

UNIVERSITE ASSANE SECK DE ZIGUINCHOR



UFR : SCIENCES ET TECHNOLOGIES

DÉPARTEMENT : INFORMATIQUE

MASTER : INFORMATIQUE

SPÉCIALITÉ : GÉNIE LOGICIEL

Sujet :

Etude et conception de services web REST pour des applications clientes

Présenté par :
KEBA MBAYE DIEDHIOU

Sous la codirection du :
Pr. SALOMON SAMBOU
Dr. YOUSOU DIENG

Membres du Jury :

Pr. SALOMON SAMBOU (Président du jury)

Dr. IBRAHIMA DIOP (Rapporteur)

Dr. YOUSOU FAYE (Examineur)

Dr. KHADIM DRAME (Examineur)

Dr. YOUSOU DIENG (Encadreur)

Année Universitaire : 2016/2017

Remerciements

C'est un grand plaisir pour moi d'achever ce projet. C'est aussi l'occasion de remercier tous les gens qui m'ont aidé, soutenu ou supporté pendant toute la durée de ce stage. Mes premiers remerciements vont à l'encontre de mon maître de stage, Mr Kémo Touré directeur de linked Partners ainsi qu'à toute son équipe.

Je le remercie pour les longues heures de travail stimulantes et surtout toujours enrichissantes. Sa passion pour la quête perpétuelle du savoir est inépuisable. Merci de m'avoir poussé dans mes derniers retranchements.

Mes remerciements vont aussi à l'encontre de mon encadreur **Dr Youssou Dieng**, malgré son emploi du temps serré avec le cumul de fonctions en tant que chef de département informatique et enseignant chercheur au niveau de l'Université Assane SECK de Ziguinchor, il m'a aidé pour la rédaction du mémoire. Je salut son professionnalisme, sa quête perpétuelle du savoir illimitée, son rigueur sur le travail et sa capacité à prendre des décisions quelques soient les conséquences. J'ai appris de vous le sens de la rigueur sur le travail ainsi que les valeurs intrinsèques d'une personne.

Je souhaite remercier les membres de mon jury :

Dr Salomon SAMBOU, professeur à l'Université Assane SECK de Ziguinchor, pour le temps qu'il a bien voulu consacrer à l'évaluation de ce travail, mais aussi, de m'avoir fait l'honneur de présider le jury de ma soutenance.

A mon rapporteur **Dr Ibrahima Diop**, enseignant chercheur à l'Université Assane SECK de Ziguinchor, pour avoir fait une critique positive du document. Je vous remercie d'avoir apporté des solutions formelles aux manquements de ce document.

Et enfin aux examinateurs **Dr Youssou Faye** et **Dr Khadim Dramé**, enseignants chercheurs à l'Université Assane SECK de Ziguinchor, tous deux ont su apportés des suggestions au document et de m'avoir fait l'honneur de participer au jury de soutenance.

Je souhaite aussi remercier l'ensemble des enseignants du département d'informatique de l'Université Assane Seck de Ziguinchor. Le savoir transmis m'a été d'une très grande utilité en entreprise.

Je tiens à remercier très chaleureusement tous les membres de ma famille principalement ma mère qui n'a ménagé aucun effort pour la réussite de mes études. Mais aussi à toutes mes promotionnaires qui m'ont su donnés le peu de leurs temps pour la réussite de ce stage.

Ce travail est dédié à mon père qui se trouvent dans les cieux. Que son âme repose en paix. Amen.

Résumé

De nos jours, nous constatons une augmentation considérable d'utilisateurs connectés à internet via leur téléphone portable ou tablette. De ce fait, un besoin impératif de mettre en place des applications qui vont tourner sur plusieurs plate-formes se fait ressentir par les entreprises. A défaut de mettre en place une bonne architecture permettant le développement des plate-formes (web et mobile) en optimisant le temps et de trouver les technologies nécessaires pour mener à bien leurs projets, les entreprises se heurtèrent à des problèmes. C'est dans ce sens que l'entreprise LinkPartnets a mis en place un sujet de stage dans le but de résoudre ces problèmes.

Dans cette mémoire, nous avons commencé au préalable par une étude générale des services Web. A l'issu de cette étude nous avons opté de travailler avec les services Web de type **REST**. Ensuite nous avons proposé une nouvelle architecture basée sur ces services permettant de développer toutes les fonctionnalités de l'aplication. Et enfin nous avons développé la plate-forme Web de l'application **wutiko**.

Table des matières

Remerciements	ii
Résumé	iii
Table des figures	v
Chapitre 1 : Introduction	1
1.1 Contexte	1
1.2 Problématique	2
1.3 Méthodologie utilisée	3
1.4 Plan du mémoire	4
Chapitre 2 : Etude et conception de services Web REST	6
2.1 Étude de l'existant	6
2.2 Conception de services Web REST	7
Chapitre 3 : Conception d'une application cliente pour la recherche d'emploi en ligne	15
3.1 Spécification des besoins	15
3.2 Diagramme des cas d'utilisation	16
3.3 Conception de l'application cliente	24
3.4 Conception détaillée (Diagramme de classes et dictionnaire)	26
Chapitre 4 : Implémentation et Présentation de l'application	30
4.1 Implémentation du backend	30
4.2 Implémentation du frontend	34
4.3 Présentation de l'application	37
Conclusion et Perspectives	49
5.1 Conclusion	49
5.2 Perspectives	49
References	50

Table des figures

1.1	Évolution des recherches Google sur les mots clés REST et SOAP	3
1.2	Processus de développement en Y	4
2.3	Cas d'anomalie de l'architecture wutiko	8
2.4	Architecture client-serveur	9
2.5	Communication entre navigateur et serveur	13
3.6	Diagramme des cas d'utilisations	17
3.7	Diagramme de cas d'utilisation Authentifier	18
3.8	Diagramme de cas d'utilisation Gestion des utilisateurs et les droits d'accès	19
3.9	Gestion des curriculums vitae	20
3.10	Gestion des emplois	21
3.11	Gestion des offres d'emploi	22
3.12	Gestion des entreprises	23
3.13	Gestion du réseau social	24
3.14	Gestion du backend	25
3.15	Architecture de notre application	26
3.16	Diagramme de classes	27
4.17	Différents processus pour l'implémentation	31
4.18	Modèle MVC	35

Chapitre 1 : Introduction

La relation entre l'internet et ses utilisateurs a grandi et évolué au fil du temps. Dans un premier temps, l'internet fourni principalement du contenu statique (pages HTML avec comme structure du texte, des images et des hyperliens). Ainsi, les utilisateurs peuvent interagir par le biais de discussions, forums, e-mail ... Depuis les années 90, grâce à des milliards d'utilisateurs connectés dans le monde, l'internet a connu une expansion massive. Ce nombre croissant d'utilisateurs a également attiré l'attention des développeurs à créer de nouveaux services ou de changer leurs activités en ligne. En raison de la nécessité de consommation et de manipulation de l'information, l'internet est devenu un vaste réseau de partage de données entre différentes entités. C'est ainsi que les services Web ont attiré l'attention que le Web n'était pas constitué que de fichiers HTML mais plutôt qu'un ensemble de connaissances accessible via des standards, des protocoles ...

Ces dernières années, les entreprises comme Google, Twitter, Amazon et Facebook ont commencé à changer de nombreux aspects de notre vie quotidienne. Ils sont intégrés dans de nombreux produits que nous utilisons (ordinateurs, téléphones portables et tablettes). Il y a plusieurs raisons pour lesquelles ces entreprises sont devenues très populaires. L'une d'eux est certainement le fait qu'ils ont tous mis leurs services Web à la disposition du public. Ce qui les a permis de propager leurs services sur différents périphériques et sur le Web. L'autre raison est le fait que la plupart de leurs utilisateurs se connectent à travers leurs téléphones portables ou tablettes. Afin de pouvoir satisfaire des millions de demandes, ces entreprises doivent avoir une très bonne architecture. Toutes ces entreprises citées ci-dessus utilisent une architecture basée sur les services Web de type REST. Toujours dans le but de satisfaire des applications clientes : Web et mobiles (Android, IOS, Windows phone, ...), il est nécessaire de mettre en place une seule architecture où toutes ces applications vont se connecter. De ce fait avec la concurrence trop rude et l'évolution rapide des nouvelles technologies, il est nécessaire de développer des produits accessibles à des millions d'utilisateurs en un temps optimal.

1.1 Contexte

1. Cadre du stage

Dans le cadre de mon stage de fin d'études de Master 2 en Informatique à l'UFR Sciences et Technologies de l'université Assane Seck De Ziguinchor, j'ai effectué un stage au sein de l'entreprise Linked Partners. C'est une entreprise sénégalaise inspirée de LinkedIn qui est une entreprise américaine travaillant sur la recherche d'emploi en ligne. LinkPartners est constituée de deux ingénieurs informatique, d'une infographiste, d'une assistante de direction et d'un chargé relation client. Elle offre des services aux entreprises dans le conseil en systèmes d'informations, l'organisation, le développement d'application, l'analyse de données, l'intégration de solutions, la gestion de projets et le transfert de compétences.. Elle a développé une application intitulée **wutiko** dont la première version Web a été lancée en 2015. Cette application permet de rechercher des emplois en ligne. Elle possède aussi un annuaire d'entreprises pour les utilisateurs et une banque de curriculum vitae (cv) pour les recruteurs. Mon stage s'intègre dans un projet de refonte totale du système de recherche d'emploi en ligne existant appelé **wutiko**.

