Charles Huber CNAM GRAND EST
2017/2018

Plan de mémoire

Refonte du moteur de recherche du Parlement Européen

Table des matières

1. Statut et mission de l'auditeur au sein du Parlement Européen

1.1. Les institutions européennes

L'Union Européenne est l'aboutissement d'une évolution communautaire née en 1951 avec la CECA (Communauté Européenne du Charbon et de l'Acier) qui comportait 6 état membres. En 1958, avec l'application du traité de Rome, elle renforce sa coopération économique et nucléaire et devient la CEE (Communauté Économique Européenne). En 1993, après l'adhésion de 6 états membres supplémentaires, le traité de Maastricht en fait finalement une Union Européenne qui étend sa coopération dans les domaines monétaire et politique. L'Union Européenne, dont la mission première a toujours été le maintien de la paix et la sécurité de ses citoyens, assure une coopération communautaire dans les domaines économiques, politiques (intérieure et extérieure) et militaires.[1] Elle compte aujourd'hui 28 états membres.

Ses processus économiques, législatifs et judiciaire sont assurés par 7 institutions[2] :

- *Le Conseil Européen :*

Il est constitué des 28 chefs d'état et de gouvernement. Il détermine les étapes de la construction Européenne et défini, par consensus, sa politique générale.

- *La Commission Européenne :*

Elle est garante du pouvoir Exécutif. Bien qu'elle puisse être sollicitée par le Conseil Européen, par le Parlement Européen et par les citoyens, elle a « le monopole d'initiative ». Elle est la seule a pouvoir proposer des textes de lois, qui sont soumis au Parlement Européen et au Conseil de l'Union Européenne.

- *Le Parlement Européen :*

Il est élu au suffrage Universel direct et représente les citoyens Européens. Dans le cadre des procédures législatives ordinaires, il amende et vote les textes de lois proposés par la commission, conjointement au Conseil de l'Union Européenne. Pour les procédures « spéciales », il a seulement un rôle consultatif. Il vote le budget de l'Union Européenne.

- *Le Conseil de l'Union Européenne[3] (ou Conseil):*

Il est avec le Parlement, le second organe législateur de l'Union Européenne. Composé de ministres des différents états, il représente les gouvernements dans le processus législatif. Il se prononce également sur le budget. Dans le cas des procédures législatives dites spéciales (accords internationaux, adhésion d'un nouveau membre), il est le seul à légiférer.

1 Publications de l'UE : 12 leçons sur l'Europe.
2 Les Institutions Européennes : https://www.touteleurope.eu
3 Le Conseil Européen et le Conseil de l'Union Européenne, ne doivent pas être confondu avec le Conseil de l'Europe, qui n'a pas le statut d'institution Européenne.

- *La Cour de Justice de l'Union Européenne :*

Elle représente le pouvoir judiciaire de la communauté Européenne. Elle a en charge de veiller à la bonne application des lois Européenne et de régler les conflits judiciaires entre les états et/ou les institutions. 28 juges siègent à la cour de justice, soit un par gouvernement.

- *La Banque Centrale Européenne :*

Elle assure la stabilité financière de la zone Euro en contrôlant l'émission de la monnaie unique, en veillant à la stabilité des prix et en définissant la politique monétaire. Elle a également un rôle d'autorité dans le contrôle de la stabilité financière des banques Européennes.

- *La Cour des Comptes :*

Son rôle est de donner son avis sur la gestion des comptes de dépenses et de recettes de l'Union Européenne. Cela concerne le budget des institutions et des organes Européens ainsi que tous les bénéficiaires des aides Européenne. Il s'agit d'un organe de contrôle externe donc elle ne fait aucune gestion financière.

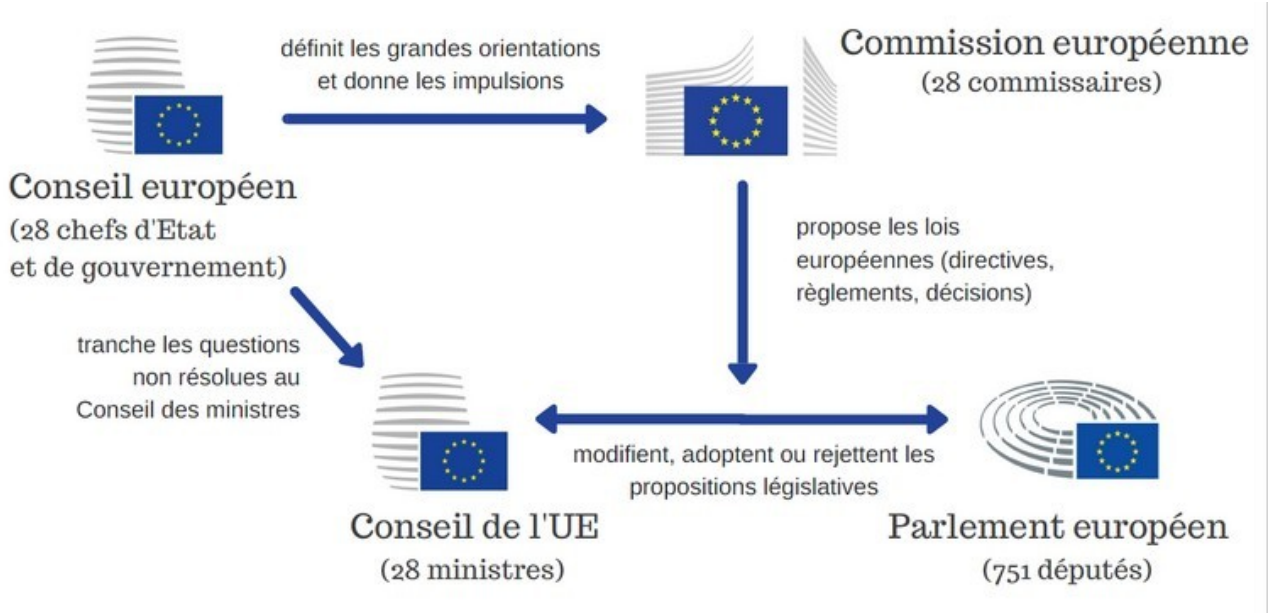


Illustration 1: Résumé des processus inter-institutionnels

1.2. Le Parlement Européen[4]

Le Parlement Européen est composé de plusieurs organes dont nous présentons ici les principaux :

4http://www.europarl.europa.eu/aboutparliament/fr

- *La Présidence*

Le président est élu pour deux ans et demi renouvelable. Il représente le Parlement Européen à l'extérieur. Il signe tous les textes adoptés et rend exécutoire le budget une fois celui-ci voté. Il dirige les sessions plénières et préside le bureau.

- *Le bureau*

Composé du président et de 14 vices-présidents, le bureau est l'organe de direction du Parlement Européen. Il règle toutes les questions administratives, financières et réglementaires internes. Il établit le budget prévisionnel.

- *Les députés*

Les députés sont élus au suffrage universel direct et regroupés en groupes politiques contenant un minimum de vingt cinq individus qui doivent représenter au moins un quart des états membres. Afin d'effectuer le travail préparatoire des séances plénières les députés sont répartis au sein de commissions spécialisées qui ont en charge l'élaboration des propositions de lois. Il y a vingt commissions permanentes auxquelles peuvent s'ajouter des commissions temporaires.

- *La conférence des présidents*

Composée du président du Parlement Européen et des présidents des groupes politiques, elle défini la feuille de route des sessions parlementaires ainsi que la composition et la compétences des délégations. Elle est responsable des relations avec les autres institutions, les parlements nationaux et les pays tiers.

- *Le secrétariat général*

Sous la direction d'un secrétaire général, le secrétariat dirige l'administration du Parlement Européen. Le secrétaire général est le plus haut fonctionnaire du Parlement Européen et favorise la coordination et la coopération entres les directions générales. Les fonctionnaires Européens travaillent au sein des treize directions générales, parmi lesquelles on trouve la direction générale de la communication et la direction générale de l'innovation et du support technologique.

1.3. L'unité « Evolution et maintenance »

« Evolution et Maintenance » est une unité de la Direction Générale de l'Innovation et du support Technologique (DGITEC).

Elle regroupe 42 fonctionnaires qui sont responsables de l'évolution et de la maintenance des applications informatiques développées par le Parlement Européen.

156 consultants externes travaillent en extra ou intra-muros aux développement de ces 216 applications qui totalisent plus de 700 composants.

1.4. La mission de l'auditeur

L'auditeur travaille comme analyste-développeur « senior » au sein de l'unité « Evolution et maintenance » de la DGITEC. Il est mandaté par la société de service en informatique « METHODS S.A », pour réaliser intra-muros, la maintenance et l'évolution de trois applications différentes. Pour chacune de ces applications il est le seul analyste-développeur et remplit les missions suivantes :

- *Étude des besoins et analyse fonctionnelle*

Dans la majorité des cas, les demandes d'évolution pour une application sont formulées de façon triviale par les utilisateurs. Elles peuvent être imprécises, contradictoires, redondantes , irréalisables ou encore mal exprimées. Cet état de faits résulte d'une vision exclusivement concentrée sur le métier, qui occulte la nature technique des applications informatiques. De la même manière, une incompréhension de réel besoin métier par un développeur peut se manifester par des conséquences dramatiques.

L'auditeur a donc pour rôle, au terme d'une longue série d'échanges avec les utilisateurs, de reformuler leur besoins en une énumération de fonctionnalités bien définies. Si l'essentiel de cette tâche se fait en amont, une grande partie d'interrogations concernant les besoins apparaissent durant la phase de conception, ou même durant la phase de tests. L'analyse fonctionnelle est un processus continue visant à supprimer toutes suppositions dans l'implémentation d'une solution par la consultation permanente de l'utilisateur final.

- *Planification des tâches*

Sans rentrer dans une phase de conception détaillée, l'auditeur traduit les fonctionnalités à mettre en place par une liste de tâches concrètes à réaliser en termes d'analyse technique et de développement. Chaque tâche peut-être décomposée en tâches subsidiaires.

Le but de ce travail est de pouvoir faire une estimation des coûts engendrés par l'implémentation d'une solution. Cette estimation est soumise au fonctionnaire responsable de l'application, qui valide ou non la demande d'évolution.

- *Conception, implémentation et tests unitaires*

L'auditeur réalise les taches planifiées. Pour chaque tâche il faut chercher la meilleure solution de développement et la mettre en œuvre. Une fois que toute les tâches de d'implémentation ont été réalisées, le développement des tests unitaires permet de valider la fiabilité du code. Les tests de validation sont réalisés manuellement par l'auditeur et par les utilisateurs sur la plateforme de développement et de pré-production.

- *Rédaction de la documentation technique*

La documentation technique à pour but de faciliter le travail des futurs développeurs qui seront en charge de maintenir et de faire évoluer les applications. Elle explique et justifie les choix techniques qui ont été fait. Que ce soit en termes d'architecture applicative ou en termes de développement.

En fonction de la nature et de l'importance des modifications, la documentation peut prendre plusieurs formes.

La création d'un nouvelle application ou d'un nouveau composant induit la rédaction de documents complets, comme le document technique d'architecture ou les spécifications techniques.

La création ou la modification d'un service web nécessite une documentation spécifique pour les développeurs des applications communicantes. Il s'agit généralement d'une documentation Java insérer dans le code, qui génère une interface documentaire au format HTML à destination des autres développeurs.

Une autre forme de documentation consiste simplement à mettre des commentaires dans son code pour en expliquer le fonctionnement.

Les trois applications pour lesquelles l'auditeur à la charge d'analyste-développeur sont :

- *Le portail d'administration des fiches techniques*

Il s'agit d'une application web développée en Java, qui permet aux utilisateurs de publier et de mettre à jour les fiches techniques de l'Union Européenne sur le site web du Parlement Européen[5]. Hormis la publication sur internet, sa principale fonctionnalité est la validation de la structure des documents publiés. Elle relaie également les fiches techniques vers le Registre documentaire.

- *Le Registre documentaire*

Cette application répond à un règlement Européen qui oblige les institutions à publier leurs documents sur un registre public accessible en ligne[6]. Sa maintenance et ses évolutions représentent l'essentiel du travail de l'auditeur au sein du Parlement Européen. Il s'agit en réalité d'un système applicatif complet qui se compose de sept applications web différentes. Ces applications sont réparties sur le réseau interne et sur la zone démilitarisée (réseau accessible sur internet).

On y trouve, une interface d'administration, un service web de publication pour les applications internes, un module d'exécution de tâches asynchrones, un module d'indexation des documents, un service web de recherche textuelle, un site web publique et un module de gestion des demandes de documents soumises par les citoyens Européen.

Toutes ces applications sont développées en Java.

5http://www.europarl.europa.eu/atyourservice/fr/displayFtu.html
6http://www.europarl.europa.eu/RegistreWeb

- *Le nouveau moteur de recherche du site internet du Parlement Européen*

Le site internet du Parlement Européen propose une interface permettant aux visiteurs d'effectuer des recherches textuelles sur l'entièreté de son contenu HTML.

Jusqu'à aujourd'hui, ce moteur de recherche était mise en place grâce à une solution technique nommée GSA (Google Search Appliance), proposée par la société Google. Cette société ayant annoncé la fin du support de sa solution, l'auditeur a été chargé de redévelopper un moteur de recherche complet. Il s'agit d'un nouveau projet qui constitue l'objet du présent mémoire.

2. La situation initiale

2.1. Présentation et fonction de la GSA

Le moteur de recherche du site internet du Parlement Européen était jusqu'à maintenant supportée par la GSA (Google Search Appliance), qui signifie littéralement Outil de Recherche de Google.

Il s'agit d'une solution matérielle et logicielle, qui se présente sous la forme d'un serveur jaune. Une fois relié au réseau, le serveur propose une interface d'administration et de configuration qui est accessible grâce à un navigateur web.

En définissant un jeu d'URL (Uniform Resource Locator), l'administrateur configurait la GSA pour qu'elle parcours et index le contenu du site internet du Parlement Européen de façon régulière (l'ensemble applicatifs composant le site internet du Parlement Européen est nommé EUROPARL). L'action de parcourir et d'indexer un site internet consiste à visiter chaque page du site en question et à stoker son contenu dans des index optimisés pour la recherche textuelle.

Une fois le contenu indexé, la GSA propose un service web qui permet d'effectuer des recherches sur le contenu du site internet. Lorsqu'un utilisateur effectue une recherche sur le site, sa requête est relayée à la GSA qui retourne la liste des pages HTML où le contenu recherché a été trouvé.

