la synchronisation des données entre Data Factory et les instances de Elasticsearch concernées par ces données.

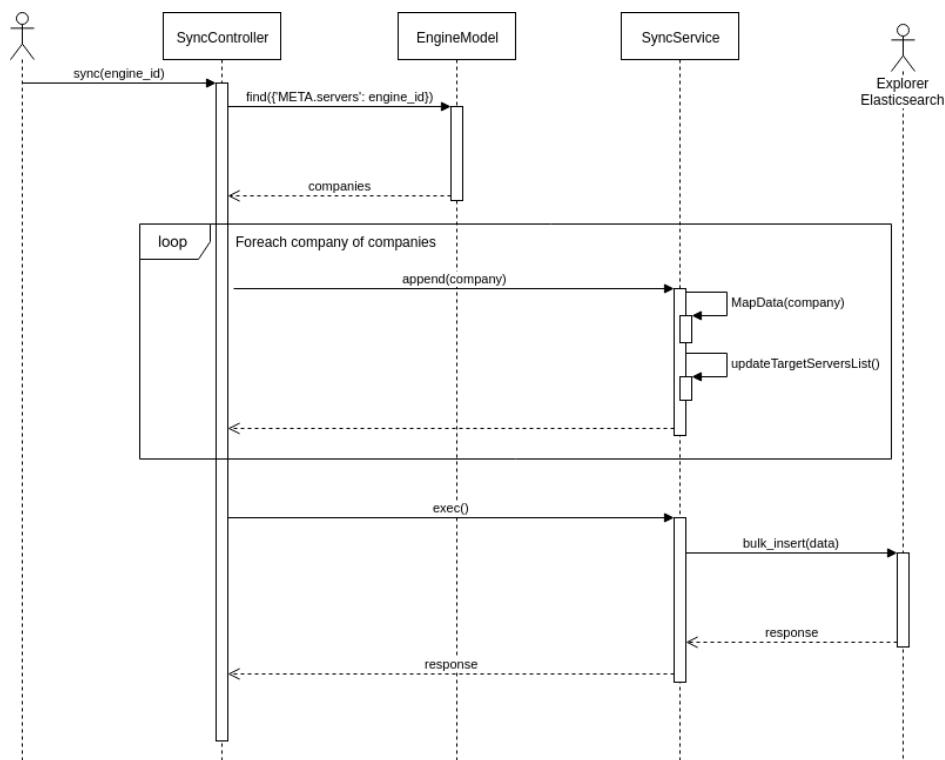


Figure V.9 – Diagramme de séquence de la synchronisation des données

Dans la figure V.9 :

- **mapData** : Cet appel permet de filtrer les données à exposer dans elasticsearch. En effet, certaines données sont privées et n'affectent pas la recherche, tel que la liste des contacts d'une entreprise que nous allons ajouter dans une prochaine itération ou la liste des moteurs auxquels appartient l'entreprise. De plus, cette fonction transforme les données du format JSON utilisé par MongoDB à celui accepté par elasticsearch.
- **updateTargetServersList** : Cet appel a pour but d'aboutir à une liste contenant tous les moteurs qui vont être affectés par l'opération de synchronisation. En effet, si une entreprise appartient à plusieurs moteurs à la fois il faut que ses données soient mis à jour sur tous ces moteurs et donc il faut générer une liste de moteurs à mettre à jour en se limitant pour chaque moteur les entreprises qui lui sont associés.

2.1.3 Réalisation

Pour la partie synchronisation, l'implémentation au niveau graphique se résume en une entrée dans un menu qui envoie une requête à la partie backend permettant de synchroniser les données du moteur sélectionné.

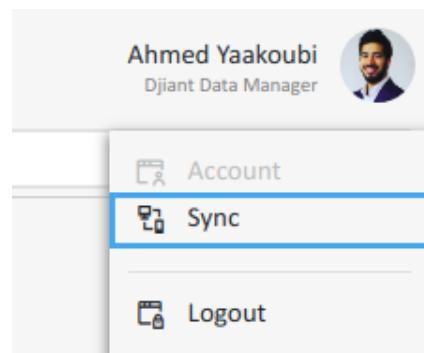


Figure V.10 – Data Factory : Menu

2.2 Iteration 3 : Approche déclarative

2.2.1 Objectif

Dans cette itération de Data Factory, nous allons ajouter la possibilité de collecter les données des entreprises par l'approche déclarative, c'est à dire par l'envoi d'email aux entreprises afin de leur demander de remplir un formulaire avec leurs données.