Il s'inscrit sur la création d'une nouvelle architecture basée sur les services web REST et la réalisation d'une application permettant aux utilisateurs finaux de rechercher des emplois en ligne.

2. La recherche d'emploi en ligne

De nos jours, trouver un emploi fiable est devenu une nécessité pour chaque personne. Ainsi différentes méthodes de recherche d'emploi ont été mises en place. La première se fait de façon aléatoire : le demandeur va d'une entreprise à une autre pour déposer son curriculum vitae et sa lettre de motivation. Ceci ne permet pas aux demandeurs de pouvoir faire un suivi de leur dossier. La deuxième en est que chaque entreprise doit poster ses offres d'emploi dans son site web. Ceci ne permet pas aussi aux demandeurs de savoir à quel moment un emploi est posté. La troisième en est que les emplois sont postés sur des sites dédiés à l'emploi. Ainsi les demandeurs peuvent aller directement au niveau de ces sites pour postuler aux offres. Ils pourront faire un suivi régulier de leur demande.

3. Les services web et REST

Les services Web (en anglais Web services) représentent un mécanisme de communication entre applications distantes à travers le réseau internet indépendant de tout langage de programmation et de toute plate-forme d'exécution. Les requêtes et les réponses sont soumises à des standards et sont normalisées. Un service Web va pouvoir être consommé via une interface de programmation, alors qu'un simple site Web par opposition, est fait pour être utilisé par un être humain. Un des avantages est que le service Web ne dépend pas du langage avec lequel il a été créé. Cela permet de découper son application en plusieurs briques qui peuvent chacune utiliser un langage différent et être hébergées sur des serveurs différents. Grâce à la normalisation des requêtes et des réponses, ces différents services pourront très bien communiquer ensemble. De plus, l'interface d'un service Web est décrite de manière à être interprétée par les machines, ce qui permet aux applications clientes d'accéder aux services de manière automatique.

Il existe aujourd'hui deux styles architecturaux pour concevoir un service Web :

- **SOAP (Simple Object Access Protocol)** : C'est un protocole créé par Microsoft permettant d'invoquer des méthodes sur des services distants. Il est basé sur le langage XML. Le transfert d'informations peut se faire en HTTP mais également par un autre protocole, comme SMTP par exemple ;
- **REST (Representational State Transfer)** : REST décrit un style d'architecture logicielle permettant de construire une application devant fonctionner sur des systèmes distribués, typiquement internet. L'auteur de cette description n'est autre que Roy Fielding dans sa thèse [Fie00], un des principaux rédacteurs de la norme HTTP 1.1. REST est aujourd'hui souvent préféré à SOAP car il est plus léger et plus simple à mettre en place. Une autre raison est également le fait que SOAP soit défini pour retourner du XML. Pour des applications en Javascript, Ruby ou Python, le format JSON est généralement préféré au XML car plus simple à manipuler. On constate clairement sur le Schéma 1 ci-dessous une augmentation de l'intérêt pour le REST, principalement depuis 2011. Le protocole SOAP est quant à lui en légère diminution au fil des années. Il montre aussi l'importance de REST aujourd'hui dans le contexte des services Web.

Schéma 1 : Évolution des recherches Google sur les mots clés REST (en rouge) et SOAP (en bleu), Source : [urla]

1.2 Problématique

En 2015, Linked Partners a mis en ligne sa première application Web de recherche d'emploi en ligne intitulée **wutiko**. Pour satisfaire les besoins des millions d'utilisateurs connectés à internet via leur téléphone portable ou tablette, l'entreprise entame au cours de cette même année le développement de l'application mobile. Ainsi elle se heurte à de nombreux problèmes comme la cohérence des données

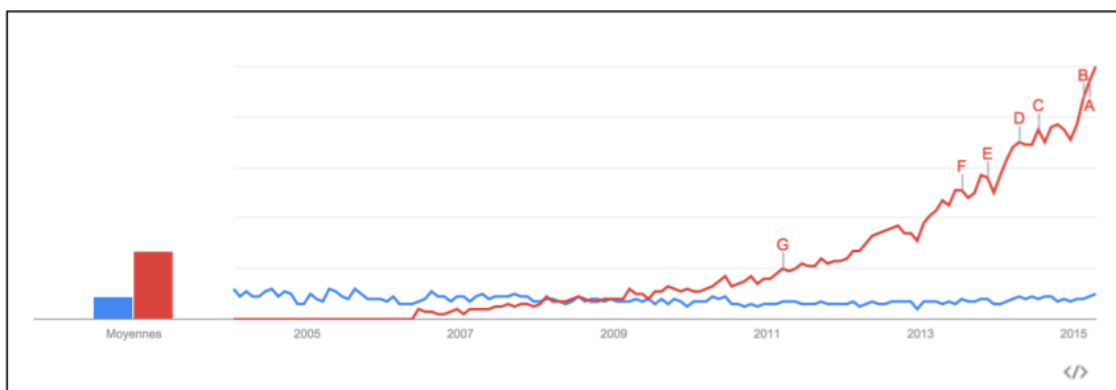


FIGURE 1.1 – Évolution des recherches Google sur les mots clés REST et SOAP

à récupérer ou à envoyer dans la base de données car les deux applications clientes tournent sur des plateformes différentes. Mais aussi, chaque changement effectué sur l'une des plateformes nécessite le même travail sur l'autre. Ce qui est coûteux en temps.

Ainsi, dans l'optique de résoudre tous ces problèmes, nous avons décidé de travailler avec une architecture REST qui n'est rien d'autre que de développer toutes les fonctionnalités de l'application dans une seule partie que nous allons appeler tout au long de notre mémoire le **backend**. Ce dernier fournira un ensemble d'interfaces afin que les applications clientes se connectent pour utiliser les ressources. Cette partie cliente sera appelée **frontend** tout au long de notre mémoire.

Afin de pouvoir répondre convenablement aux besoins des utilisateurs nous allons essayer de suivre ces différentes étapes :

- faire une étude des services Web REST ;
- faire l'état de l'existant ;
- proposer une nouvelle architecture ;
- implémenter un service Web REST proposant un système de recherche d'emploi en ligne ;
- implémenter un moteur de recherche pour faire des recherches rapides sur les ressources disponibles ;
- implémenter dans un premier temps un client web utilisant l'ensemble des ressources de notre service Web REST ;
- implémenter dans un second temps un client mobile utilisant l'ensemble des ressources de notre service Web REST.

1.3 Méthodologie utilisée

Pour mener à bien notre mission, nous avons choisi de travailler avec la méthode **2TUP** qui signifie « 2 Track Unified Process ». Cette méthode apporte une réponse aux contraintes de changement continu imposées aux systèmes d'information des organisations. En ce sens, il renforce le contrôle sur les capacités d'évolution et de correction de tels systèmes. Il s'articule sur trois branches comme l'illustre le schéma 2 ci-dessous :

- **La branche gauche (fonctionnelle)** : Les principales étapes de la branche fonctionnelle se présentent comme suit :
 - l'étape capture des besoins fonctionnels : produit le modèle des besoins focalisé sur le métier

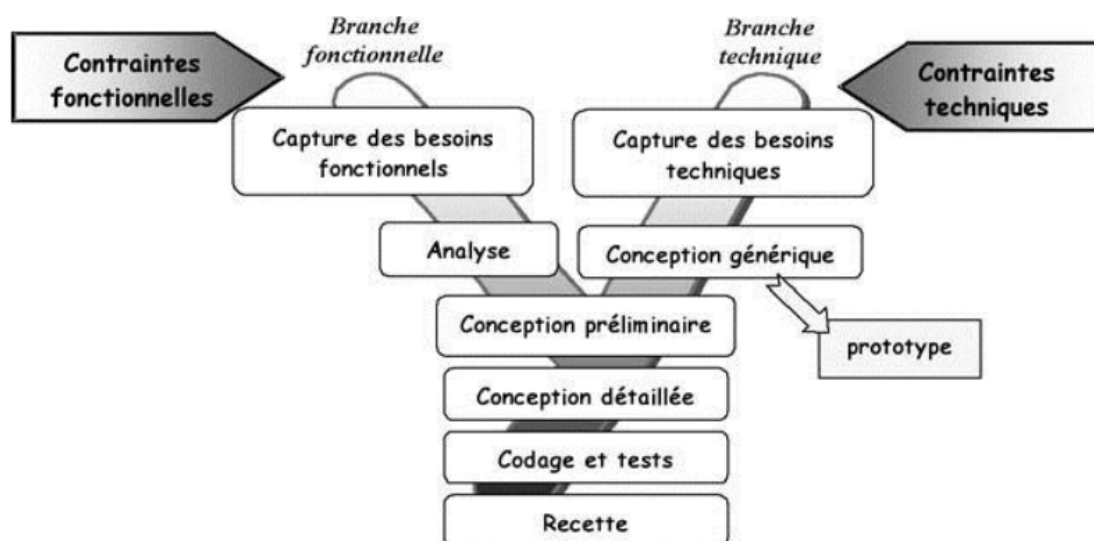


FIGURE 1.2 – Processus de développement en Y

des utilisateurs. Elle qualifie, au plus tôt le risque de produire un système inadapté aux utilisateurs ;

- l'étape d'analyse consiste à étudier précisément les spécifications fonctionnelles de manière à obtenir une idée de ce que va réaliser le système en terme de métier.
- **La branche droite (technique)** : capitalise un savoir-faire technique pour le système indépendamment des fonctions à réaliser. Cette branche comporte les étapes suivantes :
 - la capture des besoins techniques qui spécifie les contraintes, les besoins non fonctionnels et les choix conditionnant la conception du système ;
 - la conception générique consiste à construire l'architecture technique du système qui doit être la moins dépendante possible des aspects fonctionnels. Cela revient à construire le squelette du système en décrivant les composants nécessaires et leurs interactions. Cette étape se concrétise par la production d'un prototype.
- **La branche du milieu** : à l'issue des évolutions du modèle fonctionnel et de l'architecture technique, la réalisation du système consiste à fusionner les résultats des deux branches citées précédemment. Cette branche comporte les étapes suivantes :
 - la conception préliminaire qui consiste à intégrer le modèle d'analyse dans l'architecture technique ;
 - la conception détaillée qui définit comment réaliser chaque composant ;
 - le codage et les tests des composants.