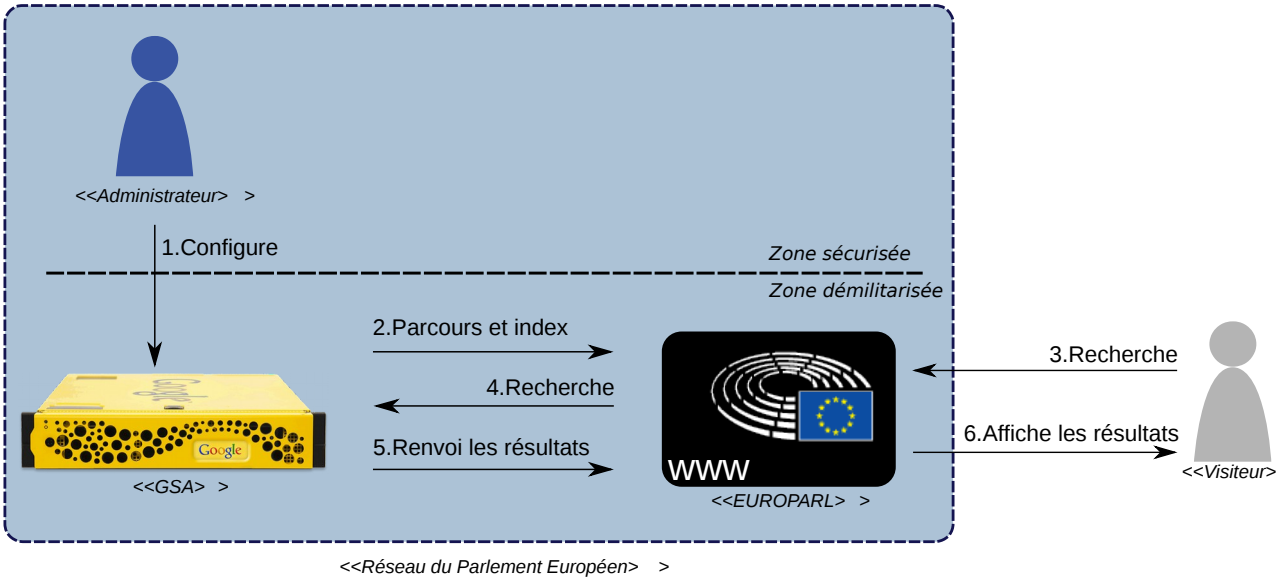


Illustration 2: Présentation de la GSA

2.2. Les lacunes du système

Avant de d'expliquer en quoi la solution de Google ne répond pas suffisamment au besoins fonctionnels, il est important de relever les problèmes plus généraux que présente la GSA.

Le principal défaut relève que l'aspect « tout en un » de la solution n'est pas adaptée à l'organisation des services du Parlement Européen. En effet, comme il s'agit d'une solution matérielle, son administration et sa configuration relève de la responsabilité du service « Opération » de la DGITEC, qui s'occupe de pourvoir aux besoins matériels pour les applications.

Ce qui se traduit par l'impossibilité pour notre service, qui est responsable de l'implémentation fonctionnelle des applications, d'accéder directement à la configuration du moteur de recherche. Toutes modifications doit se faire au travers d'une demande au service concerné. Actuellement, notre service ne contient aucune personne qui sait de façon précise comment la GSA est configurée car l'accès direct à son interface d'administration est cloisonné. Il s'agit d'une boîte noire.

Le second défaut vient du fait que c'est une solution générique et comme toute solution générique elle n'est pas en mesure de répondre à des besoins spécifiques. Ainsi toutes contraintes ou fonctionnalités particulières au site internet du Parlement Européen et à son modèle de données est difficile ou impossible à mettre en place.

Les lacunes fonctionnelles du système qui ont été remontées ne témoignent pas nécessairement d'un défaut du produit mais peut être tout simplement d'une méconnaissance totale de son fonctionnement. Dans la mesure où la GSA était vouée à disparaître et qu'elle ne remplissait que partiellement sa fonction primaire, ni d'éventuelles fonctions secondaires, aucune étude détaillée n'a été faite pour améliorer la situation.

a) Une fonction primaire insatisfaisante

La fonction primaire du moteur de recherche est de permettre de faire des recherches textuelles sur le contenu du site internet du Parlement Européen dans les 24 langues officielles. La GSA remplit cette fonction mais de façon insatisfaisante dans la mesure où les résultats proposés ne sont pas pertinents. Comme tout index « inversé » (nous détaillerons plus tard leur fonctionnement), elle recherche de façon indépendante chaque mot de la phrase recherchée et renvoie toutes les pages qui contiennent au moins l'un d'entre eux.

Par exemple, la recherche « *textes législatifs adoptés* » va retourner toutes les pages qui contiennent au moins l'un des trois mots. Aucune optimisation sur la proximité ou l'association des mots dans les pages retournées n'était configurée et le résultat obtenu fournissait une expérience de recherche décevante.

Il était possible de faire des recherches de phrases en utilisant des guillemets. Cependant il apparaît alors un autre problème. La recherche « ‘‘*textes* ***législatifs*** *adoptés*’’ » ne retournait pas les pages contenant le texte « *textes adoptés* ***législatifs***».

b) Des fonctions secondaires absentes

La configuration de la GSA ne permettait pas,

- de proposer des suggestions de phrases lorsque le texte recherché était mal orthographié,

- de définir la localisation des données à extraire et à indexer dans la structure des pages (les menus étaient ré-indexés dans chaque page),

- de fournir des critères de recherches supplémentaires relatifs à des informations spécifiques contenues dans les pages recherchées,

- de proposer une auto complétion lors de l'écriture des mots recherchés,

- d'indexer des données qui ne sont pas au format HTML.

2.3. Une échéance à relever

La première évaluation témoignant des lacunes du moteur de recherche a été réalisée par la Direction Général de la Communication (DGCOM) en mai 2016. A cette date, il été déjà acté que les licences pour la GSA ne seraient plus renouvelables à cause de l'arrêt du support par Google.

Il a donc été décidé d'attendre le remplacement du moteur de recherche pour intégrer les commentaires de cette évaluation à la nouvelle expression de besoin.

La GSA ne serait plus opérationnelle à partir de août 2018. Le projet de remplacement de la GSA a démarré en novembre 2016.

3. Cahier des charges

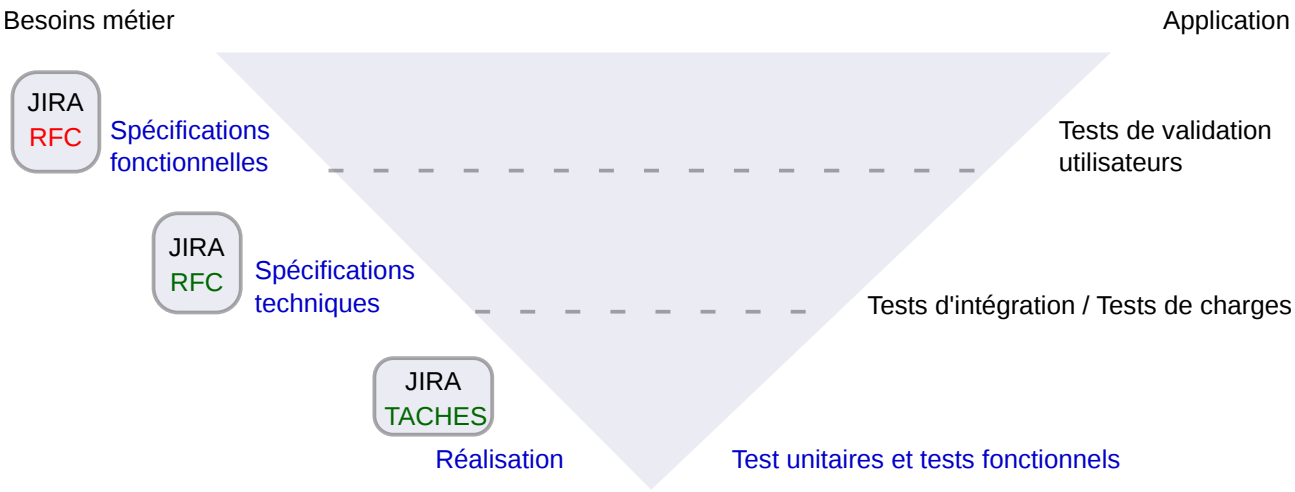
3.1. Le cadre de travail du service « Evolution et Maintenance »

a) Présentation

L'unité « Evolution et Maintenance » a rédigé un document nommé ASTRA (Application Strategy Framework) qui recense l'ensemble des processus participants au cycle de vie d'une application. Chaque processus implique des parties prenantes bien définies, utilise des outils particuliers et transforme des données en entrée vers des données de sortie. Les données sont généralement des documents dans un certain statut, des sources ou des rapports.

Par exemple, « *le responsable d'application* », « *valide* » une « *demande de changement* » (RFC) après avoir considéré « *l'analyse technique* » et « *l'estimation du temps de travail* », en utilisant « *la plateforme de suivi* » JIRA. La dite RFC, une fois validée, servira d'entrée au processus de création des tâches de développement.

Il s'agit d'une vue purement théorique car dans les faits les tâches sont définies (mais pas créées) avant la validation de la demande de changement. Ce sont elles qui permettent de faire l'estimation du temps de travail qui sert à la validation de la RFC.



note *: Les tâches en bleu, incombent à l'auditeur*

Illustration 3: Le modèle en V de l'Assurance Qualité défini par ASTRA

Parmi les processus définis par ASTRA on trouve la rédaction des différents types de documentation, les différentes campagnes de tests à effectuer, la méthode de gestion des versions du code sources, l'intégration continue, les méthodes de déploiement et la formation utilisateur.

L'ensemble de ces processus forme un corpus d'exigences qui ont chacune un degré de recommandation. En bout de chaîne, l'audit de chaque application est réalisé par la validation, ou la non validation, de chacune d'entre elle. Les applications se voient donner une note qui témoigne de leur bonne application du cadre de travail ASTRA.

b)La documentation

- *Les spécifications fonctionnelles*

Elles regroupent les analyses fonctionnelles externes et les analyses fonctionnelles internes.

Les analyses fonctionnelles externes témoignent du besoin de la maîtrise d'ouvrage et font abstraction du détail de l'implémentation.

Les analyses fonctionnelles internes sont rédigées par la maîtrise d'oeuvre et représentent la première phase de conception. Elle décompose les besoins métiers de l'analyse fonctionnelle externe en une série de fonctionnalités élémentaires à implémenter au niveau des composants applicatifs.

- *Les spécifications techniques*

Elles sont sensé détailler les composants techniques utilisés. En pratique, les informations qu'elles doivent contenir sont répartis dans le Document Technique d'Architecture et la documentation technique de l'application.

- *Le Document Technique d'Architecture (DTA)*

C'est un document créer au démarrage d'un projet, qui détaille toutes les machines et applications utilisées ainsi que tous les protocoles de communication établis entre elles. Ce document sert de donnée d'entrée à toute demande d'un nouveau serveur.

- *Le documentation technique de l'application*

Décris les détails techniques de l'implémentation d'une application.

- *Le guide utilisateur*

Ordinairement ce document est rédigé par la maîtrise d'ouvrage. Dans le cas du remplacement du moteur de recherche pour EUROPARL, le service « Evolution et Maintenance » a également le rôle de maîtrise d'ouvrage. Ce document a donc été rédigé par l'auditeur.

Comme préconisé par ASTRA, l'intégralité de la documentation est contenue dans une base documentaire dédiée à l'application.

c) Les tests unitaires

Les tests unitaires sont dit unitaire car chaque test doit validée une fonctionnalité. En terme de développement ont distinguera les tests réalisés pour valider une exigence métier et qui représentent un cas d'utilisation de l'application, des tests réalisés pour valider les fonctionnalités internes implémentées dans le code source. Les premiers seront souvent effectués manuellement, les seconds seront systématiquement automatisés.

d)L'intégration continue

L'intégration continue est un processus de travail qui permet de visualisé directement l'impact du développement d'une fonctionnalité sur l'environnement de développement. Il utilise trois plateformes interconnectées. Un plateforme de pilotage (Jenkins), une plateforme de gestion des versions (SVN) et une plateforme d'analyse de code (SONAR).

Lorsque l'utilisateur enregistre sa version de travail sur SVN, Jenkins télécharge la nouvelle version, la compile et exécute les tests unitaires automatiques. Si il n'y a pas d'erreur la nouvelle version est déployée sur la plateforme de développement. Le code source et les rapports de tests sont ensuite envoyés à Sonar pour analyse.

3.2. Contexte du projet

Le site internet du Parlement Européen est un projet inter-institutionnel qui regroupe plusieurs applications web. Il est désigné par le nom EUROPARL et est accessible à l'adresse « http://www.europarl.europa.eu ».

EUROPARL est divisé en plusieurs « planètes », qui sont chacune un site web indépendant. Chaque planète à son propre système d'information et une URL d'accès unique sous le nom domaine commun « europarl.europa.eu ».

Certaines planètes son maintenues par la DGCOM et d'autres sont sous sous la responsabilité de l'unité « Evolution et Maintenance » . La DGCOM fournie le modèle de style à appliquer pour toutes des planètes afin d'assurer la cohérence visuelle de l'ensemble.

Toutes les planètes sont accessibles dans les 24 langues officielles de la Communauté Européenne. Certaines sont accessibles directement depuis le menu commun à EUROPARL, d'autres le sont par des liens contextuels.

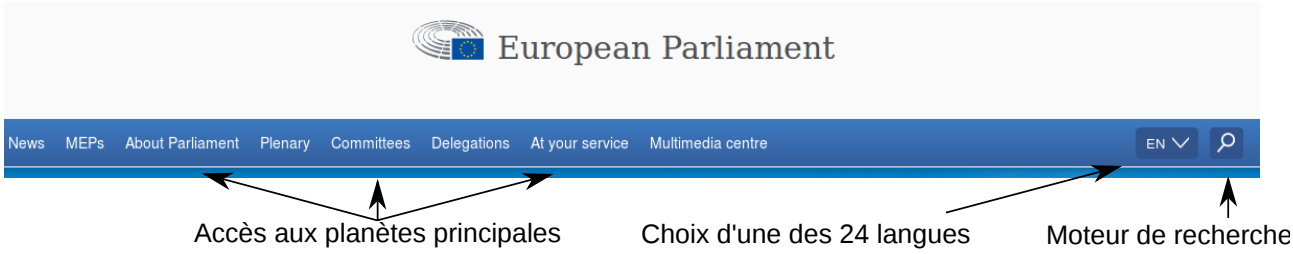


Illustration 4: Menu principal du site EUROPARL

Dans ce document nous allons souvent faire référence aux versions linguistiques des différentes planètes. Par exemple, la version « Anglaise » de la planète « News » et distincte de sa version « Française », qui est différente de la version « Allemande » de la planète « Committees ». Pour y faire référence nous utiliserons désormais le terme « VLP ». Comme il y a 24 langues pour 10 planètes, EUROPARL contient 240 VLPs.

3.3. L'analyse fonctionnelle « externe »

Nous synthétiserons ci-dessous les besoins fonctionnels « externes ». Contrairement à l'analyse fonctionnelle dite « interne », ils font abstraction des détails de l'implémentation et représentent l'exigence de la maîtrise d'ouvrage. Le service « Evolution et Maintenance » ayant à la fois le rôle de maître d'ouvrage et celui de maître d'œuvre, nous détaillerons l'analyse fonctionnelle interne lors de la description de la phase de conception.

Les besoins fonctionnels « externes » seront présentés sous la forme de tableaux fonctionnels avec la légende suivante concernant les critères de flexibilité[7]:

F0 : flexibilité nulle – niveau impératif

F1 : flexibilité faible – niveau peu négociable

F2 : flexibilité moyenne – niveau négociable

F3 : flexibilité forte – niveau très négociable

Les fonctions sont décomposées en fonctions principales (FPx), qui expriment la satisfaction des besoins et en fonctions contraintes (FCx) qui traduisent le contexte de leur réalisation.

a) Les besoins de l'unité « Evolution et Maintenance »

Fonction de service	Flexibilité
FP1 : Proposer à l'utilisateur une recherche textuelle sur la VLP courante qui fournissent des résultats pertinents.	F0

7 Norme AFNOR NF X50-151

FP2 : Proposer à l'utilisateur des résultats de recherche qui surligne le texte recherché.	F1
FP3 : Proposer à l'utilisateur des résultats de recherche paginés et triés par ordre de pertinence (les résultats sont proposés sur plusieurs pages. L'utilisateur choisi la page qu'il veut consulter).	F1
FP4 : Proposer à l'utilisateur de trier ses résultats par date de publication.	F2
FC1 : Permettre d'étendre une recherche à toutes les VLP de la langue sélectionnée.	F0
FC2 : Configurer les fréquences d'indexation du moteur de recherche pour chaque VLP.	F0
FC3 : Rajouter, mettre à jour ou supprimer une VLP dans le moteur de recherche.	F0
FC4 : Supprimer des résultats de recherche possibles, les pages qui ne sont plus publiées sur EUROPARL.	F0
FC5 : Configurer une politique d'indexation en fonction de la date de publication d'une page HTML (les pages qui changent ne change pas n'ont pas besoin d'être ré-indexées régulièrement).	F1
FC6 : Configurer pour chaque VLP, l'endroit des données à indexer dans les pages parcourues (par exemple, changer la localisation de la date de publication dans les pages).	F1

b) Les besoins spécifiques de la Direction Général de la Communication

Fonction de service	**Flexibilité**
FP5 : Proposer à l'utilisateur une recherche non textuelle sur les catégories des pages. Les pages d'une même VLP pouvant appartenir à des catégories différentes. (Par exemple, l'utilisateur recherche toutes les pages appartenant à la catégorie « Agriculture, Écologie »)	F0
FP6 : Proposer à l'utilisateur une recherche non textuelle sur les mots clés des pages.	F0
FP7 : Proposer à l'utilisateur des exemples de recherches pertinents, en cas de recherche mal orthographiée.	F1
FP8 : Proposer une auto-complétion du texte à l'utilisateur qui écrit sa recherche textuelle	F1
FP9 : Intégrer la popularité d'un résultat de recherche dans le calcul de sa pertinence.	F1

3.4. Les contraintes techniques

a) Les technologies supportées et autorisées

- **Les systèmes applicatifs**

Le service « Opération » impose que les applications soient des serveurs Tomcat 8 déployées sur des systèmes d'exploitation Linux Redhat.