2.2.2 Conception

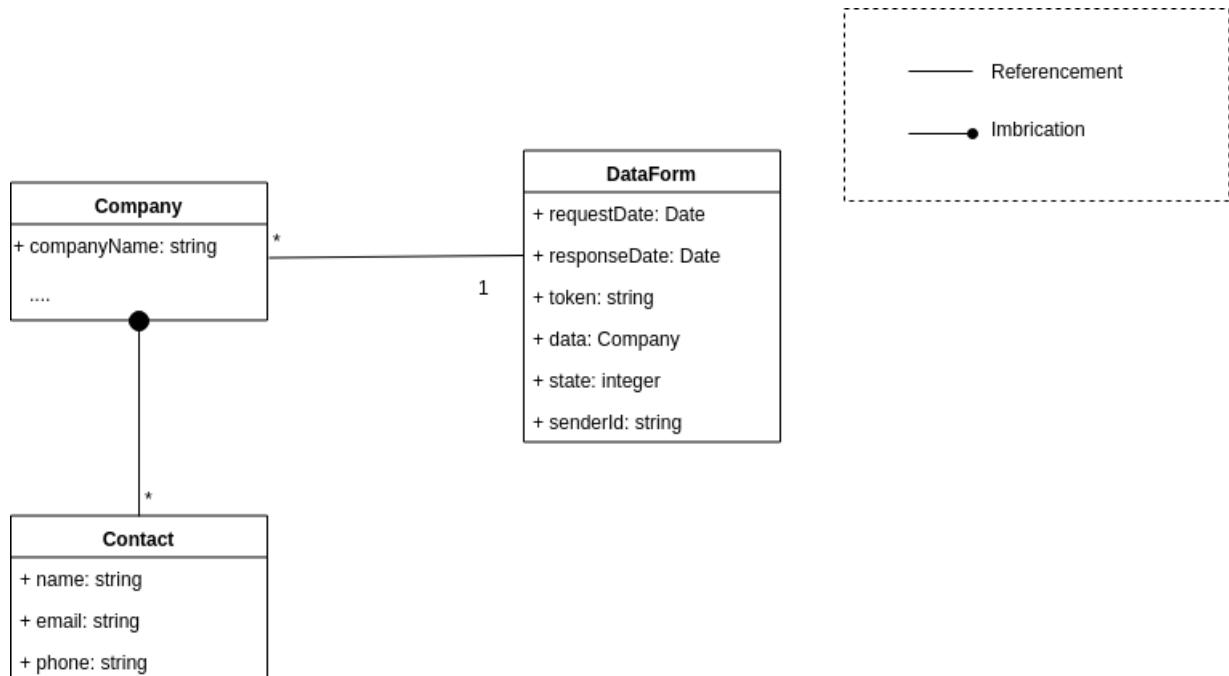


Figure V.11 – Modèle de données

Comme exprimé par la figure V.11, nous avons ajouté deux entités :

- **Contact** : c'est une structure qui contient les informations de base d'un contact chez une entreprise.
- **DataForm** : c'est la représentation du formulaire de données à envoyer dans la base de données.

Elle inclut les données saisies (le champ data de type "Company") ainsi que des données complémentaires tel que la date d'envoi, la date de réponse, etc

De ces données complémentaires, les deux champs les plus importants sont :

- *token* : Il permet à une entreprise d'accéder au formulaire de ces données à travers un lien qui contient un jeton qui lui est unique. Ceci permet ainsi d'identifier les entreprises ainsi que de sécuriser l'accès à leurs données.
- *state* : Il permet de garder une trace de l'état d'avancement du processus de validation des données saisies. Cet attribut peut contenir un de N états possibles. En effet, le formulaire peut être à l'état "Pending" qui est l'état par défaut, ou "Accepted" si l'entreprise accède à l'interface web du formulaire et accepte explicitement la demande, si non l'état est à "Rejected" par l'entreprise dans le cas opposé. Une fois les données sont saisies, l'état passe à "Filled". A ce stade, nous pouvons utiliser Data Factory pour, soit valider les données saisies et donc faire passer l'état à "Validated", soit les rejeter ce qui fait passer l'état à "Ignored".

2.2.3 Réalisation

Dans cette section, nous allons présenter l'implémentation du scénario d'envoi d'email contenant un lien qui redirige vers la page du formulaire, à un contact que nous allons ajouter. Nous commençons, par insérer les coordonnées du contact. Pour ce test nous allons utiliser une adresse mail pour laquelle nous avons un accès.