Schéma 2 : Processus de développement en Y, source : [urlb]

1.4 Plan du mémoire

Le travail est divisé en quatre parties :

Dans la première partie, nous avons fait une étude et conception de services Web REST. Elle est basée sur l'architecture existante et les composants logiciels utilisés. A travers cette étude, des critiques ont été

émis sur ces derniers. Nous avons abordé aussi dans cette partie la conception de services Web REST afin de définir les principes directeurs d'une bonne architecture REST.

Dans la seconde partie, nous avons fait une conception de l'application cliente à travers une spécification des besoins, la mise en place des diagrammes de cas d'utilisation et de classes, et une conception générale présentant la nouvelle architecture.

Et enfin nous avons implémenté nos services Web REST pour l'application **wutiko** et aussi une implémentation de la partie Web capable d'utiliser les ressources de ces services puis nous avons procédé à une présentation de l'application.

Chapitre 2 : Etude et conception de services Web REST

2.1 Étude de l'existant

2.1.1 Environnement logiciels.

Dans cette partie, nous allons parler des outils logiciels utilisés pour la création de l'application existante.

Les différentes outils et logiciels utilisés pour le déploiement de l'application sont :

- **système d'exploitation** : Le système d'exploitation utilisé est Microsoft Windows 8 Professionnel ;
- **serveur d'application** : Apache est un serveur d'application certifié pour le déploiement d'application Web. Ainsi , il a servi d'hébergeur pour l'application Web ;
- **outil de gestion et d'administration de base de données** : WampServer est un logiciel de gestion et d'administration de base de données. Il contient un serveur de base données appelé MySQL où sont stockées nos données.
- **Langage de programmation** :
 - **PHP**

PHP est tout d'abord un langage de script interprété, gratuit, Open Source et distribué sous une licence autorisant la modification et la redistribution. Il a été utilisé pour créer une interaction entre le client et la base de données ;
 - **HTML**

HTML (Hyper Text Markup Language / langage hypertexte) est le langage dans lequel sont écrites les pages du Web. Un site Web est constitué d'un ou plusieurs documents HTML, appelées aussi pages. Pour se déplacer d'une page à l'autre dans nos modules on passe par l'intermédiaire d'hyperliens. Pour ajouter des objets graphiques on utilise HTML d'autre part pour tester des pages Web HTML en local, il suffit d'ouvrir le fichier dans un navigateur. HTML n'est pas un langage de programmation comme le C++. Les langages dynamiques comme PHP et Javascript vont d'ailleurs générer des pages HTML statiques ;
 - **Java script**

JavaScript est un langage de programmation de scripts principalement utilisé pour les pages Web interactives comme les pages HTML. JavaScript est exécuté sur l'ordinateur de l'internaute par le navigateur lui-même. C'est une extension du langage HTML qui est incluse dans le code. C'est un langage de programmation qui permet d'apporter des améliorations au langage HTML en permettant d'exécuter des commandes. Ce code est directement écrit dans la page HTML, c'est un langage peu évolué qui ne permet aucune confidentialité au niveau des codes ;
 - **CSS**

Les CSS, Cascading Style Sheets (feuilles de styles en cascade), servent à mettre en forme des documents Web, type page HTML ou XML. Par l'intermédiaire de propriétés d'apparence (couleurs, bordures, polices, etc.) et de placement (largeur, hauteur, côte à côte, dessus, dessous, etc.), le rendu d'une page web peut être intégralement modifié sans aucun code supplémentaire dans la page Web. Les feuilles de styles ont d'ailleurs pour objectif principal de dissocier le contenu de la page de son apparence visuelle.

2.1.2 Les critiques de l'existant.

Nous allons présenter dans cette partie l'architecture de l'application **wutiko** (voir schéma 3 ci-dessous). Cette architecture se présente en deux parties :

- web : une partie web, développé avec PHP ;
- mobile : une partie mobile, développé avec Android.

Chacune de ces parties permet de mettre en place un système de recherche d'emplois en ligne appelé **wutiko**. Mais l'implémentation de ce système est différente au niveau chacune des plates-formes Web et mobile. La partie web communique avec la base de données à travers des requêtes basées sur le langage sql. Par contre l'autre partie communique avec la base de données à travers des flux **json** qui sont sérialisés ou désérialisés via le processus. Par exemple supposons que les développeurs ont à ajouter une nouvelle fonctionnalité comme un réseau social dans chacune des plates-formes. Le temps pour ajouter cette nouvelle fonctionnalité est le même pour chacune des parties, ce qui est coûteux en terme de temps. Étant donné que Linked Partners est une start-up avec des ressources humaines très limitées, avec la concurrence très rude, nous nous devons de mettre en place une application fiable en un temps optimal.

De plus, imaginons qu'on est à ajouter un utilisateur au niveau de chaque application cliente comme l'illustre le schéma ci-dessous. Le client Web insère dans la base de données trois paramètres(id, prénom, adresse) et pour le client mobile(id, prénom, nom). Chacun des clients a omis un paramètre pour insérer des données dans la table utilisateur et l'insertion a bien réussi à défaut de contraintes dans cette table. Si nous suivons cette logique la liste des utilisateurs qui est affiché au niveau de chaque client sera incohérente du fait de la présence de champs vides pour certains et pas pour d'autres. Cette omission sera fatale pour le système. Si le problème est rencontré sur toutes les autres tables, le système d'information sera non fiable et nous risquons de perdre tous nos utilisateurs au profit des autres.

L'autre cas en est que nous disposons d'un système de gestion de base de données (SGBD) appelé **WampServer**. Ce système est composé d'une base de données **mysql** stockant toutes nos données et d'un serveur d'application où tourne l'application. Il est accessible via une adresse ip par exemple 192.168.10.132. De ce fait, imaginons qu'un utilisateur ait accès à l'application, il aura accès aussi à la base des données car ces deux parties ont la même adresse. Tout en sachant que la partie données est la plus importante dans notre système d'information, nous risquons d'ouvrir toutes les données aux utilisateurs. Ainsi, l'application présente quelques failles de sécurité qui risquent de violer les règles de confidentialité.

Pour palier à ces types de problèmes et réduire le temps de développement, nous allons proposer une architecture regroupant l'ensemble des fonctionnalités communément appelé REST utilisable par ces deux parties (**web et mobile**) par le biais d'une communication de flux json.

Schéma 3 : Cas d'anomalie de l'architecture wutiko

2.2 Conception de services Web REST

2.2.1 Définition.

REST est l'acronyme de Representational State Transfer. REST décrit un style d'architecture logicielle permettant de construire une application devant fonctionner sur des systèmes distribués, typiquement internet. L'auteur de cette description n'est autre que Roy Fielding dans sa thèse [Fie00], un des principaux rédacteurs de la norme HTTP 1.1.