Si besoin, les technologies d'indexation disponible sont des serveurs Elasticsearch.

- **L'architecture applicative**

Il est recommandé d'utiliser la librairie java « Spring » pour implémenter un environnement de développement intégrant des fonctionnalités JEE (Java Enterprise Edition).

Le logiciel Maven est obligatoire pour gérer l'intégralité des librairies externes et internes qui sont utilisées par les applications.

Le logiciel NPM (Node Package Management), qui permet de bénéficier de librairies Javascript pour créer des interfaces graphiques basées sur des technologies récentes, n'est pas encore autorisé.

b)Un projet pilote vers le Cloud

Compte tenu de la difficulté que le service « Opération » a à pourvoir l'intégralité des machines qui lui sont demandées, l'idée d'un déploiement de certaines applications sur les plateformes Cloud n'était pas nouvelle, mais n'avait encore jamais été expérimentée.

Le moteur de recherche pour EUROPARL ne manipule que des données publiques et nécessite au moins deux machines distinctes. Une hébergeant l'application parcourant le site internet, l'autre hébergeant les indexes. A ce titre il fut désigné candidat idéal à l'expérimentation d'un déploiement sur le Cloud.

Toutefois, la possibilité de la mise place d'une passerelle sécurisée (VPN), entre le réseau du Parlement Européen et un éventuel fournisseur de technologies Cloud est pour le moment proscrite. Cela a pour conséquence que toute connexion à des machines (virtuelles ou non) situées sur le Cloud est impossible. Le type de service IAAS[8] (Infrastructure As A Service) n'est donc pas une solution envisageable. La solution devra être implémentée à l'aide de service PAAS[9] (Plateform As A Service) ou SAAS[10] (Software As A Service).

4. La gestion du projet

D'un point de vue budgétaire il y a eu en réalité deux projets.

Le premier (PRJ-999) a été lancé en novembre 2016 et consistait à choisir un fournisseur pour les technologies Cloud.

8 IAAS : Fournit des machines munies uniquement d'un système d'exploitation et qu'il faut entièrement configurer.
9 PAAS : Fournit une plateforme web à partir de laquelle on peut déployer son application. L'utilisateur a peu de marge de manœuvre sur l'infrastructure. Le développement des applications est contraint par la plateforme.
10 SAAS : Fournit un logiciel dédié à l'utilisateur, que celui ci peut utiliser à distance.

Le second projet (PRJ-1855) a démarré en juin 2017, une fois le contrat avec le prestataire signé, et consistait à réaliser la solution.

4.1. Budget

Pour PRJ-999 un budget de 10 jh (jour * homme) était prévu.

Du début de la réalisation en juin 2017 à l'échéance en août 2018, PRJ-1855 devait occuper 75 % du temps de l'auditeur. Ce qui correspond à un budget de 200 jh.

4.2. Plannings prévisionnels

PRJ-999 – *Comparatif des solutions Cloud*	
Études des fonctionnalités couvertes par la GSA. *Analyse fonctionnelle externe du service « Evolution et Maintenance ». La future solution ne devant pas apporter de régression.*	1jh
Recherche des solutions d'indexation sur le Cloud. *Recherche et études des solutions proposées*	2jh
Réalisation d'un POC[11] générique. *Le prototype devait parcourir des pages d'EUROPARL pour les indexer dans une des solutions d'indexation. Il devait proposer une interface de recherche afin de comparer les résultats obtenus.*	3j
Adaptation du POC aux différentes solutions	3j
Rapport comparatif et choix de la solution	1j
	10j

PRJ-1855 – *Conception et réalisation de la solution*	
Analyse fonctionnelle interne *Décomposition des fonctionnalités métier en fonctionnalités techniques élémentaires, ou plus simplement, chercher les solutions d'implémentation.*	20j
Mise en place de l'intégration continue *Préparation des environnements de développement*	1j
Réalisation de l'application d'indexation *Implémentation des solutions trouvées*	50j
Réalisation de l'application exposant les services web de recherche *Implémentation des solutions trouvées*	10j
Développement d'une librairie cliente pour interroger les services web *Définition de l'interface applicative de recherche en accords avec les cas d'utilisation*	2j

11 POC ou P roof Of Concept. Littéralement « preuve du concept », le terme est utilisé pour désigner un prototype sommaire, réalisé dans le but de valider une solution informatique.

PRJ-1855 – *Conception et réalisation de la solution*	
Intégration de la librairie cliente à EUROPARL *La librairie permet à EUROPARL de relayer les recherches au service web.* *(La tâche sera confiée à un développeur d'EUROPARL mais est intégrée au budget dédié à PRJ-1855)*	3j
Développement des tests unitaires automatiques pour les deux applications (application d'indexation et services web)	60j
Rédaction de la documentation utilisateur pour les services web *Documentation à l'intention des développeurs des système clients.* *La documentation est publiée avec les services web.*	2j
Rédaction de la documentation technique pour les deux applications (application d'indexation et services web) *Détails et justification des choix d'implémentation.*	25j
Rédaction de la documentation utilisateur pour les administrateurs *Explication des manipulations à faire pour réaliser les cas d'utilisation de l'administrateur.*	1j
Formation utilisateur pour les administrateurs *Démonstration et explication de la documentation utilisateur*	1j
Tâches transverses de préparation et de déploiement des livrables	2j
	177j

5. La conception du projet

5.1. Architecture technique prévisionnelle

Avant d'avoir choisi, ni même testé les différentes plateformes de Cloud, le squelette d'une architecture de base pouvait être dressé grâce a des contraintes techniques et a des choix évidents.

Le premier des choix était de sortir du modèle « Tout en Un » proposé par la GSA. Il fallait séparer la fonctionnalité responsable de parcourir et d'indexer EUROPARL, de la fonctionnalité responsable de fournir les réponses aux requêtes de recherche.

Non seulement il s'agit de deux fonctionnalités tout à fait différentes, qui doivent être logiquement séparées, mais surtout, une dégradation de l'une ne devait en aucun cas induire la dégradation de l'autre. La solution logicielle et matérielle unique de la GSA serait donc décomposée en trois applications différentes, indépendantes de leur support matériel ; un indexeur, un service web et une solution d'indexation.

Il ne faut pas confondre l'indexeur et la solution d'indexation. L'indexeur parcours EUROPARL pour en extraire les données. Il les envoie à la solution d'indexation qui est responsable de leur stockage. Afin de faciliter les déploiements et pour pousser l'expérimentation, nous avons fait le choix de déployer un maximum de composants sur le Cloud . Il s'est avéré que l'indexeur ne

pourrait pas en faire partie car sa plateforme de développement, en étant à l'extérieur du réseau du Parlement Européen, ne pourrait pas parcourir la plateforme de développement d'EUROPARL. Avant même de faire l'évaluation des différentes solution d'indexation, nous pouvions élaborer un architecture technique prévisionnelle.

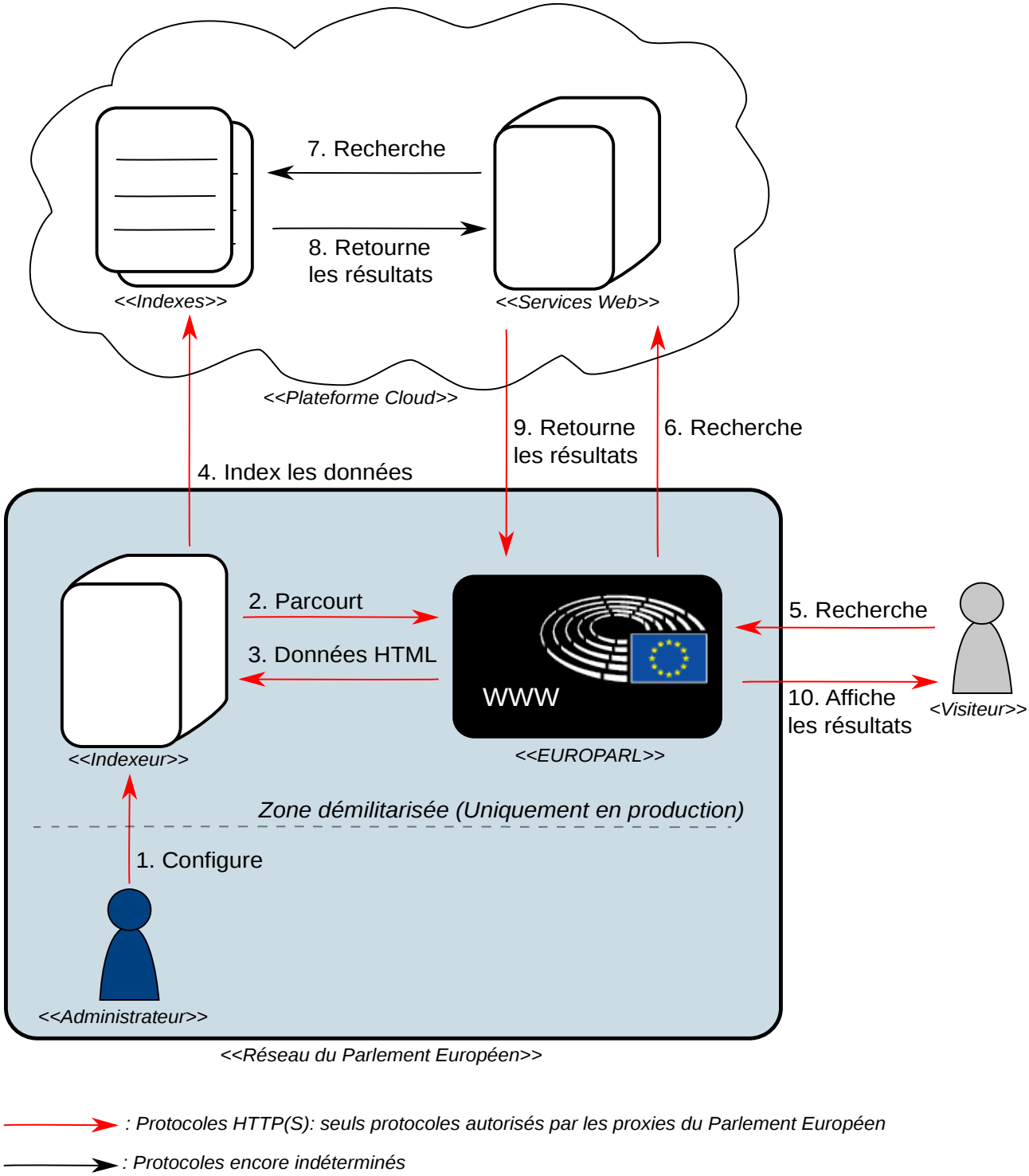


Illustration 5: Architecture prévisionnelle

5.2. État de l'art des solutions techniques applicables

Bien que l'application exposant les services Web soit déployée sur le Cloud, les plateformes de développement sont celles du Parlement Européen et à ce titre les technologies utilisable sont celles définies par le cahier des charges.

Les deux applications, l'Indexeur et le services Web, doivent donc être nécessairement des applications Web développée avec le langage Java 8 est déployée sur un serveur applicatif Tomcat 8. Sans douter qu'une étude comparative entre les différentes technologies de serveurs applicatifs et d'applications web puissent être enrichissante, je me concentrerai uniquement sur l'étude des solutions applicables aux choix techniques restants.

Cela comprend le choix du mode de communication entre EUROPARL et les services Web, et la construction de l'interface graphique des deux applications Java ainsi que leur mode de communication avec le serveur.

Pour finir j'expliquerai le principe de la technologie employée par l'intégralité des solutions d'indexation.

a) Les services Web

L'application hébergeant les services Web a pour rôle de fournir une interface de recherche centralisée, sécurisée et simple d'utilisation. Les services de recherche qu'elle propose doivent refléter les besoins métiers mis à jour par l'analyse fonctionnelle externe et cacher la complexité technique du système d'indexation.

Le choix se pose de la technologie de communication a mettre en place entre l'application Java hébergeant les services Web et les différentes applications Java qui composent EUROPARL. Les différents options sont :

- *l'accès direct aux services,*

Il s'agit d'une technologie permettant à une application cliente d'utiliser les composants de l'application distante, comme si il s'agissait de composant locaux. L'architecture sur laquelle repose cette technologie se nomme CORBA (Common Object Request Broker Architecture) et permet à des applications de technologies différentes d'inter-opérer entre elles. La mise en place de cette architecture nécessite un déploiement compliqué car elle nécessite de publier une définition très détaillée des services et de préparer des « squelettes » permettant d'y accéder de façon distante.

Java en propose une version simplifiée nommée RMI (Remote Method Invocation) qui permet d'établir une communication distante entre des composants JAVA. La plus célèbre implémentation de RMI sont les composants EJB (Enterprise Java Bean) de la plateforme JEE (Java Enterprise Edition).

Le problème commun à ces deux solutions est la nécessité de communiquer sur un port particulier. Cela impliquerait forcément de rapatrier les services Web sur le réseau du Parlement Européen. RMI limiterait les potentiels clients aux applications codées en Java, ce qui pourrait posé un

problème si il fallait fournir le service à d'autres institutions. CORBA a le désavantage d'être beaucoup trop complexe à mettre en place autant du point de vue serveur que du point de vue du client.

Ces deux technologies sont tombées en désuétude fasse à la simplicité et à la flexibilité des communications distantes basées sur le protocole HTTP (Hyper Text Transfer Protocol).

- *les services SOAP,*

Les services Web basés sur SOAP (*Simple Object Access Protocol*) résolvent tous les problèmes posés par CORBA ou RMI. Il s'agit d'un protocole applicatif reposant sur le protocole HTTP. Ce qui implique qu'il est accessible sur un port communément utilisé et que des applications de technologie très différentes peuvent communiquer entre elles.

Le principe consiste à publier sur une URL donnée une définition des services au format XML (Extensible Markup Language). La définition d'un service consiste à lister les fonctions disponibles ainsi que la structure des données qu'elles acceptent en entrée et retournent en résultats. Les messages envoyés aux services sont également au format XML, ils contiennent le nom de la fonction invoquée ainsi que les données attendues en paramètres. Le service répond avec un message, toujours en XML, qui contient les résultats.

Comme CORBA ou RMI, cette solution amène un couplage très fort entre le serveur et le client. Selon les cas cet aspect peut être un avantage ou un inconvénient. Un couplage faible apporte de la flexibilité, un couplage fort apporte de l'intégrité.