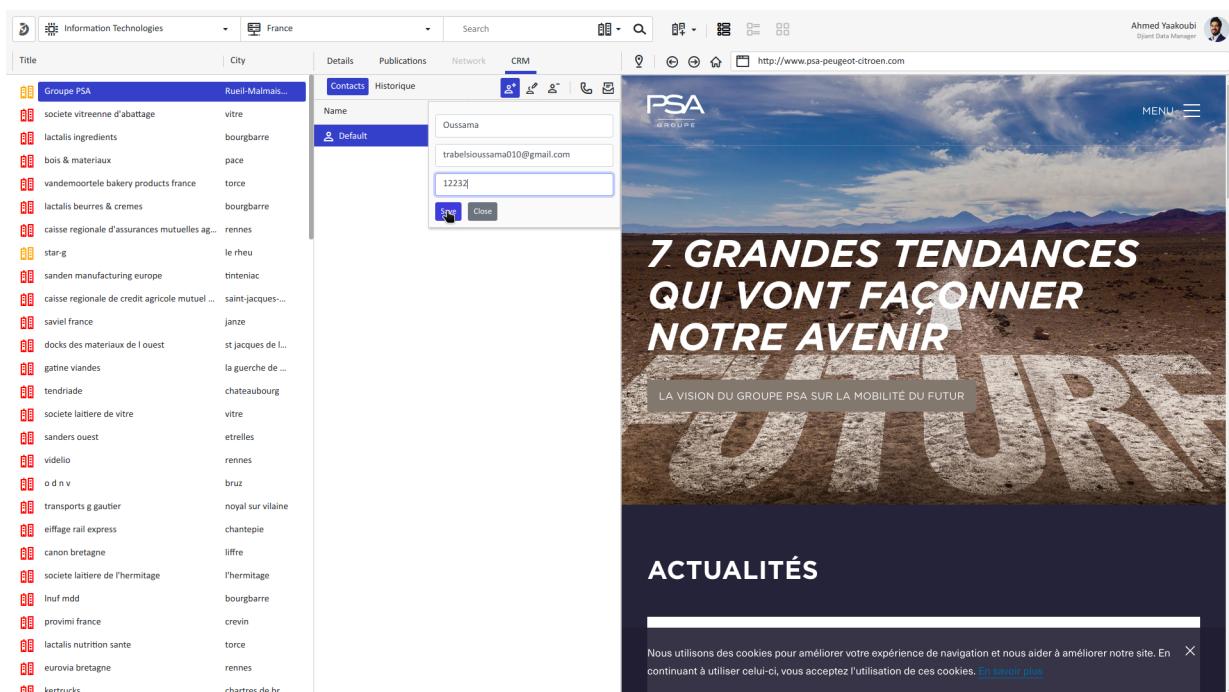


Figure V.12 – Ajouter un contact

V.2 Data Factory

Ensuite, nous envoyons l'email de demande de remplissage de formulaire de données.

The screenshot shows the Data Factory interface. On the left, there is a list of contacts with columns for Title and City. A contact named 'oussama' is selected. On the right, a preview window shows a webpage for PSA Group with a large banner reading '7 GRANDES TENDANCES QUI VONT FAÇONNER NOTRE AVENIR' and 'LA VISION DU GROUPE PSA SUR LA MOBILITÉ DU FUTUR'. Below the banner, there is a section titled 'ACTUALITÉS'.

Figure V.13 – Envoi du mail de l'interface de Data Factory

On peut vérifier si l'email a été bien envoyé

The screenshot shows an email inbox. A test email from 'Djiant Team' is visible, with the subject line 'A test - Hi oussama , Thi is a test email where we test the template function Thanks for you feedback And this is a link to Djiant just for fun.' and the time '2:59 PM'.

Figure V.14 – Vérification de l'envoi du mail

The screenshot shows the content of the test email. It includes the recipient information ('Djiant Team test@djiant.com via spmailit.com to trabelsioussama010'), the date ('Sun, Jun 2, 3:59 PM (6 days ago)'), the message body ('Hi oussama , Thi is a test email where we test the template function [Thanks for you feedback](#) And this is a link to [Djiant](#) just for fun.'), and two buttons at the bottom: 'Reply' and 'Forward'.