2.2.2 Principes directeurs.

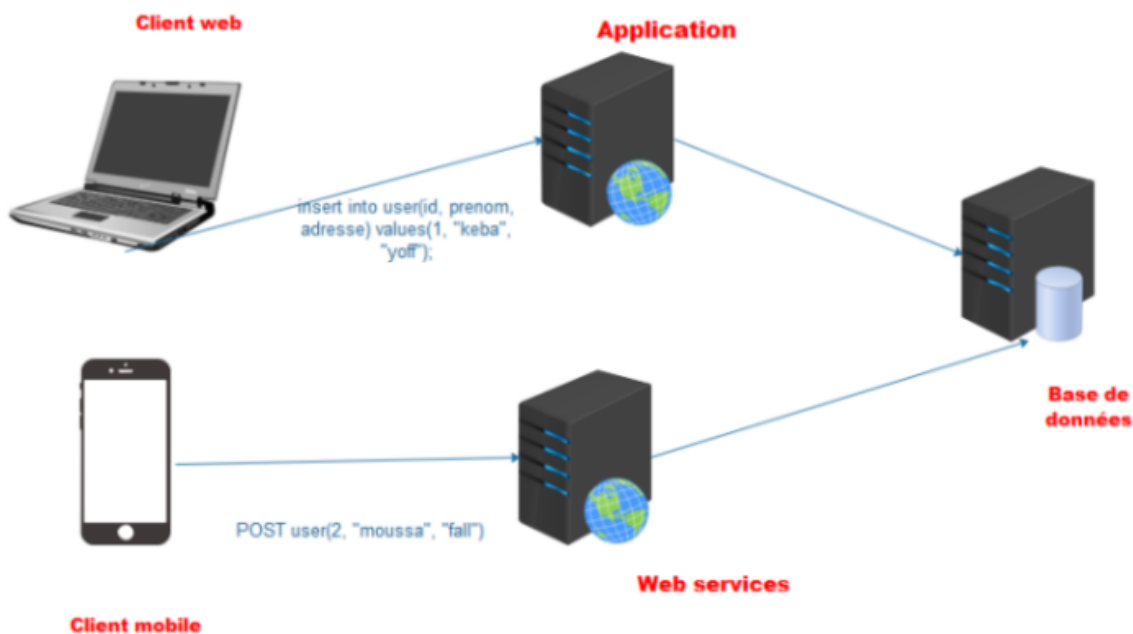


FIGURE 2.3 – Cas d’anomalie de l’architecture wutiko

La conception d’un service Web REST implique l’utilisation des éléments suivants :

- une architecture client-serveur ;
- des requêtes sans état (2 requêtes d’un client sont indépendantes) ;
- l’utilisation de mécanismes de cache possible ;
- une interface uniforme.

Architecture client-serveur

L’architecture **client-serveur** comme l’illustre le schéma 4 ci-dessous désigne un mode de communication entre plusieurs programmes :

- un client envoie des requêtes ;
- un serveur attend les requêtes des clients et y répond .

Par extension, le client désigne également l’ordinateur sur lequel est exécuté le logiciel client, et le serveur, l’ordinateur sur lequel est exécuté le logiciel serveur.

Requête stateless (sans état)

Conformément à l’architecture REST, le serveur ne stocke aucun état concernant la session cliente. Cette restriction est appelée stateless. Ainsi par définition, une requête stateless signifie que chaque requête cliente au serveur doit contenir toutes les informations nécessaires pour que ce dernier puisse interpréter la demande. L’état de la session est donc entièrement gardé sur le client. Le client est responsable du stockage et de la gestion de toutes les informations liées à l’état de la demande côté client. Le serveur ne s’appuie jamais sur des informations provenant des requêtes précédentes.

Il existe des avantages remarquables pour disposer d’un service Web REST sans état :

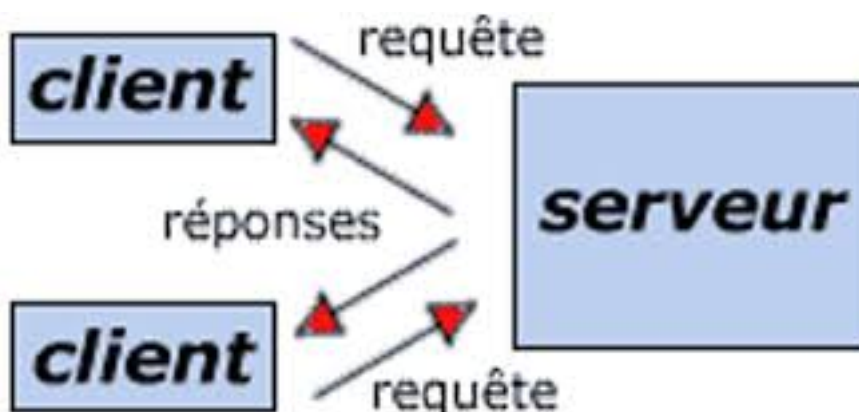


FIGURE 2.4 – Architecture client-serveur

Schéma 4 : Architecture client-serveur, Source : [urlc]

- une requête sans état permet d'étendre notre service Web à des millions d'utilisateurs simultanés en le déployant sur plusieurs serveurs. Le serveur peut gérer toutes les requêtes car il n'y a pas de dépendances liées à la session ;
- le service Web REST est moins complexe en supprimant toute la logique de synchronisation de l'état côté serveur ;
- il est également facile de mettre en cache le résultat d'une requête.

Mécanisme de cache

La mise en cache est la possibilité de stocker des copies de données fréquemment consultées lors d'une conversation entre un client et un serveur. Lorsqu'un client demande des ressources, la requête passe par un cache ou une série de caches (cache local, cache proxy ou proxy inverse) vers le service hébergeant la ressource. Si l'un des caches le long du chemin de la demande possède une nouvelle copie de la ressource demandée, elle utilise cette copie pour satisfaire la demande. Si aucune des caches ne peut satisfaire la demande, elle se déplace complètement vers le service Web (ou le serveur d'origine tel qu'il est formellement connu). Les caches peuvent prendre une copie d'une réponse le long du chemin de la réponse, mais seulement si les métadonnées de mise en cache leurs permettent de le faire.

L'optimisation du réseau à l'aide de la mise en cache améliore la qualité de service globale de notre système de la manière suivante :

- réduire la bande passante ;
- réduire la latence ;
- réduire la charge sur les serveurs ;
- masquer les pannes du réseau.

Interface uniforme

Dans sa dissertation Fielding [Fie00] définit une interface uniforme comme quatre contraintes qui sont :

- l'identification des ressources : cela indique essentiellement qu'une demande devra être identifiée par les ressources qu'elle recherche (via un URL) ;

- la manipulation des ressources par des représentations : cela indique essentiellement que lorsqu'un client accède à une ressource donnée, ainsi que toutes les métadonnées, il devrait avoir suffisamment d'informations pour modifier ou supprimer la ressource ;
- un message auto-descriptif : cela indique qu'un message doit contenir suffisamment d'informations pour permettre au serveur de savoir comment le traiter (c'est-à-dire le type de demande, les types de mime, etc.) ;
- l'hypermédia comme moteur de l'état d'application : cela indique que l'accès au service Web devrait être similaire à l'accès à une page Web(c'est-à-dire découvrir d'autres fonctionnalités du service Web comme un utilisateur qui est sur une page et devrait cliquer sur un lien).

C'est la différence principale par rapport aux services Web précédents(SOAP, SOA) : tout élément offert à la manipulation par l'application est nommé ressource et est identifié de manière unique. HTTP définit les Identifiants de Ressource Uniforme (URI ci-après) suivant l'exemple ci-dessous :

- **http** décrit le protocole utilisé pour l'échange d'informations ;
- **host** décrit le nom de domaine de l'application ;
- **port** décrit le port utilisé pour accéder à l'information ;
- **path** décrit le chemin définit pour se connecter à l'interface ;
- **?query** décrit la ressource identifiée.

URI = "http : " "/" host [" : " port] [path [" ? " query]]

Exemple : Syntaxe d'une URL

2.2.3 Actions sur les ressources.

Les différentes actions possibles sur ces ressources sont données par les différents types de requêtes HTTP, principalement :

- GET ;
- POST ;
- PUT ;
- DELETE.

On manipule des représentations de ressources par les ressources directement. Les ressources sont donc encodées selon un format spécifique dans les messages HTTP. Au lieu d'être incluse dans un message, une ressource peut être référencée par un hyperlien.

REST n'est pas un protocole, il n'existe donc pas de norme en tant que telle, mais plutôt des conventions de codage respectant les principes cités ci-dessus. Pour un protocole applicatif respectant ces principes on parlera d'**implémentation RESTful**, et comme exemple de réalisation, un service Web utilisant HTTP sur ces principes sera également qualifié de RESTful.

2.2.4 Principes d'implémentation.

Pour définir un service Web REST, les étapes suivantes doivent être suivies :

- définition des ressources manipulées, collection de ressources, ou ressource unique ;
- codage de la représentation des ressources (les attributs d'une ressource et le format qui va être utilisé)
- sémantique des messages (les actions possibles sur les ressources sont indiquées par les messages du protocole de transport).

1. Sémantique

Les différentes actions possibles sont définies par les verbes HTTP :

- GET : récupération d'une ressource ou d'une liste de ressources ;
- PUT : mise à jour d'une ressource existante, création d'une ressource en spécifiant l'URI de la ressource ;
- POST : création d'une sous ressource (le serveur décide de l'URI), ajout d'informations à une ressource existante ;
- DELETE : suppression d'une ressource ;
- HEAD : informations sur une ressource.

Une ressource donnée ne sera pas obligatoirement manipulable par tous les messages. Par exemple, une ressource accessible en lecture seulement peut n'être accessible que par les messages de type GET.

2. Génération d'un message

Pour générer un message, il faut suivre les étapes suivantes :

- définir la ou les ressources visée (s) (URI collection) ;
- requête : définir l'action demandée, et donc le type de message (GET, PUT, ...);
- réponse : décider du code d'erreur ;
- la représentation de la ressource : ce codage se prête bien à une programmation orientée objet et à l'utilisation de bibliothèques de sérialisation/désérialisation.