Le plus gros désavantage est que les messages sont au format XML. Il s'agit d'un format très verbeux, si bien que les données utiles d'un message peuvent représenter une infime partie du message total. Ce qui en fait un protocole très peu optimisé en terme de charge réseau car il véhicule énormément de données inutiles.

- *les services REST*

Comme les services SOAP, les services REST (Representational State Transfert) utilisent le protocole HTTP. Ils sont simples d'accès et universel. Leur particularité est qu'ils utilisent le protocole HTTP sans couche protocolaire supplémentaire.

L'idée est d'utiliser les quatre types de requêtes proposés par le protocole HTTP pour effectuer les quatre actions élémentaires sur une ressource :

- GET : permet de récupérer une resource
- DELETE : permet de supprimer une ressource
- PUT : permet d'insérer ou de mettre à jour une ressource (la ressource a déjà un identifiant)

- POST : permet de soumettre une nouvelle ressource (la ressource n'a pas d'identifiant au moment de la soumission)

Chaque type de ressource a une URL dédiée, qui peut contenir ou un identifiant dans le cas des méthodes GET et DELETE.

Pour les méthodes PUT et POST, les ressources sont directement contenues dans le corps de la requête. Ces services ont l'énorme avantage de ne pas contraindre le format du message. Dans la majorité des cas on véhiculera directement des objets au format JSON (JavaScript Object Notation), car contrairement au XML, elles sont très légères et directement interprétable par les interfaces Web écrite en Javascript.

Exemples commentés d'actions REST :

//Crée un nouveau chat, le service renvoie la ressource créée en acquittement

POST http://monservice/chat {"id" : null, "nom" :"minet", "age" :2}

=> {"id" : **3**, "nom" :"minet", "age" :2}

//Modifie le nom du chat

PUT http://monservice/chat {"id" : 3, "nom" :"minou", "age" :2}

=> {"id" : **3**, "nom" :"minou", "age" :2}

//Récupère le premier chat

GET http://monservice/chat/1

=> {"id" : **1**, "nom" :"cerise", "age" :9}

//Supprime le premier chat

DELETE http://monservice/chat/1

=> {"id" : **1**, "nom" :"cerise", "age" :9}

//Récupère tous les chats

GET http://monservice/chat

=> {"id" : **2**, "nom" :"marco", "age" :5},{"id" : **3**, "nom" :"minou", "age" :2}

//Ajoute un jouet au chat 2

POST http://monservice/chat/2/jouet {"id" : null, "nom" :"ballon"}

=> {"id" : **2**, "nom" :"ballon"}

//Récupère tous les jouets du chat 2

GET http://monservice/chat/2/jouet

=> {“id” : **1**, “nom” :“souris”}, {“id” : **2**, “nom” :“ballon”}

Ces services sont extrêmement souples et très faciles à intégrer. Leur inconvénient majeur est qu’il est impossible de fournir une interface bien définies car aucune définition n’est publiée[12]. Contrairement au service SOAP, il n’est pas possible de générer automatiquement les composants Java à partir d’une définition claire du service. Ce manque d’intégrité implique de rédiger et de maintenir une documentation très détaillée sur la structure de données échangées et de bien identifier tous les clients pouvant être impactés par une modification.

- *Conclusion*

L’auditeur a fait le choix d’utiliser les services REST car ils sont faciles à mettre en place, optimisés pour les transferts et renvoient des données directement exploitable par les interfaces Web. Afin d’exposer une définition claire des fonctions disponibles et des données échangées, l’auditeur fournira une librairie Java aux applications d’EUROPARL. La librairie exposera des interfaces correspondant aux besoins métiers et sera en charge de mettre en forme les appels REST. Elle remplacera les composants Java qui sont habituellement générés automatiquement à partir de la définition d’un service SOAP. Cette solution devrait coupler l’intégrité des services SOAP avec la souplesse des services REST.

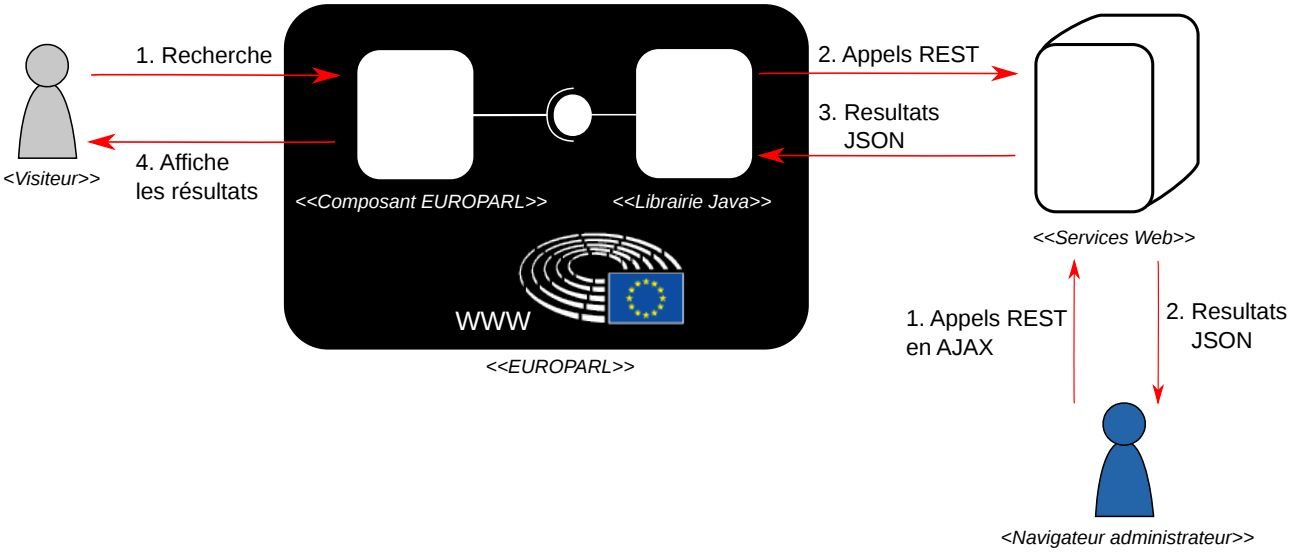


Illustration 6: Solution retenue pour les services Web

L’application hébergeant les services Web pourra également proposer aux administrateur une interface de recherche graphique au format HTML. Les composants pourront directement adresser les services web exposés grâce a des appels AJAX (Asynchronous Javascript And XML) et intégrer

12 Les tentatives de standard pour la définition des services REST sont marginales et mal supportée (WADL)

leur résultats JSON à la page HTML Les administrateurs auront accès aux fonctionnalités de recherche sans aucune couche applicative intermédiaire.

Cette séparation complète entre l'interface graphique et les services exposés par le serveur va a l'encontre des architectures habituellement rencontré dans les applications Web JEE.

b)Les interfaces graphiques

- *Spring MVC et les JSP :*

Les JSF (Java Server Face) et JSP (Java Server Page) sont les composants standards des applications Web écrites en Java pour générer des pages ou des composants HTML. L'étude des JSF sort du cadre de ce chapitre car elles sont une technologie des plateformes pleinement JEE et à ce titre elles ne font pas partie de l'environnement de développement proposé par le Parlement Européen.

Les JSP sont concrètement des fichiers au format XML, qui contiennent à la fois des balises HTML et des balises interprétables par le serveur d'application et défini par la norme JSTL (Java Standard Tag Library).

Cette combinaison de balises HTML et de balises interprétables, permet au serveur de générer un contenu HTML qui dépend du contenu des données à afficher.

Considérons l'exemple extrait du site pédagogique de Jean- Michel Doudoux [13] :

```
<TABLE>
     <c:forEach var="personne" items="${listeDePersonnes}">
     <TR>
          <td><c:out value="${personne.id}" /></td>
          <td><c:out value="${personne.nom}" /></td>
          <td><c:out value="${personne.prenom}" /></td>
     </TR>
     </c:forEach>
</TABLE>
```

Le serveur va utiliser la donnée « listeDePersonnes » contenue dans la session de l'utilisateur pour afficher un tableau contenant la liste des personnes.

Dans ce cas, la solution la plus triviale consisterait à utiliser le composant Java standard responsable de la réception des requêtes HTTP (une Servlet), qui irait chercher la liste des personnes dans la base de données pour la transmettre à la JSP responsable de la génération du HTML pour l'utilisateur.

13 JSTL: https://www.jmdoudoux.fr/java/dej/chap-jstl.htm – Jean Michel Doudoux est un formateur Java influent et a été nommé « Champion Java » par la communauté de développeurs.

Cette façon simple de faire ne respecte pas le concept d'architecture de base qui consiste à séparer le composant responsable des données (le Modèle), du composant responsable de l'affichage (la Vue). Il est conseiller d'interfacer un composant responsable de faire le lien entre les données et leur affichage (le Contrôleur). C'est ce que l'on appel l'architecture MVC (Modèle Vue Contrôleur).

Pour ce faire la librairie la plus populaire est nommé Spring MVC. Elle est basée sur la librairie Spring[14] et propose des composants clé en main pour mettre en place une architecture MVC facilement.

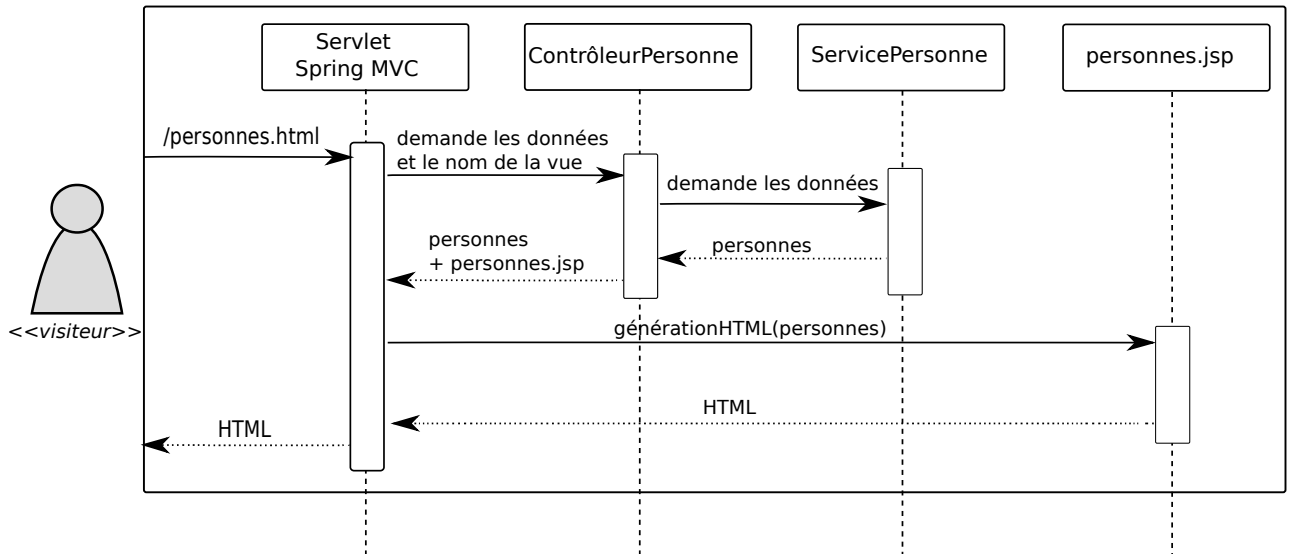


Illustration 7: Simplification d'une séquence Spring MVC

Cette architecture, bien que basée sur un le modèle MVC, présente plusieurs désavantages :

- le contenu HTML renvoyé est statique et propose une expérience de navigation décevante.
- Le serveur est responsable de la génération de l'interface graphique et du traitement des données. Le concept de l'architecture MVC n'est donc pas pleinement respecté.
- Le serveur consacre de la puissance de calcul à la génération d'interfaces graphiques pour des clients qui ont les ressources pour le faire eux-même.

Cette façon de faire, bien que décevante, est encore très largement répandue. Elle s'explique par le fait que l'on peux réaliser une application Web uniquement avec des développeurs Java ayant des connaissances limitées dans technologies Web.

14 http://redmonk.com/fryan/2017/06/22/language-framework-popularity-a-look-at-java-june-2017/ - Comme montré dans cet article, Spring est le framework le plus populaire dans la communauté Java. Il s'agit avant tout d'un framework d'injection de dépendance, qui fournit en plus la majorité des API définie par les normes JEE. Il peut être utilisé dans n'importe quel type d'application Java. C'est le concurrent directe des plateformes pleinement JEE.

La réalisation d'interfaces graphiques dynamiques, qui réquisitionnent la puissance de calcule du client plutôt que celle du serveur, requiert des connaissances avancées en Javascript.

- *Javascript et SPA :*

De nos jours les interfaces graphiques se doivent d'être réactives ; de fournir des informations à l'utilisateur de façon continue et sans intervention de sa part.

Elles sont réalisées en Javascript et présentent l'avantage d'être exécutées par le poste client sans mobiliser les ressources du serveur. Ces interfaces deviennent complexes à développer et nécessitent une méthode de développement bien structurée. La pratique du langage Javascript est sortie de cadre du bricolage pour rentrer dans un cadre industriel.

Avec le framework AngularJS créé en 2009 par Google[15], l'architecture MVC a été déplacée du serveur vers le client. C'est un composant Javascript qui demande les données au serveur grâce a des appels AJAX et qui génère le HTML associé. La même année, l'apparition de Javascript dans l'écriture de programmes serveur avec NodeJS, a vu l'apparition de la programmation modulaire en Javascript et celle de l'outil de gestion NPM (Node Package Management) qui permet de gérer les dépendances aux librairies externes.

Bien que tous ces outils ne soient pas obligatoires pour faire de la programmation modulaire basée sur le modèle MVC en Javascript, ils l'ont propulsés comme langage de premier plan et ont participé grandement l'expansion des services REST, qui sont directement intégrables dans ces interfaces.

Aujourd'hui, le framework Angular (différent de AngularJS) créé par Google, permet de réaliser des applications complètes sur une seule page HTML. La page ne contient en réalité que les scripts a exécuter. Le reste n'est qu'une succession d'appels au services exposés par le serveur et de génération dynamique des structures HTML. C'est ce que l'on appel des SPA (Single Page Application).

- *Conclusion*

Pour les deux applications développées par l'auditeur, le choix a été fait de réaliser des SPA basées sur une programmation Javascript modulaire et reposant sur l'architecture MVC.

Pour l'application de services Web, l'interface de recherche a l'avantage de tester directement les services utilisés par les applications d'EUROPARL.

Pour l'interface d'administration de l'Indexeur, le principal avantage est de pouvoir visualiser de façon dynamique la progression des tâches d'indexation.

Comme l'outil NPM permettant d'utiliser des librairies externes n'est pas compris dans les environnements de développements, aucun framework n'a été utilisé.

15 https://angularjs.org/

Les classes et les modules ont été crées avec les fonctionnalités natives du langage Javascript dans sa version ES5.

c) Les systèmes d'indexation

Pour la majorité des applications développées au Parlement Européen, les données qu'elles utilisent sont enregistrées dans une base de données relationnelle. Si nous appliquions cette solution au moteur de recherche, cela consisterait à créer une table « PAGE_HTML », avec une propriété nommée « ID » qui servirait à stocker un identifiant unique et une propriété une nommée « CONTENU » servant à stocker le texte qu'elles contiennent. Rechercher toutes les pages contenant le texte « documents parlementaires », reviendrait à parcourir toutes les entrées de la table « PAGE_HTML » et à analyser le texte de chacune d'elle pour y trouver le texte recherché. Cette solution seraient énormément consommatrice de ressource et ne serait pas du tout optimisée.