Figure V.15 – Vérification du contenu de test du mail

V.3 Insertion de données supplémentaires

Enfin, nous accèdons à l'interface du formulaire afin de saisir les données de l'entreprise.

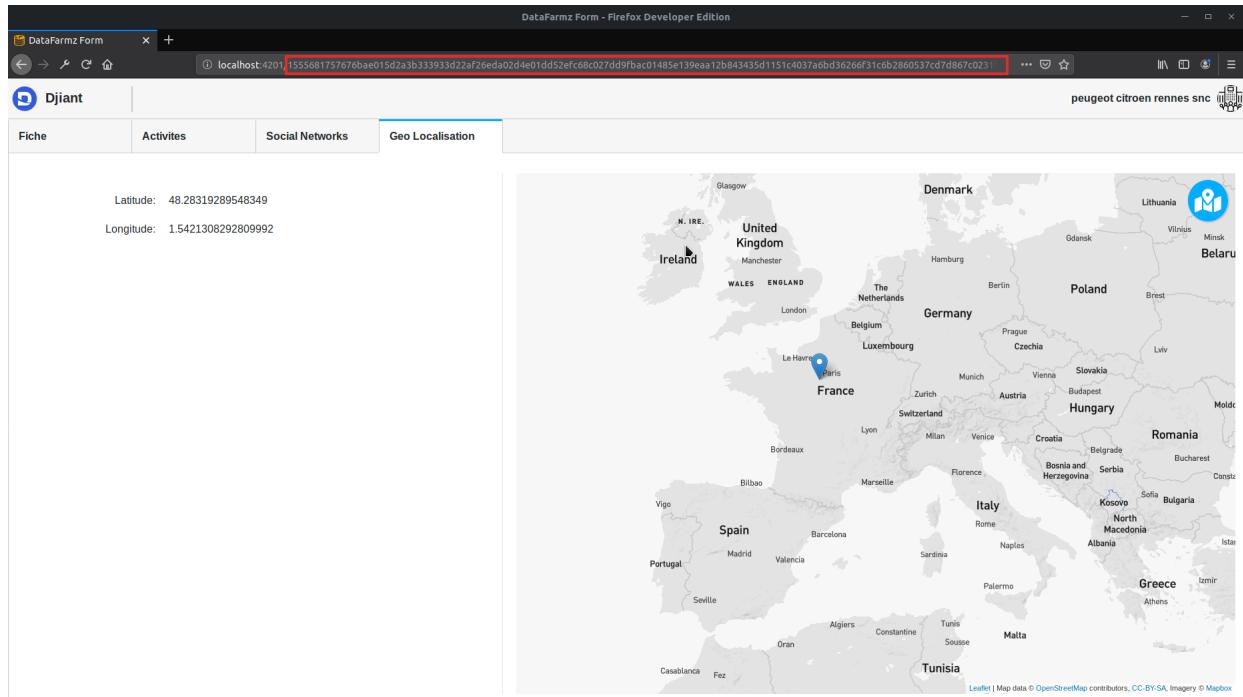


Figure V.16 – Interface du formulaire à remplir

3 Insertion de données supplémentaires

3.1 Description

Après avoir développé le module Explorer, nous avons eu le besoin d'avoir nouveaux types d'objets stockés dans la base Elasticsearch, autres que les entreprises (Actualites, Formations, etc...) afin de faire des tests plus réalistes et des démos plus riches. Nous avons donc effectué l'extraction de ces objets à partir de sites web ou des flux RSS à travers des scripts python que nous avons créé pour automatiser cette tâche lente et répétitive.

4 Evaluation et perspectives

Pour cette release, l'évaluation des nouvelles interfaces créées n'était pas encore faite au moment de la rédaction de ce rapport. Pour la recherche par contre, nous avons remarqué que nous avons une bonne base pour des démos complètes, mais ce n'est certainement pas suffisant pour passer en production avec des recherches complexes par des phrases entières par exemple. Concernant les perspectives, au niveau du module Data Factory, et plus précisément la fonctionnalité de gestion des contacts d'une entreprise, afin de respecter les lois de protection des données personnelles, il est recommandé que l'ajout d'un contact soit effectué sur deux étapes :

- La première consiste à envoyer une demande d'ajout dans la base de Data Factory par mail à la personne en question pour demander sa permission
- La deuxième effectue l'ajout effectif des coordonnées du contact dans la base.