3. Représentation des ressources

Que ce soit au niveau du serveur ou au niveau des clients, un service Web REST manipule des ressources par une représentation de celles-ci. Le principe en pratique est de passer de la représentation utilisée en interne, que ce soit sur le client ou le serveur, à la représentation utilisée dans le message HTTP. Cette opération s'appelle la sérialisation. De ce fait, pour mettre en place une représentation nous devons suivre les cinq étapes ci-dessous.

(a) Sérialisation

C'est le processus visant à coder l'état d'une information qui est en mémoire sous la forme d'une suite d'informations plus petites (atomiques), le plus souvent des octets voire des bits. Cette suite pourra être utilisée pour la sauvegarde (persistance) ou le transport sur le réseau (proxy, RPC...).

(b) Désérialisation

L'activité symétrique, visant à décoder cette suite pour créer une copie conforme de l'information d'origine, s'appelle la désérialisation (ou unmarshalling).

(c) Complexité

D'apparence simple, ces opérations posent en réalité un certain nombre de problèmes, comme la gestion des références entre objets ou la portabilité des encodages. Les choix entre les diverses techniques de sérialisation ont une influence sur les critères de performances comme la taille des suites d'octets sérialisées ou la vitesse de leur traitement.

(d) Format

Comme spécifié plus haut, les messages d'un service Web REST sont indépendants les uns des autres, ce qui implique que le codage de la représentation des ressources peut être différent entre deux messages et donc qu'il faut spécifier dans le message le codage utilisé. Le format d'encodage des messages diffère selon les cas d'utilisation des services Web REST mais aussi selon les applications clientes utilisées. Dans notre cas, nous allons utiliser le format **JSON** du fait de sa simplicité de mise en œuvre.

(e) JavaScript Object Notation : JSON

Raison du succès

Pour les services Web REST, les encodages les plus utilisés sont XML et JSON. XML est un standard incontesté mais souffre de quelques inconvénients :

- verbeux ;
- difficilement lisible par un humain ;
- dualité entre les attributs et les éléments.

JSON est un format texte qui permet de représenter des données et de les échanger facilement à l'instar d'XML.

Description du langage

JSON permet de décrire le modèle objet de JavaScript grâce à deux types de structures disponibles :

- objet : une collection de paires nom/valeur (un tableau associatif) ;
- tableau : une liste ordonnée de valeurs.

Les valeurs peuvent être des types suivants : booléen, chaîne de caractères, nombre, ou valeur nulle ...

Le format JSON ne fait pas de différence entre attribut et élément comme en XML. Il est donc moins sujet à variation, et est globalement plus concis et plus lisible. Les valeurs sont typées alors qu'en XML on ne dispose que de chaînes de caractères.

La syntaxe est celle de JavaScript et voici en guise d'illustration l'exemple ci-dessous qui montre une description d'un format JSON d'une compagnie donnée avec ses attributs et ses valeurs :

```
[
  {
    "id": 1,
    "address": "goumel",
    "country": "Senegal",
    "region": "ziguinchor",
    "name_company": "wutiko",
    "full_name": "wutiko SA",
    "founded_at": "1989-1245",
    "logo": "http://localhost:8000/backend/upload_files/1482163728_4900482_14958988_1773402122918726_549427011_o.jpg",
    "cover_photo": "http://localhost:8000/backend/upload_files/1482163728_5295305_14958988_1773402122918726_549427011_o.jpg",
    "latitude": -125456.235,
    "longitude": 1245878.32,
    "reach": "international",
    "size_range": "12642345-655656",
    "website": "www.wutiko.com",
    "modified": "2016-12-19",
    "is_active": true,
    "total_revenu": "2124577498-5158478",
    "is_always_open": true,
    "legal_id": 1,
    "sector": 1,
    "myuser_c": []
  }
]
```

Exemple : Exemple de description en format JSON d'une compagnie

2.2.5 Écosystème.

Dans cette section, nous allons aborder les techniques et composants logiciels fréquemment utilisés dans la conception de services Web REST.

1. HTTP

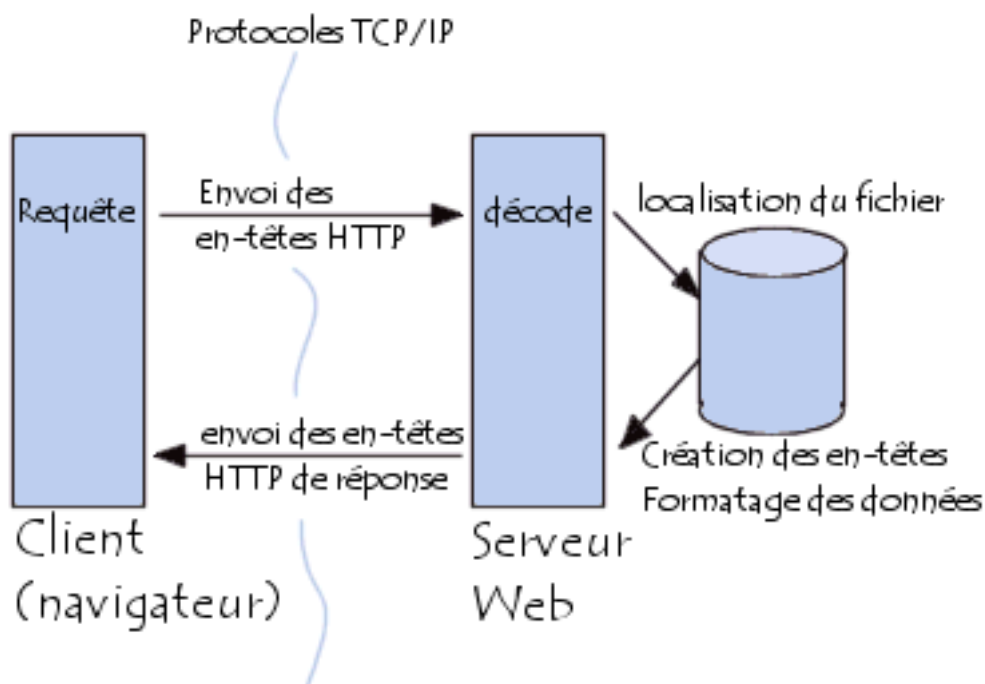


FIGURE 2.5 – Communication entre navigateur et serveur

Le protocole HTTP (HyperText Transfer Protocol) est le protocole le plus utilisé sur Internet depuis 1990. La version 0.9 était uniquement destinée à transférer des données sur Internet (en particulier des pages Web écrites en HTML). La version 1.0 du protocole (la plus utilisée) permet désormais de transférer des messages avec des en-têtes décrivant le contenu du message en utilisant un codage de type MIME. C'est la caractéristique principale de l'architecture REST, il joue un rôle central, pas seulement dans le nommage des ressources (URI) et la sémantique des messages entre un client et un serveur. Le schéma 5 ci-dessous illustre une communication entre un navigateur (client) et un serveur utilisant le protocole HTTP comme moyen d'échange de données entre ces deux entités. Cette communication peut être représentée en trois parties :

- le navigateur effectue une **requête HTTP** avec des en-têtes ;
- le serveur traite la requête puis se charge de la localisation des fichiers ;
- après la localisation des fichiers, les données sont formatées puis retournées au serveur qui se charge de l'envoi de la **réponse HTTP** avec les en-têtes correspondant à la requête.

Schéma 5 : Communication entre navigateur et serveur, Source : [urlc]

Requête HTTP

Une requête HTTP est un ensemble de lignes envoyé au serveur par le navigateur. Elle comprend :

- **une ligne de requête** : c'est une ligne précisant le type de document demandé, la méthode qui doit être appliquée, et la version du protocole utilisée. La ligne comprend trois éléments devant être séparés par un espace :
 - la méthode ;
 - l'URL ;
 - la version du protocole utilisé par le client (généralement HTTP/1.0).
- **les champs d'en-tête de la requête** : il s'agit d'un ensemble de lignes facultatives permettant de donner des informations supplémentaires sur la requête et/ou le client (Navigateur,

système d'exploitation, ...). Chacune de ces lignes est composée d'un nom qualifiant le type d'en-tête, suivi de deux points (:) et de la valeur de l'en-tête ;

- **le corps de la requête** : c'est un ensemble de lignes optionnelles devant être séparées de lignes précédentes par une ligne vide et permettant par exemple un envoi de données par une commande POST lors de l'envoi de données au serveur par un formulaire.

Réponse HTTP

Une réponse HTTP est un ensemble de lignes envoyées au navigateur par le serveur. Elle comprend :

- **une ligne de statut** : c'est une ligne précisant la version du protocole utilisé et l'état du traitement de la requête à l'aide d'un code et d'un texte explicatif. La ligne comprend trois éléments devant être séparés par un espace :
 - la version du protocole utilisé ;
 - le code de statut ;
 - la signification du code.
- **les champs d'en-tête de la réponse** : il s'agit d'un ensemble de lignes facultatives permettant de donner des informations supplémentaires sur la réponse et/ou le serveur. Chacune de ces lignes est composée d'un nom qualifiant le type d'en-tête, suivi de deux points (:) et de la valeur de l'en-tête ;
- **le corps de la réponse** : il contient le document demandé.

2. Ajax

Ajax repose sur l'utilisation de la fonction JavaScript XMLHttpRequest, qui permet à un navigateur web de générer une requête HTTP à partir d'un événement JavaScript, et ce de manière asynchrone. Cela permet de créer des interfaces web qui interrogent le côté serveur tout en continuant à répondre aux sollicitations de l'utilisateur, se rapprochant des interfaces classiques.