Toutes les solutions permettant de faciliter les recherches sur un contenu textuel sont basées sur ce que l'on appel un index inversé. Au lieu d'avoir un identifiant référençant un document, chaque mot d'un dictionnaire référence l'ensemble des documents qui le contiennent. La recherche sur un mot ou sur un ensemble de mots renvoie aussitôt les documents concernés. A titre d'exemple, considérons un index vide dans lequel nous voudrions indexer trois documents.

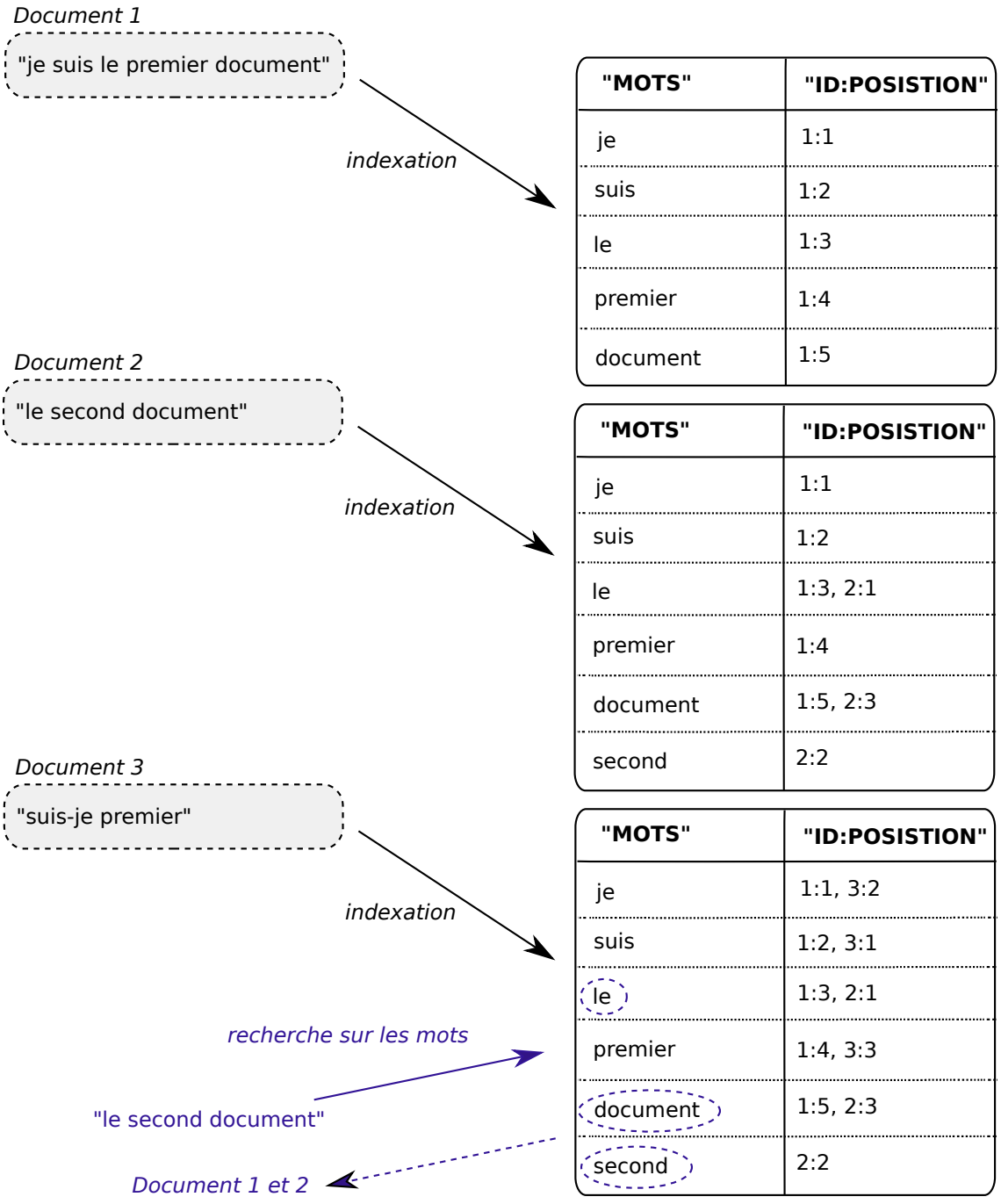


Illustration 8: Principe de fonctionnement des indexes inversés

Nous remarquons comment chaque mot devient l'identifiant des documents qui le contiennent. Dans cette exemple nous effectuons une recherche sur les mots, le résultat contient donc tous les documents qui contiennent au moins l'un d'entre eux. Grâce aux informations de position, nous

pourrions effectuer une recherche sur la phrase complète « *le second document* », qui rechercherait tous les documents contenants les trois mots de façon consécutive.

Il s'agit ici d'un exemple de principe. Cette façon d'indexer chaque mot sans modification sera toujours utilisée car elle permet de faire des recherches sur les phrases. En revanche, il y a plusieurs optimisations qui sont réaliser pour la recherche plus classique sur les mots. Ainsi, chaque champ textuel est indexé de plusieurs façons. Il subit une indexation brute pour la recherche de phrase et une indexation optimisée pour la recherche sur les mots. Il y a plusieurs type d'optimisation dont les principaux sont décrits ci-dessous.

- **Les « stopwords »**

Par exemple, la recherche « *le premier document* » va retourner tous les documents contenant le mot « *le* » ; ce qui concernera la presque entièreté de l'index. Certains mots doivent donc être ignorés, à la fois à l'indexation et pour la recherche. C'est ce que l'on appel les « *stopwords* » et ils dépendent de la langue d'indexation.

- **Stemming[16] et Lemmatisation[17]**

Une autre optimisation consiste à regrouper les mots par famille linguistique. Ainsi les mots « *premier* » et « *première* » seront indexés en tant que « *prem* » et les mots « *document* » et « *documentation* » seront indexés en tant que « *docum* ». Les recherches « *premier document* » et « *première documentation* » seront donc identiques et renverrons les mêmes résultats.

Il y a deux façons de réduire un mot à sa famille linguistique. Soit par un simple raccourcissement ; c'est ce que l'on appel la « racinisation » (ou « stemming » en anglais). Par exemple « coureur » va être indexé en tant que « cour ». Soit grâce à un remplacement par la racine linguistique ; c'est ce que l'on appel la « lemmatisation ». Par exemple « sont » va être indexé en tant que « être ».

Dans les deux cas cette optimisation va fortement dépendre de la langue d'indexation. Bien que la lemmatisation fournisse des résultats de recherche beaucoup plus pertinents, peu de solution d'indexation la propose car elle ne repose sur aucun algorithme et par conséquent elle est plus difficile à mettre en œuvre.

- **Capitalisation et accentuation**

Cette optimisation, bien que très basique, est encore fortement dépendante de la langue. Elle consiste tout simplement à ne pas tenir compte de la case et de certaines accentuations. Par exemple une recherche sur « *ane saint nicolas* » retournera les documents contenant le texte « L'âne de Saint-Nicolas ».

- **Résumé**

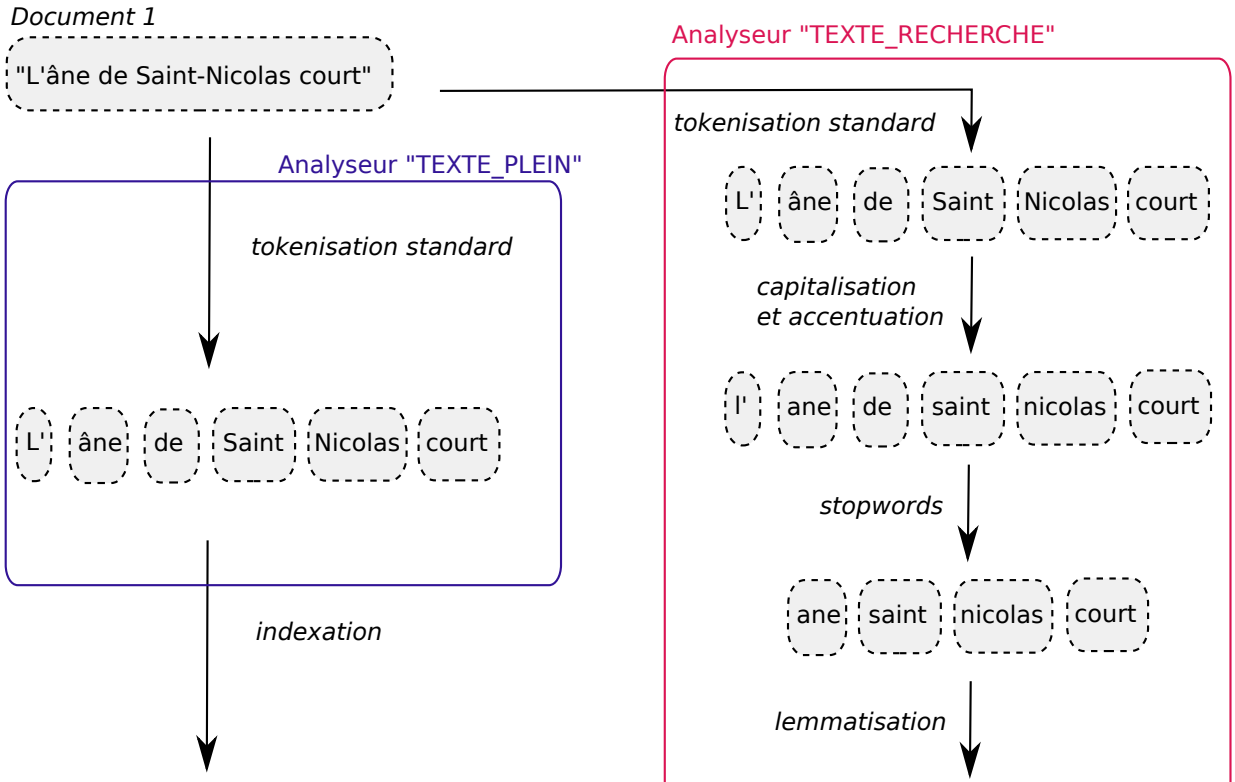
16 https://fr.wikipedia.org/wiki/Racinisation
17 https://fr.wikipedia.org/wiki/Lemmatisation

Lorsqu'un champ textuel d'un document est indexé, il est d'abord découpé en morceaux. Habituellement chaque morceau correspond à un mot mais nous verrons plus tard qu'il existe d'autres façons de découpé un texte, notamment lorsque nous détaillerons les fonctionnalités d'auto-complétion et de suggestion. On appel ce processus de découpe la « tokenisation ». Il est réalisé par un « tokeniser ». Chaque mot passe ensuite par une série de filtres qui seront chacun responsable d'une optimisation, jusqu'à obtenir les termes finaux.

Ce processus d'analyse et de transformation du texte est réalisé par un analyseur qui est associé au tokeniser et aux filtres. Les analyseurs sont fortement lié à une langue. Les mêmes données d'entrée peuvent être analyser par des analyseurs différents, en fonction du champ de l'index à alimenter. Chaque analyseur va indexer le même texte d'une façon particulière et ainsi définir une fonctionnalité pour le champ correspondant.

Il est important de noter que les textes associés aux recherches sont traités par les mêmes analyseurs que ceux utilisés lors de l'indexation. En choisissant le champ sur lequel nous effectuons une recherche, nous définissons indirectement la chaîne de traitement associé au texte et par la même la fonctionnalité de recherche. Si nous considérons l'exemple suivant, une recherche stricte sera faite sur le champ « texte_plein », une recherche plus souple sera faite sur le champ « texte_recherche ».

Soit un index contenant les champs " ***texte_plein*** " et " ***texte_recherche*** ' alimentés à partir des mêmes données,
nous pouvons imaginer l'indexation d'un Document 1 comme suit:



"TEXTE_PLEIN"	"ID:POSISTION"
L'	1:1
âne	1:2
de	1:3
Saint	1:4
Nicolas	1:5
court	1:6

"TEXTE_RECHERCHE"	"ID:POSISTION"
ane	1:1
saint	1:2
nicolas	1:3
courir	1:4

Illustration 9: Exemples d'analyseurs

Les indexes inversés sont particulièrement efficaces pour effectuer des recherches sur les champs textuels. Cependant ils sont aussi capable d'indexer d'autres type de données. Ainsi, a une page HTML nous pourrons associés plusieurs champs textuels issus de l'analyse de son contenu, mais également ses dates d'indexation et de création, ses mots clés et un identifiant issus de son URL d'accès.

Le texte associé aux mots clés et à l'identifiant est considéré comme un type de donnée particulier. Il s'agit de texte invariable sur lequel aucun traitement n'est réalisé et dont la taille est limitée. Il est conçu pour faire des recherches sur des valeurs textuelles constantes.

5.3. Recherches et prototypage

La première phase de conception a été associé au projet PRJ-999. Elle avait pour but de valider l'architecture prévisionnelle et de choisir un prestataire pour les solutions Cloud. Cette phase à donc d'abord consisté à développer un prototype d'indexeur dont la destination et les protocoles de communication des indexes était facilement configurable. Il a ensuite fallu tester plusieurs solutions d'indexation afin de faire un comparatif. Nous soulignons ici que toutes les technologies testées fonctionnent avec des services REST avec un échange des données au format JSON. Que ce soient pour les interfaces d'indexation ou pour les interfaces de recherche. Cela tend à confirmer l'importance que ce mode de communication a pris dans les architectures fortement orientées services[18].

a) Un POC du module d'indexation

Ce prototype, conçu dans un premier temps pour parcourir et indexer le contenu d'une seul VLP, a permis de définir les briques fondamentales de l'architecture applicative définitive de l'indexeur.

- **La création des tâches d'indexation**

Afin de fournir des résultats de recherche pertinents, il fallait entre autre répondre avec des données récentes. Pour ce faire l'objectif était de renouveler l'intégralité des données des 240 VLP (soient 3 millions de documents) dans un délai de 48 heures. Compte tenu de ces contraintes, l'indexeur devait d'or et déjà être capable d'indexer plusieurs page d'une même VLP en parallèle pour que l'entièreté de ces 50.000 pages soient indexés en moins de 2 heures.

Après étude, il a été conclu que le composant le plus adapté pour ce faire était les composants natifs du langage Java appelés les « ForkJoinPool ».

Le principe consiste à définir un nombre défini de threads (sous processus), responsables d'effectuer un nombre indéterminé de sous tâches. Quand un thread est libre, le composant regarde si une sous tâche est en attente d'exécution. Si oui, il affecte la tâche au thread qui l' exécute. Chaque sous tâche étant capable de créer des nouvelles sous tâches. Quand plus aucune sous tâches

18 Communément appelée SOA (Service Oriented Architecture)

n'est en cours ou en attente d'exécution, tous les threads sont supprimés et la tâche mère est terminée.

L'exécution récursive de sous-tâche propre au « ForkJoinPool » est illustré par le schéma suivant[19].