Au niveau du module de recherche "Explorer", il est impératif d'insérer une couche intermédiaire entre la partie frontend et Elasticsearch dont le rôle est d'adapter les requêtes humaines, où les mots-clés sont dilués dans des phrases complètes, aux contraintes d'Elasticsearch afin de récupérer des résultats plus pertinent.

Conclusion

Dans ce chapitre, nous avons présenté les dernières itérations du module Data Factory ainsi que la première version du module de recherche "Explorer", mettant en place ainsi toutes les briques du système précédemment décrit par l'architecture globale du deuxième chapitre. Nous allons donc conclure ce rapport par une conclusion générale et les perspectives que nous proposons pour le projet Djiant en général.

Conclusion Générale et Perspectives

La plateforme créée permet d'enrichir des moteurs de recherche spécialisés par des données collectées par des *Data Managers* et d'y effectuer des recherches par les utilisateurs finaux. Afin d'arriver à ce résultat, nous avons développé cinq modules en se basant sur le principe de séparation des responsabilités :

- Le premier module, *Data Factory*, permet la collecte des données des entreprises.
- Le deuxième module, *Matrix*, permet de gérer les listes des moteurs et de leurs métadonnées.
- Le troisième module, *Crawler*, permet la récupération des pages web de façon efficace et décentralisée afin de les utiliser éventuellement dans des traitements d'extraction d'information.
- Le quatrième module, *Keyword Extractor*, permet d'effectuer des traitements sur les données collectées par le module *Crawler* afin d'extraire des mots-clés qui vont servir à mieux indexer les objets associés aux différents sites web.
- Le dernier module, *Explorer*, représente le front-office de la plateforme. Il permet aux utilisateurs d'effectuer des recherches sur les données collectées.

Afin de respecter les délais et de pouvoir suivre les changements fréquents et imprévus, nous avons utilisé une approche itérative et incrémentale dans laquelle nous avons réalisé trois releases constituées, chacune, de plusieurs itérations dont le nombre varie selon la complexité de ses objectifs.

Tout au long du processus de développement, le travail réalisé a été évalué par des membres de l'équipe ainsi que des parties tierces à travers des démos privées, dans le but de valider les aspects fonctionnels et surtout les aspects ergonomiques de l'application.

Cette évaluation a donné naissance à plusieurs idées d'amélioration dont certaines étaient implementées au cours de ce stage et d'autres gardées en tant que perspectives futures. On peut distinguer deux types de perspectives :

- **Des perspectives à court terme :**
 - Des statistiques de base au niveau du module *Data Factory* qui permettent de mesurer la performance des Data Managers .
 - Externaliser la partie authentification comme un service indépendant qui soit utilisé par les différents modules qui demandent des autorisations d'accès.
- **Des perspectives à long terme :**
 - Intégration de modèles de *machine learning* qui permettent d'extraire des mots-clés à partir des pages web récupérées de façon intelligenteliliasfaxy@gmail.com.
 - Ajout de nouveaux types d'objets dans les modules *Data Factory* et *Explorer*.

Bibliographie

- [1] Meistertask. <https://www.meistertask.com/>. 6
- [2] Architectural principles. <https://docs.microsoft.com/en-us/dotnet/standard/modern-web-apps-azure-architecture/architectural-principles>, (2019). 15
- [3] Performance evaluation of sql and mongodb databases for big e-commerce data. <https://ieeexplore.ieee.org/document/7369245>, (2019). 20
- [4] LUCIANO MAMMINO. *Node.js Design Patterns - Second Edition*. Packt Publishing (2016). 24, 31
- [5] The node.js event loop. <https://nodejs.org/fa/docs/guides/event-loop-timers-and-nexttick/>. 24
- [6] What is mongodb. <https://www.mongodb.com/what-is-mongodb>. 25
- [7] Angular : Architecture overview. <https://angular.io/guide/architecture>. 25
- [8] Introduction to redis. <https://redis.io/topics/introduction>. 27
- [9] 36
- [10] J. JEFFREY HANSON. An introduction to the hadoop distributed file system. <https://www.ibm.com/developerworks/library/wa-introhdfs/index.html>, (2011). 36, 38
- [11] Open source log management. <https://www.elastic.co/solutions/logging>. 37
- [12] About spark. <https://databricks.com/spark/about>. 40
- [13] Apache spark mlib. <https://spark.apache.org/mllib/>. 40
- [14] Rdd programming guide. <https://spark.apache.org/docs/latest/rdd-programming-guide.html>. 40