Chapitre 3 : Conception d'une application cliente pour la recherche d'emploi en ligne

3.1 Spécification des besoins

3.1.1 Les besoins fonctionnels.

Les besoins fonctionnels ou besoins métiers représentent les actions que le système doit exécuter, il ne devient opérationnel que s'il les satisfait.

Notre application doit couvrir principalement les besoins fonctionnels suivants :

- gestion des utilisateurs ainsi que les droits d'accès : il permet aux utilisateurs du système de :
 - créer leur compte en ligne ;
 - s'authentifier par adresse mail ou par les réseaux sociaux (Facebook, Google, Twitter...);
 - mettre à jour leur profil.
- gestion des curriculums vitae (cv) en ligne : il s'agit de fournir une vue aux recruteurs afin de consulter la banque de curriculums vitae, de les filtrer en fonction de leurs besoins ;
- gestion des offres : il permet aux utilisateurs du système de :
 - consulter aux offres disponibles ;
 - postuler à autant d'emplois qu'ils désirent ;
 - consulter ses emplois favoris.
- gestion des emplois : il permet aux recruteurs du système de :
 - créer des emplois en ligne ;
 - consulter tous les emplois qu'ils ont créé ;
 - modifier les emplois.
- gestion des entreprises : il permet aux utilisateurs du système de :
 - créer des entreprises en ligne ;
 - disposer d'un annuaire des entreprises ;
 - consulter ses entreprises favoris.
- gestion du réseau social : il s'agit de fournir une vue aux utilisateurs permettant de créer des liens d'amitié et aussi de publier des articles (images, vidéos, texte ...);
- Un système permettant aux utilisateurs de faire des recherches rapides sur toutes les ressources (utilisateurs, emplois, entreprises).

3.1.2 Les besoins non fonctionnels.

Ce sont des exigences qui ne concernent pas spécifiquement le comportement du système mais plutôt identifient des contraintes internes et externes du système.

Notre application doit nécessairement assurer ces besoins :

- **l'extensibilité** : l'application devra être extensible, c'est-à-dire qu'il pourra y avoir une possibilité d'ajouter ou de modifier de nouvelles fonctionnalités. De ce fait, le code doit être clair pour permettre des futures évolutions ou améliorations ;
- **la sécurité** : l'application doit respecter la confidentialité des données c'est-à-dire que le site web est accessible par un identifiant et un mot de passe ;
- **l'interface** : avoir une application qui respecte les principes des Interfaces Homme/Machine (IHM) tels que l'ergonomie et la fiabilité ;
- **la performance** : l'application devra être performante c'est-à-dire que le système doit réagir

- dans un délai précis, quelque soit l'action de l'utilisateur ;
- **la convivialité** : l'application doit être simple et facile à manipuler même par des non experts ;
- **l'ergonomie** : le thème adopté par l'application doit être inspiré des couleurs et du logo de notre entreprise ;
- garantir l'intégrité et la cohérence des données à chaque mise à jour et à chaque insertion.

3.2 Diagramme des cas d'utilisation

Le diagramme des cas d'utilisation montre les interactions fonctionnelles entre les acteurs et le système. Les différents éléments qui interviennent à la conception d'un diagramme des cas d'utilisation sont :

- **Acteur** : c'est le rôle joué par un utilisateur humain ou un autre système qui interagit directement avec le système étudié. Un acteur participe à au moins un cas d'utilisation.
- **Cas d'utilisation** : c'est un ensemble de séquences d'actions réalisées par le système produisant un résultat observable intéressant pour un acteur particulier. Collection de scénarios reliés par un objectif utilisateur commun.
- **Association** : elle est utilisée dans ce type de diagramme pour relier les acteurs et les cas d'utilisation par une relation qui signifie simplement " participe à " .
- **Inclusion** : on le retrouve sous le nom de **include** dans la conception. C'est le cas d'utilisation de base qui incorpore explicitement un autre, de façon obligatoire, à un endroit spécifié dans ses enchaînements.
- **Extension** : on le retrouve sous le nom de **extends** dans la conception. C'est le cas d'utilisation de base qui incorpore implicitement un autre, de façon optionnelle, à un endroit spécifié indirectement dans celui qui procède à l'extension.
- **Généralisation** : C'est les cas d'utilisation descendants qui héritent de la description de leur parent commun. Chacun d'entre eux peut néanmoins comprendre des relations spécifiques supplémentaires avec d'autres acteurs ou cas d'utilisation.

Dans notre cas comme le décrit le schéma 6 ci-dessous, nous avons deux acteurs qui sont :

- **Utilisateur** : C'est l'élément central de notre système. Il est chargé d'effectuer toutes les tâches clientes. Ces tâches clientes ne sont rien d'autre que la relation entre l'utilisateur et les cas d'utilisation définit dans le schéma 6 ci-dessous.
- **Recruteur** : Il est chargé de créer des entreprises, des emplois et d'avoir à sa disposition tous les profils des utilisateurs.
- **Administrateur** : Il est chargé d'effectuer tous les tâches métiers avec l'ajout, la modification, la récupération et la suppression de fonctionnalités.

Le schéma 6 ci-dessous définit aussi les cas d'utilisation préliminaires de notre système. Ces cas d'utilisations sont :

- **Authentification** : permet d'identifier chaque utilisateur, et de lui donner l'accès aux fonctionnalités propices.
- **Gestion des utilisateurs et les droits d'accès** : permet à l'utilisateur de créer un compte afin de s'authentifier, de mettre à jour son profil et de consulter la liste des utilisateurs.
- **Gestion des curriculums vitae** : permet à un recruteur de lister les curriculums vitae.
- **Gestion des emplois** : permet aux recruteurs de créer, modifier, lister des emplois.
- **Gestion des offres d'emploi** : permet à l'utilisateur de consulter et d'appliquer aux offres d'emploi disponibles.
- **Gestion des entreprises** : permet à l'utilisateur de créer, modifier, lister des entreprises et aussi postuler à ses entreprises.

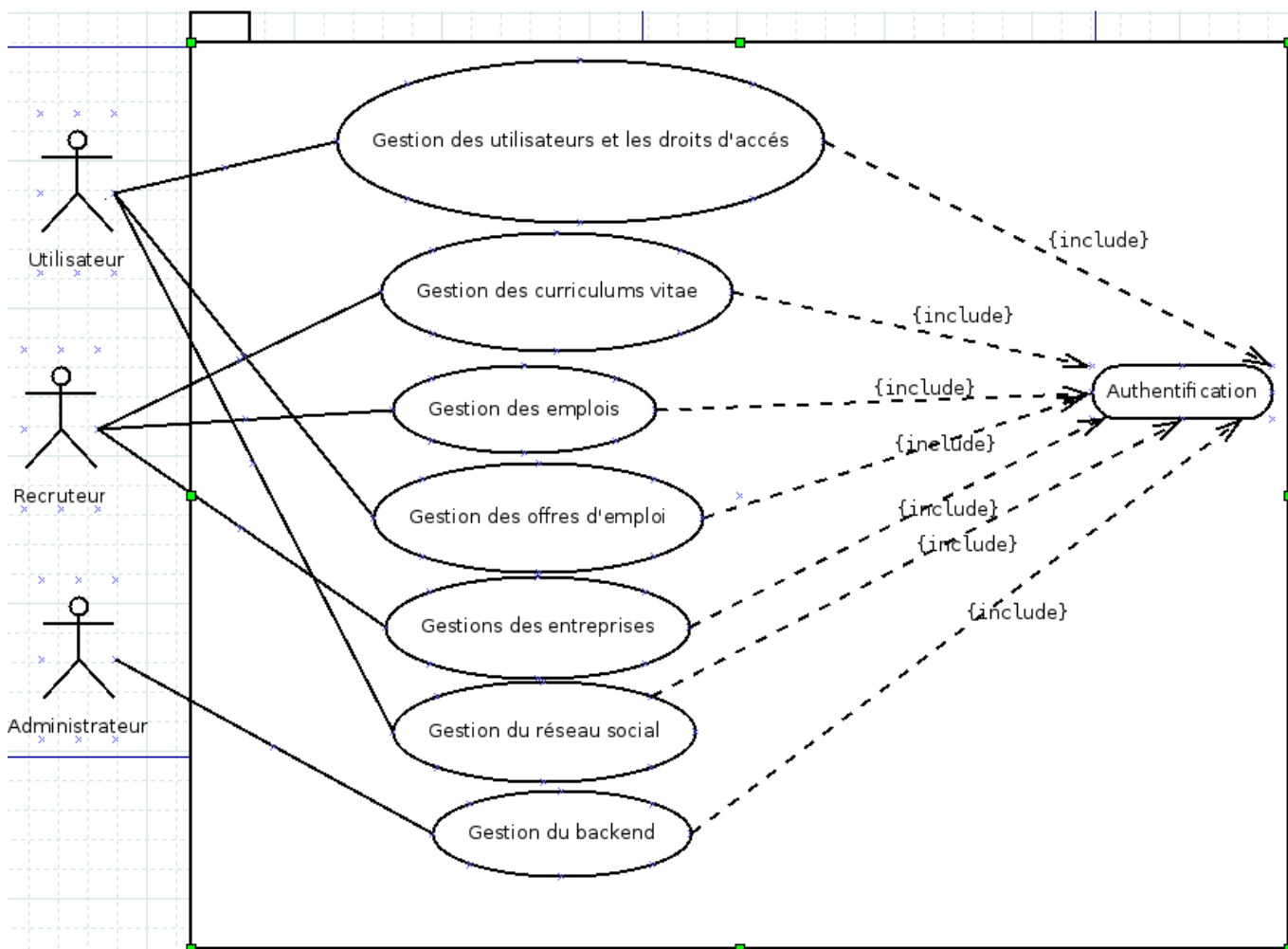


FIGURE 3.6 – Diagramme des cas d'utilisations

- **Gestion du réseau social** : permet à l'utilisateur de créer des liens d'amitié avec d'autres utilisateurs et aussi de publier des articles, images, vidéos...
- **Gestion du backend** : permet à l'administrateur de créer, modifier, consulter, supprimer des fonctionnalités.