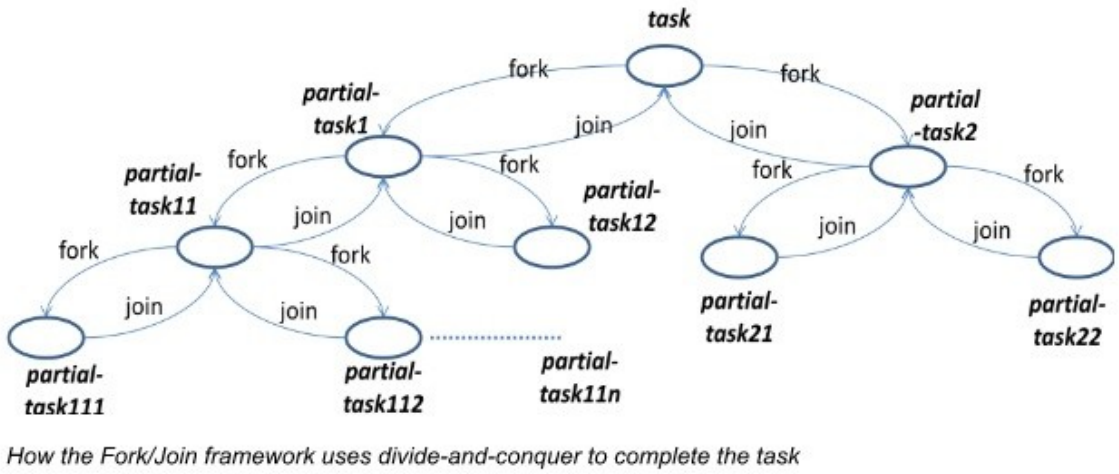


Illustration 10: Exécution récursive des tâches grâce au "ForkJoinPool"

Dans notre cas, lorsqu'on lance l'indexation d'une VLP[20], on crée une « ForkJoinPool » d'un nombre défini de threads. On crée ensuite une seule sous-tâche d'indexation qui correspond à la première page HTML de la VLP. Pour chaque lien contenu dans la page qui correspond à la VLP et qui n'a pas encore été indexé, on crée une nouvelle sous-tâche. Quand tous les threads ont effectués toutes les sous tâches, l'indexation de la VLP est terminée. Les diagrammes d'activités complets de cet algorithme sera présentés lors de la présentation des politiques d'indexation

- **La distinction entre le modèle de donnée de l'indexeur et celui des indexes**

Une tâche d'indexation est divisée en deux étapes. La première étape consiste à récupérer les données issues de la lecture d'une page HTML. La seconde consiste à les stocker dans un index. Le prototype devait pouvoir facilement changer de technologie d'indexation. A ce titre, il n'était pas question de changer la façon dont les données étaient extraites d'une page HTML en fonction de la technologie d'indexation. Il fallait donc séparer le modèle des données extraites de la lecture d'une page HTML du modèle des données envoyées à la technologie d'indexation, même si celui-ci était similaire.

Les différents modules responsables d'indexer les données dans les différentes solutions d'indexation, étaient tous masqués derrière une interface commune. Cette interface propose une

19 Extrait du site https://javatechnocampus.wordpress.com/2015/10/03/544/
20 On verra dans la phases de réalisation, comment plusieurs VLP peuvent être indexées simultanément.

seule fonction d'indexation qui prenait en paramètres d'entrée les données issues de la lecture d'une page HTML. Dans le schéma suivant on montre comment une tâche d'indexation (*IndexingTask*) utilise une composant pour extraire le contenu d'une page HTML (*Parser*) et une interface d'indexation (*Indexer*) qui masque une implémentation spécifique à la technologie utilisée.

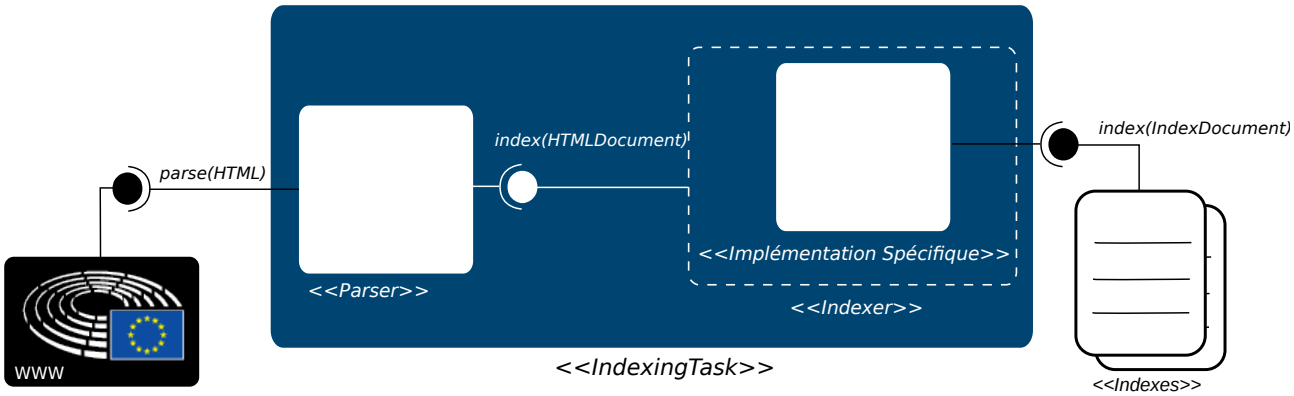


Illustration 11: Composants et interfaces d'une tâche d'indexation

Cette façon de bien séparer les composants par fonctionnalité et de les interfacer avec des définitions invariables est un modèle d'architecture de code fondamental connu sous le nom de « separation of concerns » (séparation des préoccupations).

Ce principe a été conservé même après s'être arrêté sur le choix d'une technologie d'indexation, et ce bien que le modèle de donnée issu de la lecture d'un page HTML soit strictement équivalent au modèle de donnée envoyé à la solution d'indexation.

L'interface graphique de recherche proposé par le prototype fonctionnait sur le même principe. Quelle que soit la solution d'indexation, les services exposés par le POC utilisait une interface générique masquant une implémentation spécifique. De cette façon, une modification de la technologie n'a nécessité aucune modification du service de recherche.

b)La solution Azur Search

La solution « *Azur Search* » est le service Cloud de Microsoft dédié aux stockage d'information sur des index inversés. Une fois les indexes créés grâce à une interface Web particulièrement bien conçue, la solution propose des services REST au format JSON pour stocker ou mettre à jour des documents et pour effectuer des recherches.

La particularité de cette solution est qu'elle masque la complexité du fonctionnement. L'administrateur défini le modèle de données des documents a indexer (les différents champs qui le composent) mais ne configure pas les chaînes d'analyse textuelles tel que nous les avons présentées.

Le client du service envoie une chaîne de mots à rechercher et la solution lui renvoie des résultats triés par ordre de pertinence. La manière dont elle définie cette pertinence est occulté à l'utilisateur.

« Azur Search » a un très bon support des différentes langues. Il est capable de fournir une analyse par lemmatisation (différent du stemming) sur 15 des langues utilisées sur EUROPARL.

En plus de la recherche textuelle, la solution d'Azur proposait des champs de type date ou de type valeur textuelle constante, permettant de répondre aux besoins de recherche par dates, par mots clés ou par catégories. Elle proposait un service d'auto-complétion du texte sur les valeurs constantes qui pouvait être utilisé pour les titres HTML des pages.

Les fonctionnalités d'auto-complétion proposées par ces solutions « Tout en Un » ont pour contrainte que le texte tapé par l'utilisateur doit impérativement commencé par le texte proposé. Par exemple le titre de page « Textes Parlementaires » ne sera proposé que si l'utilisateur tape les caractères « tex ». Nous verrons plus tard comment il est possible grâce aux solutions configurables de proposer la même titre si l'utilisateur tape les caractère « par ».

« Azur Search » fournit une très bonne expérience de recherche et offre des fonctionnalités couvrant tous les besoins exigés et la majorité des autres besoins. Il simplifie l'implémentation du moteur de recherche en soustrayant aux développeurs la configuration délicate de l'analyse textuelle. Comme toutes les solutions Cloud, en cas de manque de ressources, la mise à l'échelle de notre capacité de stockage et de notre puissance de calcul, peux se faire grâce à un simple clique.

c) Les solutions Amazon

- **CloudSearch**

« CloudSearch » est le strict équivalent de la solution « AzurSearch » mais sur la plateforme « Amazon ». Une différence mineure se situe au niveau de l'interface d'administration qui est beaucoup moins bien conçue. La différence majeure avec « AzurSearch » est le support des langues. Les analyseurs de langues fonctionnent avec du stemming et non pas de la lemmatisation. Le résultat final est une expérience de recherche acceptable mais beaucoup plus décevante que celle offerte par « AzurSearch ».

- **Elasticsearch service**

Avant de détailler comment « Amazon » propose des services « Elasticsearch », décrivons brièvement de quoi il s'agit. « Elasticsearch » un logiciel opensource développé en Java. Il repose sur la librairie nommée « Lucene », également opensource, qui fournit une implémentation des indexes inversés. En plus d'exposer des services qui exploitent les fonctionnalités offertes par « Lucene », « Elasticsearch » propose une architecture applicative ainsi que de nombreuses fonctionnalités supplémentaires. Notamment des services d'études statistiques sur les données.

L'architecture applicative proposée par « Elasticsearch » est un système de cluster pouvant héberger plusieurs noeuds (machines virtuelles ou non) qui représente chacun une instance de

« Elasticsearch »[21]. Chaque nœud ou instance distribue sa charge de travail sur plusieurs exécution indépendante de « Lucene ». Une exécution active de « Lucene » est appelé un «Shard» (tesson en français). Ainsi lorsqu' un administrateur crée un index, il défini le nombre de « Shard » sur lesquels les données de l'index seront distribuées, ainsi que le nombre de replicas (copies) pour chacun de ces « Shard ». Ces copies sont hébergées sur des nœuds différents. Cette architecture garantie un niveau de disponibilité élevée du service et une répartition de la charge. Les administrateurs ou les simples utilisateurs ne se soucient pas de l'endroit où sont réparties les données. Le seul point d'entrée est le cluster. C'est Elasticsearch qui se charge de répartir ou de récupérer les données.

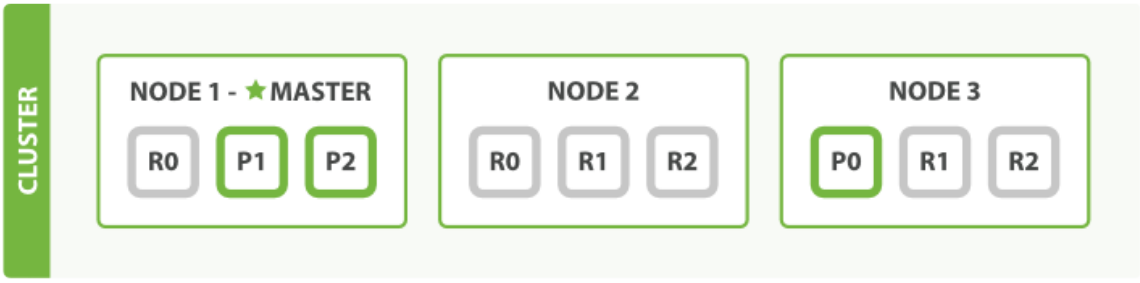


Illustration 12: Exemple du cluster "Elasticsearch"

Le schéma précédent, issus de la documentation officielle, illustre un index réparti sur les trois « shards » primaires P0, P1 et P2. Chacun de ces « shards » a deux replicas réparties sur les différents nœuds. Si un nœud ne fonctionne plus, l'ensemble des données est encore disponible. La documentation officielle insiste sur le fait que ceci est vrai même si le nœud maître tombe en panne. Celui-ci n'ayant un rôle particulier qu'a la création et à la suppression des indexes.

« Elasticsearch » ne propose pas d'interface graphique d'administration. Une fois que l'application est déployée tout se configure grâce aux services REST qu'elle expose. Le schéma suivant est un exemple simplifié et commenté de la façon dont nous créons un index avec « Elasticsearch » (les parties manquantes seront détaillées lors de la description de la réalisation des fonctionnalités).

21 https://www.elastic.co/guide/en/elasticsearch/guide/current/distributed-cluster.html

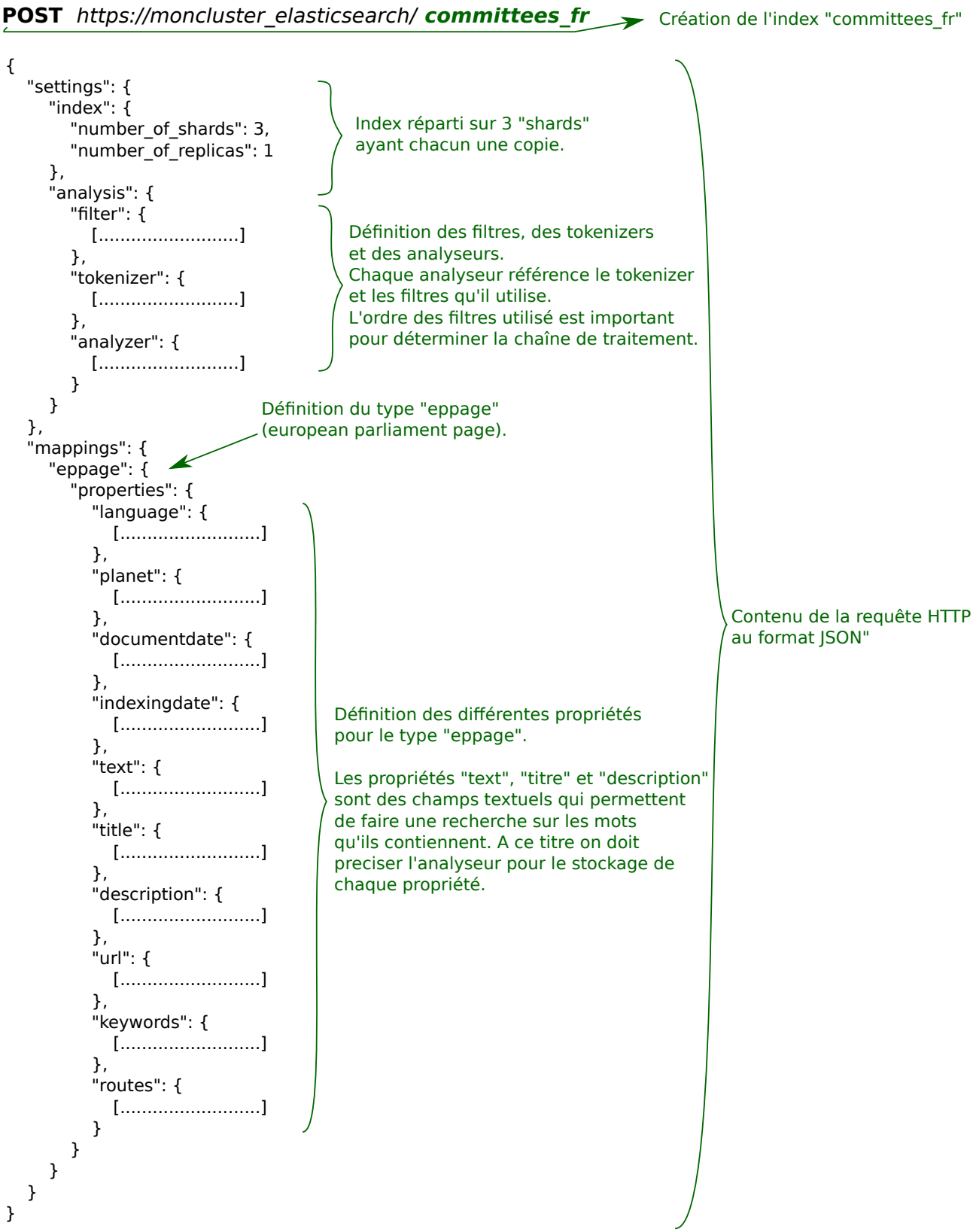


Illustration 13: Exemple commenté de la création d'un index "Elasticsearch"

D'un point de vue logiciel, Amazon ne propose rien de plus que ce que propose nativement « Elasticsearch ». La valeur ajouté que Amazon apporte dans son service « Elasticsearch » est la facilité avec laquelle il est possible de déployer des clusters avec le nombre désiré de nœuds. Une fois l'infrastructure en place, toute l'administration des indexes se fait avec des requêtes REST au format JSON.