Schéma 6 : Diagramme des cas d'utilisations

3.2.1 Description des cas d'utilisation.

1. Authentification

Nous allons présenter le cas d'utilisation **Authentification** à partir du schéma 7 ci-dessous. Ce schéma présente les acteurs et les cas d'utilisation qui vont être décrits.

Description textuelle

- **Acteurs** : Les différents acteurs du cas d'utilisation Authentification sont l'utilisateur, le recruteur et l'administrateur.
- **Description** : Tous les utilisateurs de l'application ou l'administrateur ou les recruteurs peuvent accéder au système. Cependant, chacun d'eux à un certain nombre de privilèges.

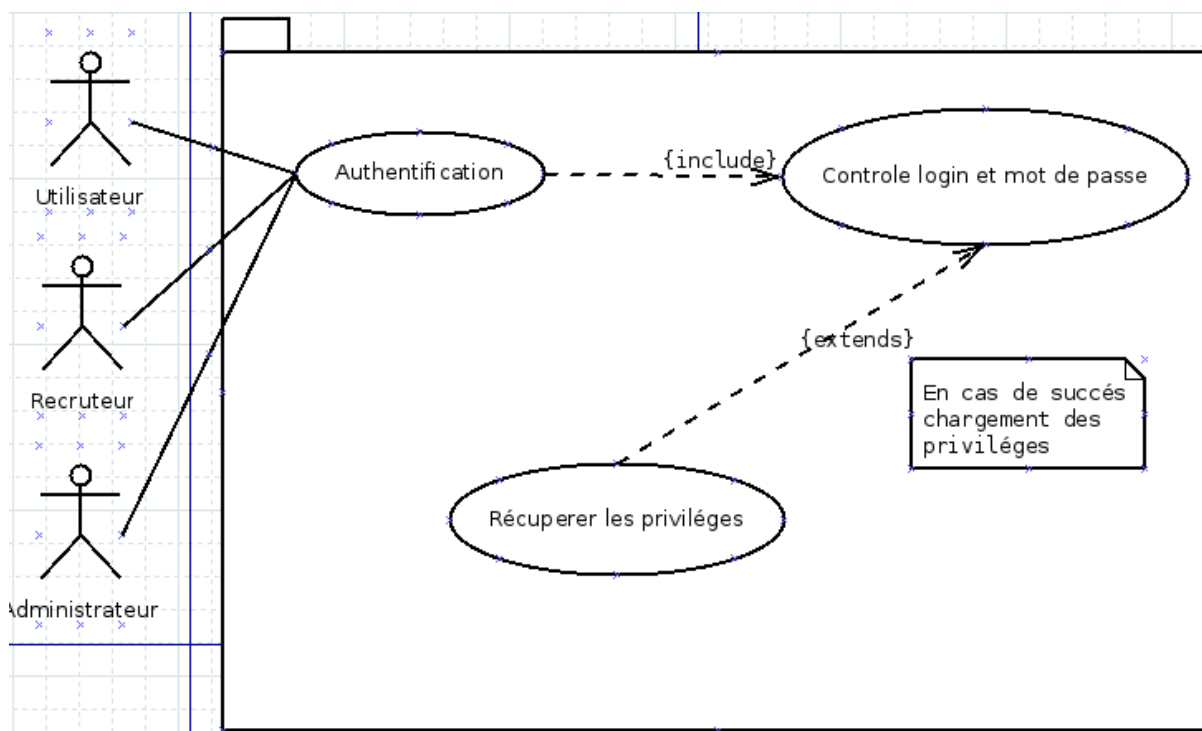


FIGURE 3.7 – Diagramme de cas d'utilisation Authentifier

C'est pour cela, qu'il faut tout au début s'identifier en donnant son login et son mot de passe et les privilèges seront attribués à ces derniers. On a choisi de commencer par traiter ce cas d'utilisation parce que c'est le cas qui initialise tous les autres cas d'utilisation. Une réalisation de ce cas d'utilisation Authentifier se fait comme suit : L'utilisateur ou l'administrateur ou le recruteur saisi son login et mot de passe sur la page Authentification. Après vérification des données, le système sélectionne l'utilisateur en cours. Une requête de recherche portant le profil de l'utilisateur se déclenche dans la base de données afin d'afficher le menu général. En cas d'existence de l'utilisateur, le système charge les privilèges attribué précédemment à l'utilisateur.

Schéma 7 : Diagramme de cas d'utilisation Authentifier

2. Gestion des utilisateurs et les droits d'accès

Nous allons présenter le cas d'utilisation **Gestion des utilisateurs et les droits d'accès** à partir du schéma 8 ci-dessous. Ce schéma 8 ci-dessous présente les acteurs et les cas d'utilisation qui vont être décrits.

Description textuelle

- **Acteurs** : Les utilisateurs de l'application cliente sont les principaux acteurs de ce cas d'utilisation.
- **Description** : Tous les utilisateurs de l'application cliente peuvent accéder aux fonctionnalités utilisateurs. Ce cas d'utilisation requiert impérativement une authentification d'où la flèche **include** sur le cas **Authentification**. Toutes les autres cas d'utilisation sont optionnels d'où la présence de la flèche **extends** sur les autres cas. Et ensuite, les utilisateurs pourront mettre à jour leur profil ou le modifier, consulter la liste des utilisateurs pour y effectuer d'autres tâches.

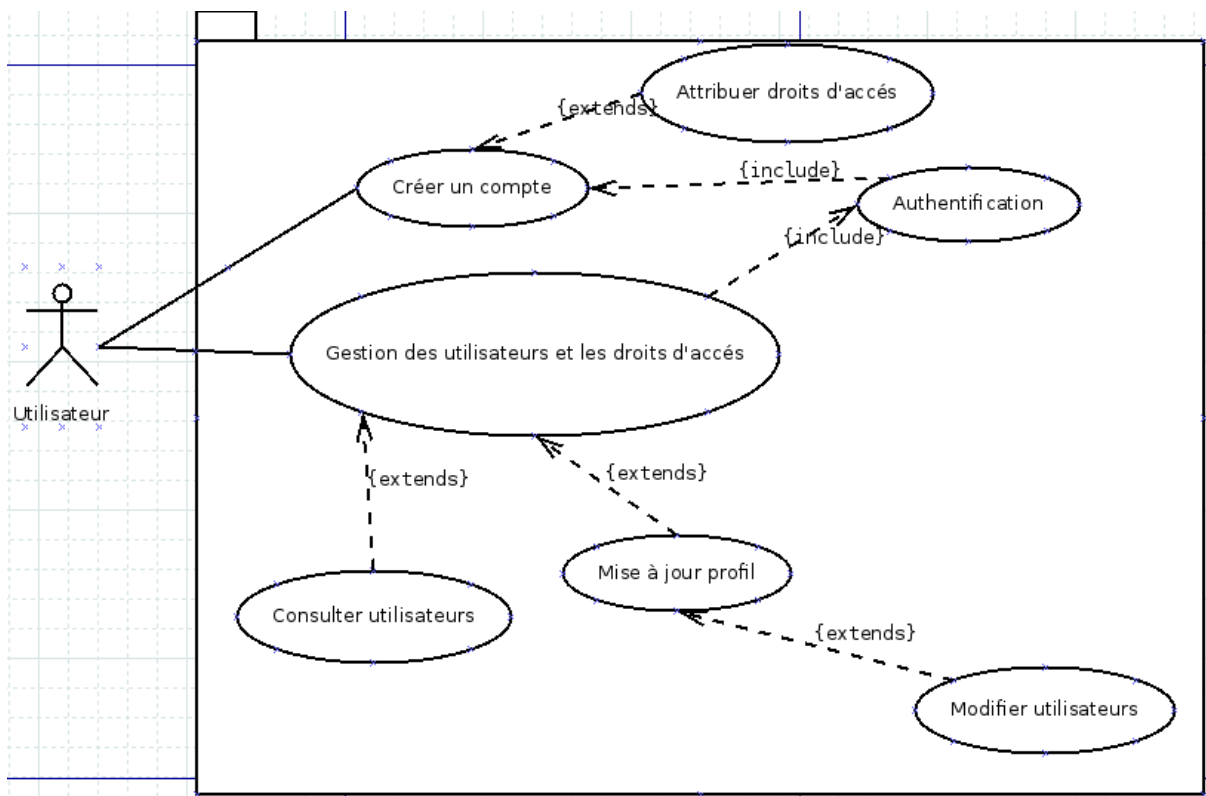


FIGURE 3.8 – Diagramme de cas d'utilisation Gestion des utilisateurs et les droits d'accès