« Elasticsearch » est la plus puissante des solutions. Il n'y a pas de fonctionnalité de recherche qui ne soit pas réalisable. Le seule point pouvant apporter une modération est qu'il ne propose pas nativement de filtre de lemmatisation pour toutes les langues désirées. Malgré cela, grâce à un configuration ouverte, c'est cette solution qui offre la meilleure expérience de recherche. La contrepartie est la complexité qu'elle apporte dans la réalisation du moteur de recherche. Une configuration précise permet d'étendre le champ des fonctionnalités mais contraint d'autant plus la façon dont les documents sont indexés ainsi que la manière dont les requêtes de recherche doivent être formatées.

d) L'hébergement des index ElasticSearch sur le réseau du Parlement

Elasticsearch est la solution que la DGITEC autorise à déployer sur le réseau interne du Parlement Européen. Les seules différences avec un déploiement sur Amazon serait le mode d'authentification des requêtes et la lourdeur des processus de déploiement. Actuellement la DGITEC ne propose pas de déployer un cluster grâce à un simple clique de la souris.

5.4. Choix de la solution

a) Étude comparative

Critère	AzurSearch	CloudSearch	Elasticsearch Amazon	Elasticsearch Interne
Répond aux besoins non négociable. (Flexibilité nulle)	OUI	OUI	OUI	OUI
Pertinence des résultats	BONNE	MOYENNE	BONNE	BONNE
Répond aux besoins négociable. (Flexibilité non nulle)	EN PARTIE	EN PARTIE	OUI	OUI
Facilité d'implémentation	FACILE	FACILE	COMPLEXE	COMPLEXE
Facilité de déploiement	FACILE	FACILE	FACILE	LABORIEUSE

Considérant que la complexité apporte une valeur ajouté faible et qu'une application simple et une application pérenne, le choix de l'auditeur s'est porté en premier lieu vers la solution proposé par « AzurSearch ». Cette proposition n'a finalement pas été retenue à cause d'une problématique comptable.

C'est finalement la solution « Elasticsearch » sur Amazon qui sera retenue car ces résultats sont meilleurs que ceux offerts par la solution « CloudSearch ». Elle offre en outre la possibilité d'être rapatriée sur le réseau du Parlement Européen dans le cas d'une rupture de contrat avec le fournisseur de technologie Cloud.

b)Schéma d'architecture technique définitif

En conclusion du projet PRJ-999 nous pouvions dressé le schéma d'architecture définitif du moteur de recherche. Nous présentons ici un schéma commenté, issu de la documentation technique du projet.

Nous noterons la présence de deux composants « Amazon » que nous n'avons pas encore détaillés ; « ElasticBeanstalk » et « Amazon S3 ». Le premier est la plateforme proposée par Amazon pour déployer des applications Web (PAAS). Ces applications sont déployées sur des machines virtuelles qui n'ont aucune capacité de stockage. Les fichiers de configuration sont donc accessibles avec le service « Amazon S3 » qui permet de stocker et de lire des fichiers grâce à des services web.

Nous pouvons aussi remarquer que toutes les requêtes HTTP soumises aux services d'Amazon comporte une signature d'authentification. Y compris les requêtes vers les indexes « Elasticsearch».

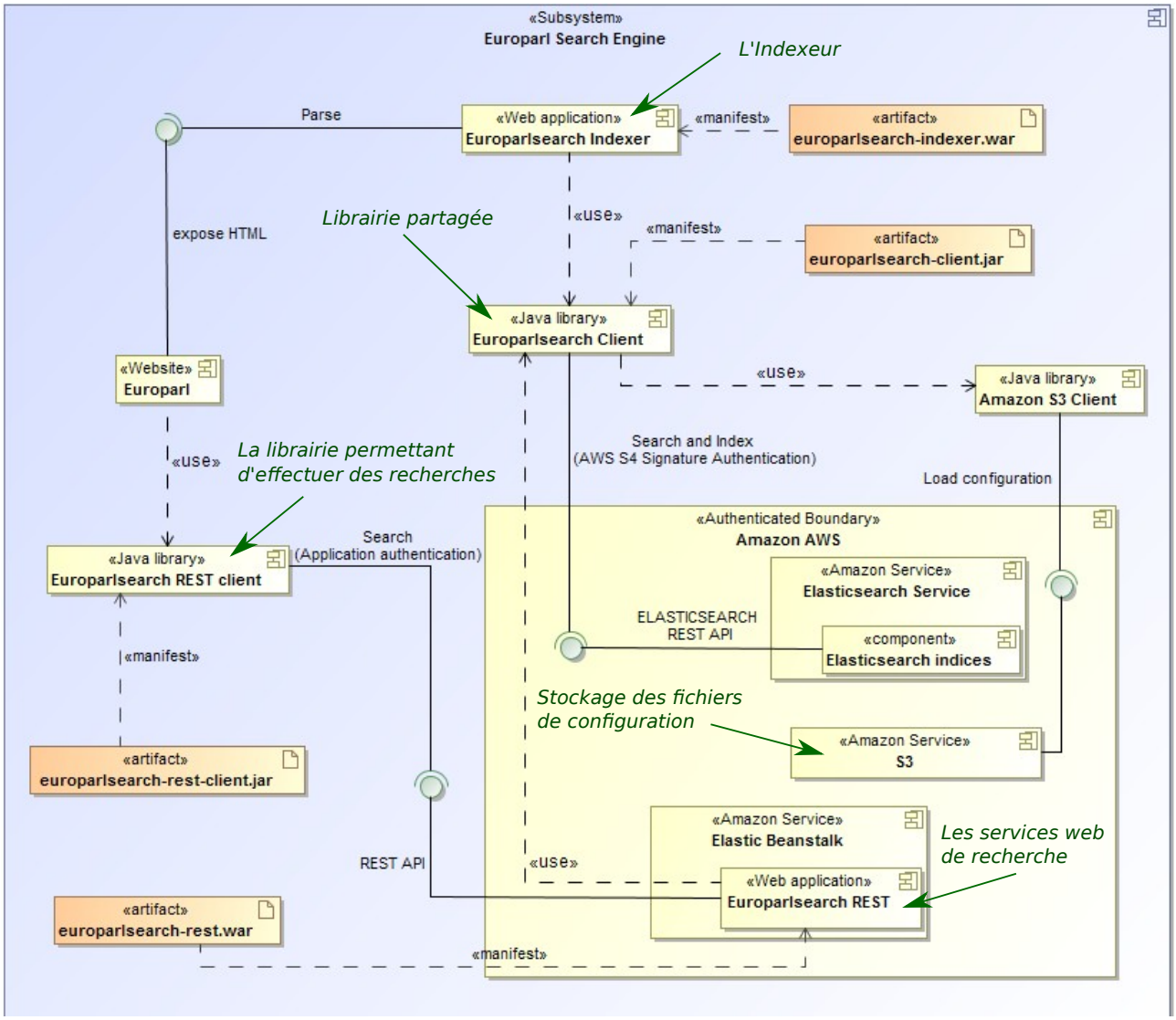


Illustration 14: Architecture définitive du moteur de recherche

Les quatre composants conçus et développés par l'auditeur et qui constituent le moteur de recherche du site EUROPARL sont :

- **Europarlsearch Indexer**

Application web responsable de parcourir régulièrement le contenu du site internet EUROPARL pour indexer son contenu. L'application propose une interface d'administration. Elle se manifeste par le fichier « europarlsearch-indexer.war ».

- **Europarlsearch REST**

Application web exposant des services REST sécurisés permettant d'effectuer des recherches sur le contenu des pages indexées du site EUROPARL. Elle est hébergée sur le service « ElasticBeanstalk » et se manifeste par le fichier « europarlsearch-rest.war ».

- **Europarlsearch Client**

Librairie Java contenant le code d'accès à la configuration et le code de manipulation des indexes. Elle est partagée par les composants « Europarlsearch Indexer » et « Europarlsearch REST ». Elle se manifeste par le fichier « europarlsearch-client.jar».

- **Europarlsearch REST client**

Librairie Java permettant d'effectuer des recherches sur les pages indexées du site EUROPARL en utilisant une interface de recherche intuitive et bien définie. Elle a vocation a être intégrée aux applications qui constituent EUROPARL et se manifeste par le fichier « europarlsearch-rest-client.jar».

5.5. Analyse fonctionnelle interne

A partir d'une étude technique considérant des possibilités offertes par les choix technologiques, les besoins exprimés ont pu être reformuler avec un dictionnaire précis de fonctionnalités. Les diagrammes de cas d'utilisations présentés ci-dessous ont été extraits de la documentation technique.

On notera que dans la norme UML (Unified Modeling Language) des diagrammes de cas d'utilisation utilisée ci-dessous, la directive « include » indique un composant obligatoirement inclus. La directive « extends » indique une fonctionnalité optionnelle. L'héritage des fonctionnalités se manifeste par des flèches pleine sans directive.

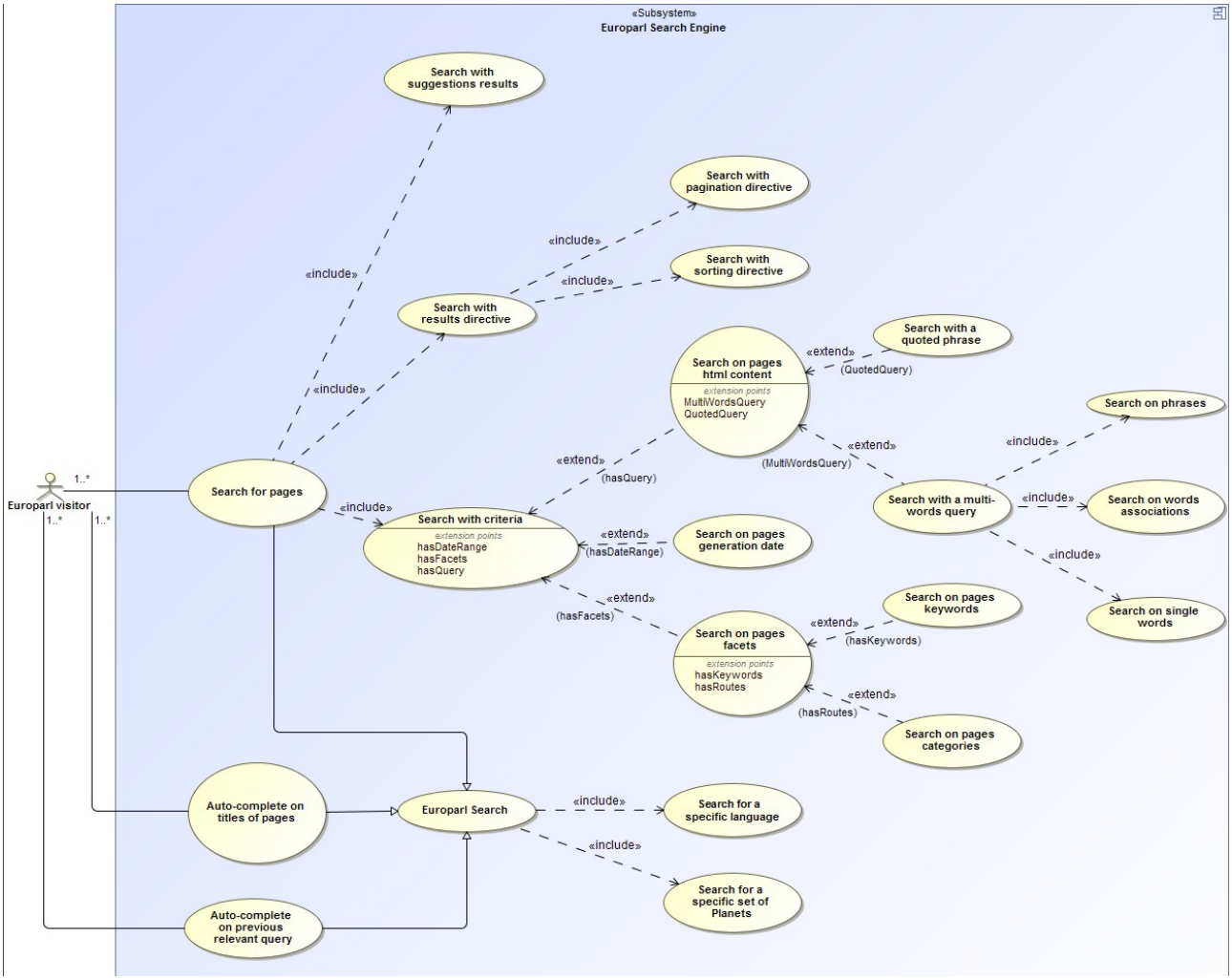


Illustration 15: Cas d'utilisations d'un visiteur sur EUROPARL

a) Cas d'utilisation d'un visiteur

L'interface de recherche propose trois fonctions principales que sont la « recherche de pages », « l'auto-complétion sur les titres » et « l'auto-complétion sur les requêtes pertinentes déjà effectuées». Cette dernière fonctionnalité a été mise en place pour remplacer l'auto-complétion sur le texte d'une page qui est trop consommatrice de ressource.

Chacune de ces fonctionnalités sont des recherches sur EUROPARL et doivent donc préciser la langue et les planètes sur lesquelles elles sont effectuées.

La « recherche de pages » inclut nécessairement des « directives de tri et de pagination » ainsi que des « critères de recherche ».

Les « critères de recherche » peuvent être exclusivement ou additionnellement,

- sur les mots clés (propriété « keywords ») et/ou les catégories (« routes »).
- sur la date du document

- sur le contenu textuel du document.

Les critères de recherche sur le contenu textuel sont soit une recherche de phrase exacte à l'aide de guillemets, soit une recherche de mots. Cette dernière va combiner une recherche sur les phrases, sur la proximité entre les mots et sur les mots isolés pour déterminer la pertinence d'un résultat. Les résultats sur les recherches textuelles mal orthographiée propose des suggestions de recherche pertinentes. (TODO : Faute schéma, déplacer le use case)

b)Cas d'utilisation de l'administrateur

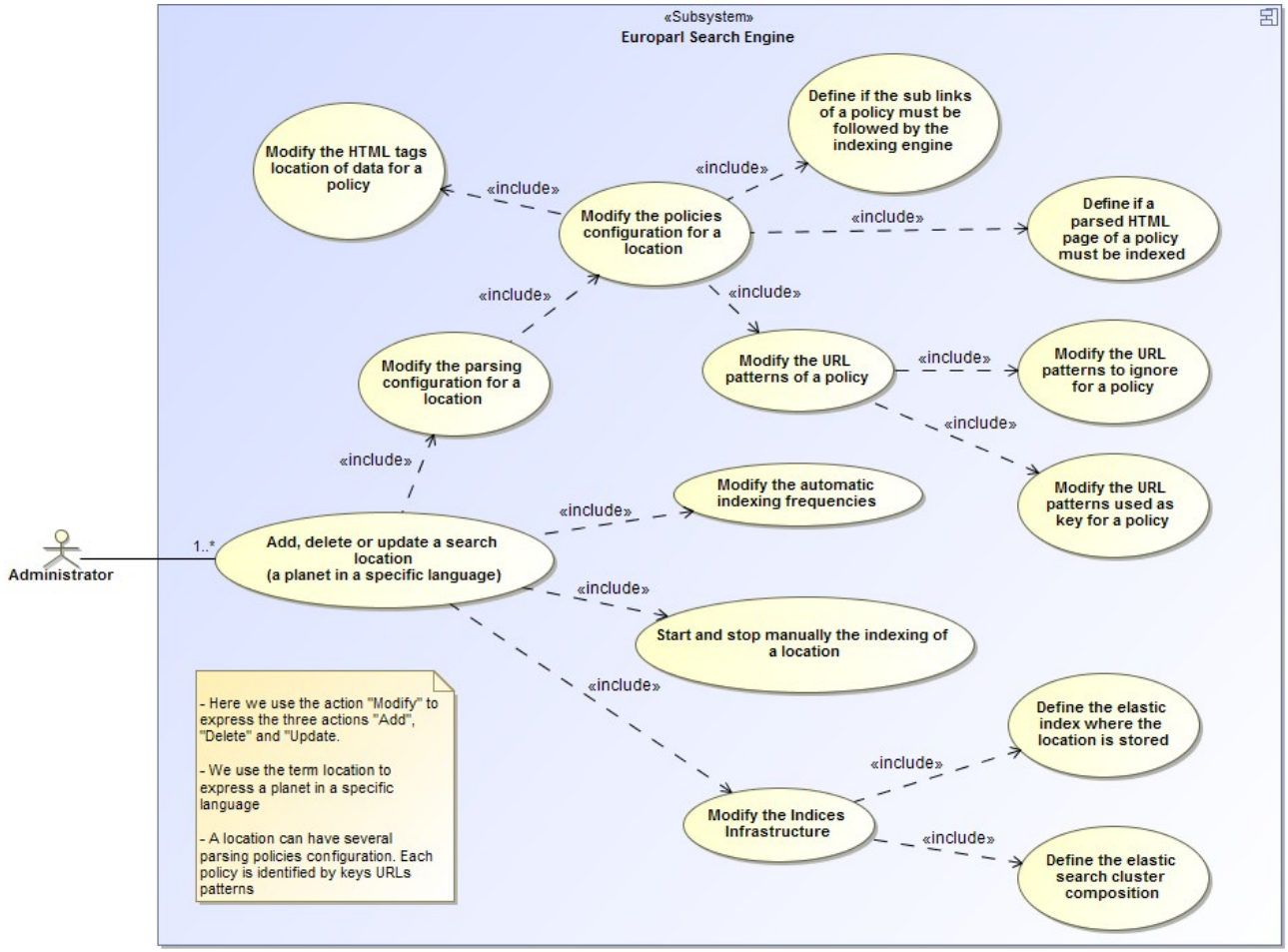


Illustration 16: Cas d'utilisation d'un administrateur

Un utilisateur doit être capable d'ajouter, de mettre à jour ou de supprimer une VLP de l'application d'indexation.

Ce qui inclut

- de définir l'index et le cluster où elle est indexées,
- de définir ses fréquences d'indexation automatique,

- de pouvoir déclencher ou arrêter manuellement son indexation,
- de définir la façon dont sont accédées et extraites ses données.

La décomposition de cette dernière fonctionnalité aborde des détails d'implémentation qui seront expliqués dans la description de la phase de réalisation.

6. La réalisation

6.1. Les fichiers de configurations

En analysant les fonctionnalités offertes par l'indexeur et les services web, l'auditeur a isolé cinq types de configuration :

- la liste des clusters « Elasticsearch », qui définit leur point d'accès ainsi que les indexes qu'ils contiennent,
- l'association entre les VLP et les indexes, qui permet de définir sur quel index est indexée une VLP,
- la définition des fréquences d'indexation pour chaque VLP,
- les politiques d'indexations pour chaque VLP qui déterminent comment elles sont parcourues par le moteur et comment en sont extraites les données,
- les politiques d'indexations générales qui définissent l'obsolescence des pages et permettent de déterminer si une page a besoin d'être ré-indexée ou supprimée de l'index.

Chacune de ces configurations a un modèle de donnée complexe et nécessitent de pouvoir être manipulée et mise à jour facilement. Une base de donnée étant surdimensionnée pour de la configuration et mal adapté a la visualisation humaine, l'auditeur a choisi de stocker ces informations sur des fichiers XML.

Le modèle de données de chacune de ces configurations est défini par des fichiers XSD (XML Schema Definition). Lors de la compilation des applications, ces définitions sont utilisées pour générer automatiquement les classes Java représentant l'intégralité du modèle associé à chaque type de configuration. En utilisant la librairie Java native « JAXB » (Java Architecture for XML Binding), les fichiers portant la configuration sont automatiquement convertis en objets Java. La lecture et l'édition des différentes données se fait donc par la manipulation d'objet Java et non pas par la manipulation directe de fichiers.

Les différents fichiers de définition sont

- « domains.xsd », pour la localisation et la composition des clusters (dans notre terminologie applicative, un cluster est appelé un domaine),

- « clients.xsd », pour le lien entre les VLP et les indexes, qui va permettre de générer un objet Java pour chaque VLP qui sera dédié à son indexation et à sa consultation,
- « triggering.xsd », pour les fréquences d'indexation automatique de chaque VLP,
- « parsing.xsd », pour les politiques d'extraction de données et d'indexation propres à chaque VLP,
- « policies.xsd », pour les politiques d'indexation générales relatives à l'obsolescence des pages.

6.2. Le développement des clients

Sur le schéma d'architecture, on remarquera la présence de la librairie « Europarlsearch Client » qui est commune à l'indexeur et aux services web. Cette librairie factorise le code relatif à l'accès aux fichiers de configuration et le code relatif à l'accès aux indexes.

a) L'accès au fichier de configurations

Les fichiers de configuration sont stocker sur le service « Amazon S3 » et nécessitent une interaction particulière. La librairie « Europarlsearch Client » propose l' implémentation générique d'une classe permettant de récupérer un fichier XML situé sur ce service. Les classes héritant de cette classe générique se contente de spécifié le nom du fichier qu'elles veulent récupérer. Ainsi, si le mode de stockage venait à changer, seule la modification de la classe générique devra être modifiée et le changement sera transparent pour les classes filles.

La librairie permettant d'accéder aux indexes, elle propose déjà deux implémentations de la classe générique. Celle permettant d'accéder à la configuration des domaines (*domains.xml*) et celle permettant d'accéder à la configuration faisant le lien entre les VLP et les indexes (clients.xml).

Ces classes d'accès aux données sont généralement appelées des DAO (Data Access Object). C'est la terminologie employée dans le schéma ci-dessous, extrait de la documentation technique relative à la libraire « Europarlsearch Client ».

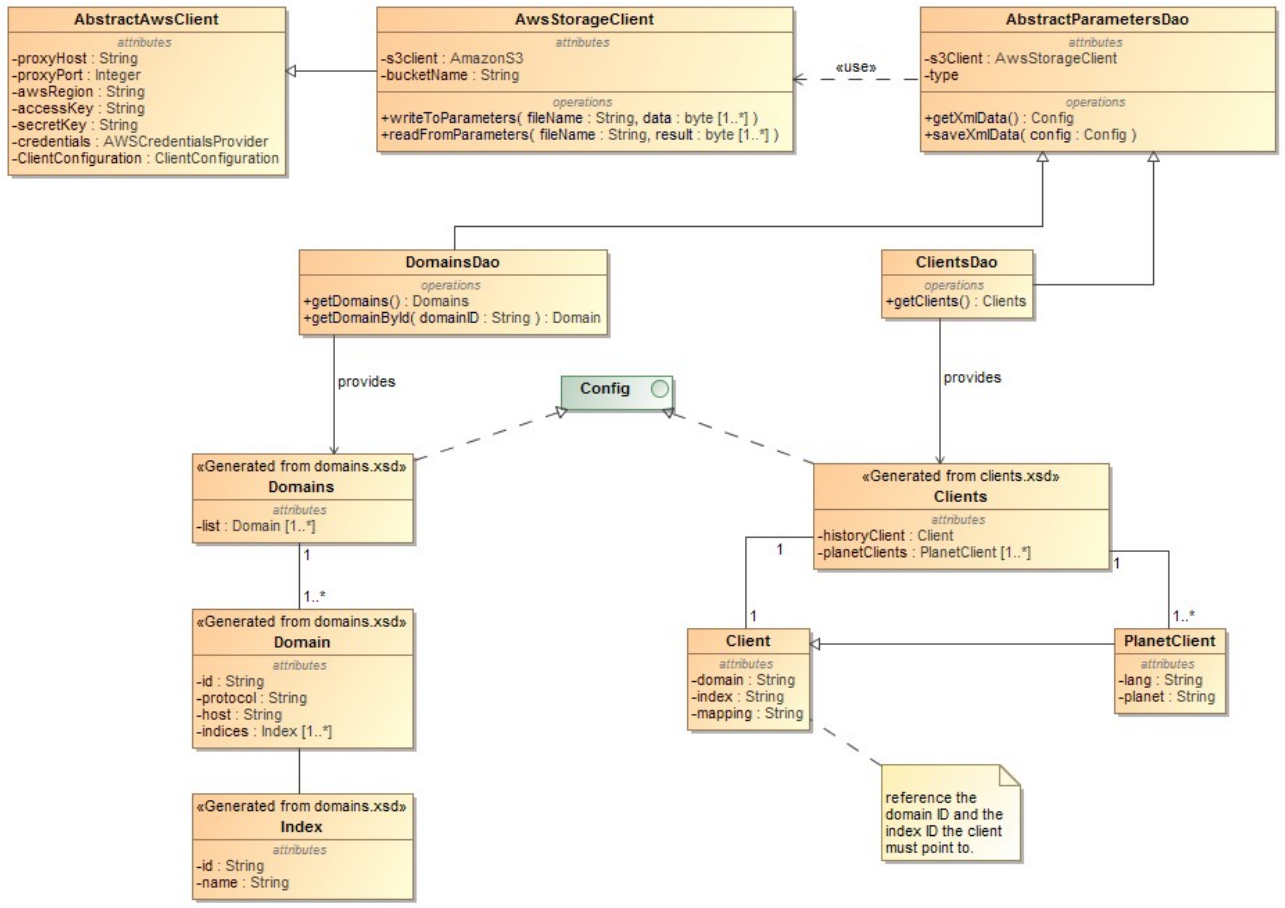


Illustration 17: Diagramme de classes d'accès à la configuration des clients

b) Le registre des clients

6.3. Développement du module d'indexation

a) Le démarrage de l'application

Lorsque l'indexeur est démarré, il doit être autonome et fonctionner sans manipulation de la part de l'administrateur. Cela implique que tous les index soit crées et que les fichiers de configuration

b) La gestion des fréquences d'indexation

c) Le lien entre les pages html et le modèle de données

d) La distinction entre les liens suivis et les pages indexées

e) Politiques d'indexation et suppression des pages obsolètes

f) La gestion des pages en erreur

(des solutions imparfaites : effort maximal)

g) Une interface d'administration

6.4. L'implémentation des fonctionnalités de recherches

a) Résumé des concepts Elasticsearch

b) Le couplage entre la structure des index et les fonctionnalités de recherche

c) La recherche sur une phrase

d) Une recherche simultanée sur la phrase et sur les mots

e) Corrections et suggestions de « phrases »

f) Auto complétion sur l'historique des requêtes pertinentes

6.5. Développement du web service de recherche

(note : Une interface qui masque la complexité)

a) Un web service REST utilisable depuis n'importe quelle source

b) La définition du modèle de donnée exposé

c) Implémentation d'une page de recherche simplifiée

6.6. Intégration du client de recherche au site internet du parlement

6.7. Un développement par itération

a) De plus prioritaire au moins prioritaire

b) Des phases d'analyse approfondies et une redéfinition des besoins

(note : Une technologie mal comprise par la maîtrise d'ouvrage et par la maîtrise d'œuvre + coûts incertains de la plateforme)

a) Validation et boucle de retro-action

(note : exemples :

fréquence parsing trop importante,

synonymes trop laborieux remplacé par la suggestion de phrase,

autocompletion fulltext remplacée par l'historique des requêtes,

Déclaration dynamique des planètes,

Refonte du mapping planete ↔ index)

7. Tests et intégration continue

7.1. Test unitaires et couvertures de code

(tests d'algorithme)

7.2. Les critères qualité de l'analyse automatique du code

7.3. Les tests d'intégration

(tests techniques d'intégration)

7.4. Les tests de validation

(tests utilisateurs sur la satisfaction des besoins)

8. La documentation

8.1. Le Document Technique d'Architecture (DTA)

8.2. La documentation technique

8.3. La documentation Java et les commentaires

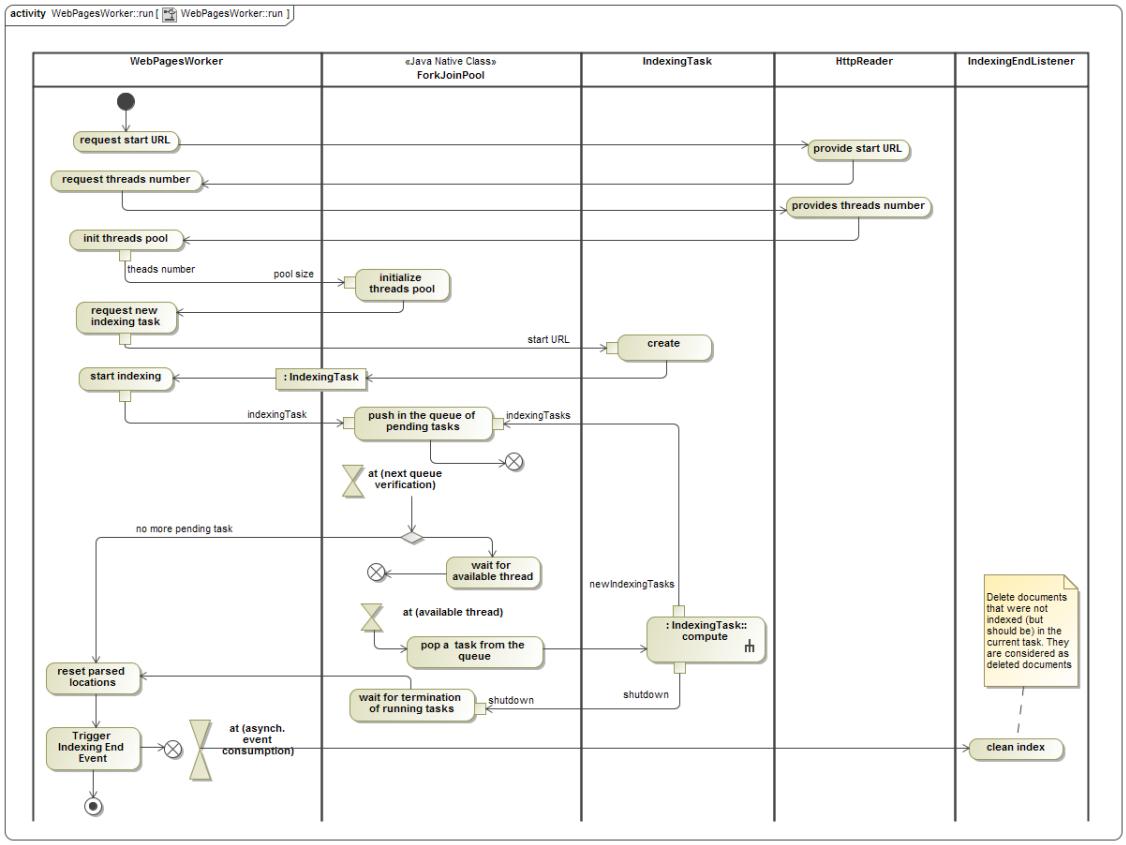
9. La formation utilisateurs

10. Retours d'expérience

11. Conclusion

12. Annexes

Annexe 1


activity WebPagesWorker::run [WebPagesWorker::run]
WebPagesWorker
«Java Native Class»
ForkJoinPool
IndexingTask
HttpReader
IndexingEndListener
request start URL
provide start URL
request threads number
provides threads number
init threads pool
theads number
pool size
initialize threads pool
request new indexing task
start URL
create
start indexing
: IndexingTask
indexingTask
push in the queue of pending tasks
indexingTasks
at (next queue verification)
no more pending task
wait for available thread
newIndexingTasks
at (available thread)
: IndexingTask:: compute
pop a task from the queue
Delete documents that were not indexed (but should be) in the current task. They are considered as deleted documents
reset parsed locations
wait for termination of running tasks
shutdown
shutdown
Trigger Indexing End Event
at (asynch. event consumption)
clean index