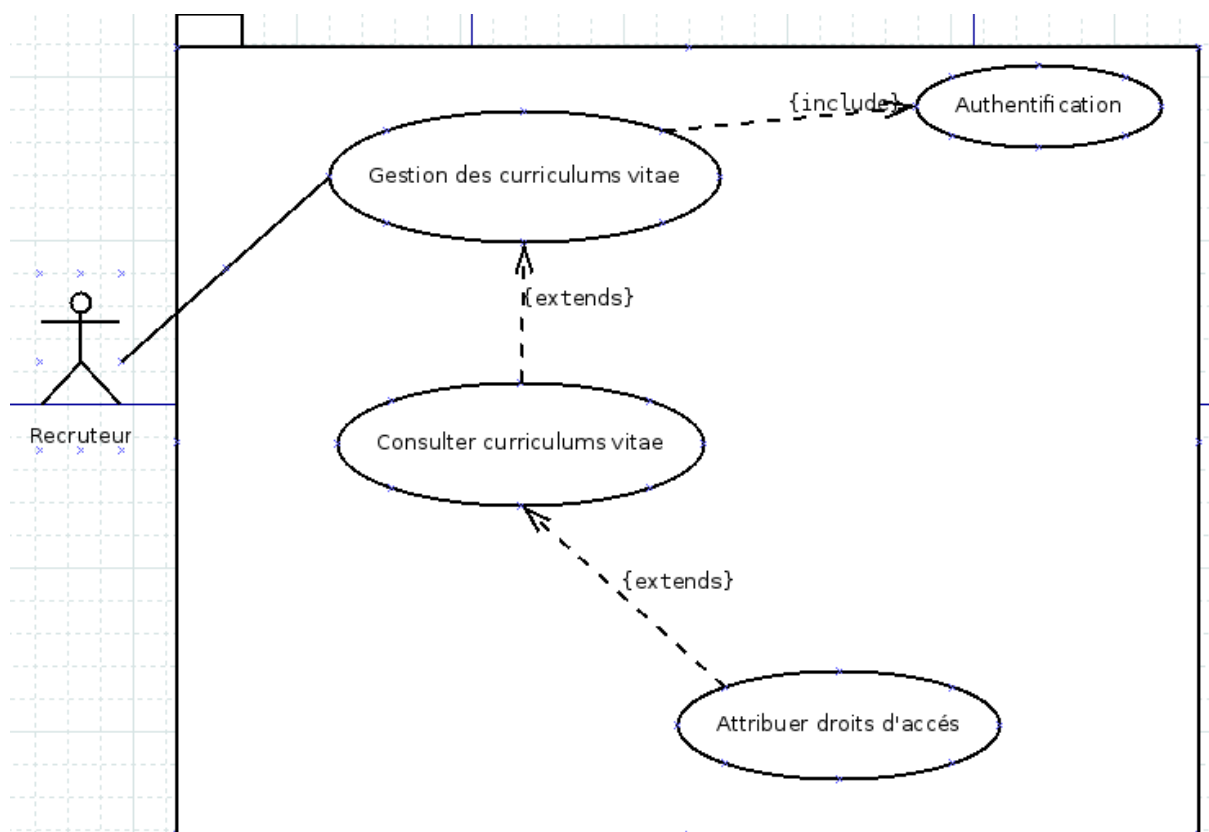


FIGURE 3.9 – Gestion des curriculum vitae

Schéma 8 : Diagramme de cas d'utilisation Gestion des utilisateurs et les droits d'accès**3. Gestion des curriculum vitae**

Nous allons présenter le cas d'utilisation **Gestion des curriculum vitae** à partir du schéma 9 ci-dessous. Ce schéma présente les acteurs et les cas d'utilisation qui vont être décrits.

Description textuelle

- **Acteurs** : Les recruteurs de l'application cliente sont les principaux acteurs de ce cas d'utilisation.
- **Description** : comme pour le cas d'utilisation précédent la flèche **include** indique impérativement qu'on doit s'authentifier avant d'exécuter les autres cas et la flèche **extends** indique que l'exécution des autres cas est optionnelle. De ce fait, après authentification, tout recruteur pourra consulter la banque de curriculum vitae disponible dans le système.

Schéma 9 : Gestion des curriculum vitae**4. Gestion des emplois**

Nous allons présenter le cas d'utilisation **Gestion des emplois** à partir du schéma 10 ci-dessous. Ce schéma présente les acteurs et les cas d'utilisation qui vont être décrits.

Description textuelle

- **Acteurs** : Les recruteurs de l'application cliente sont les principaux acteurs de ce cas d'utilisation.

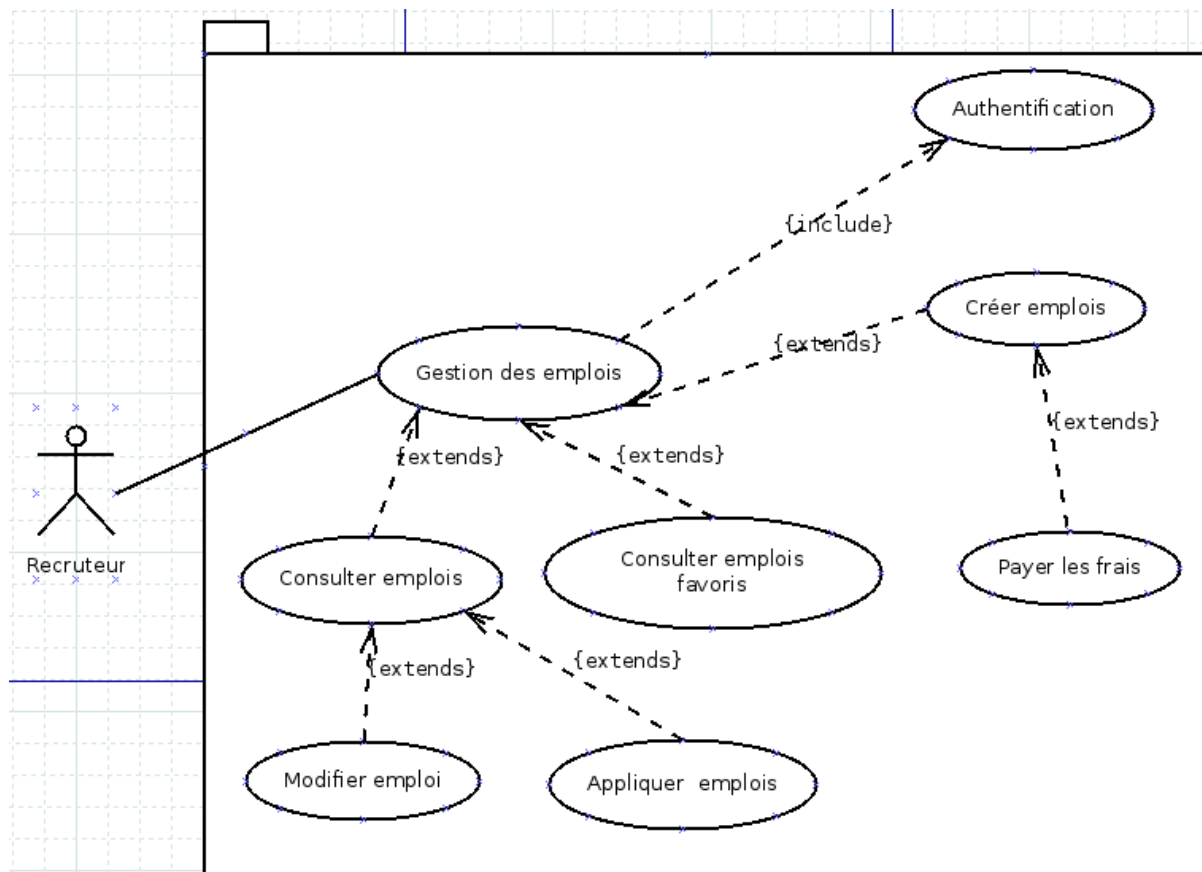


FIGURE 3.10 – Gestion des emplois

- **Description** : Comme dans les cas précédents les flèches **include** et **extends** représentent la même chose. De ce fait, après authentification, tout recruteur peut créer des emplois tout en payant les frais de création de l'emploi, consulter les emplois afin d'y modifier.

Schéma 10 : Gestion des emplois

5. Gestion des offres d'emploi

Nous allons présenter le cas d'utilisation **Gestion des offres d'emploi** à partir du schéma 11 ci-dessous. Ce schéma présente les acteurs et les cas d'utilisation qui vont être décrits.

Description textuelle

- **Acteurs** : Les utilisateurs de l'application cliente sont les principaux acteurs de ce cas d'utilisation.
- **Description** : Comme dans les cas précédents les flèches **include** et **extends** représentent la même chose. De ce fait, après authentification, tout utilisateur peut consulter les offres d'emploi disponibles, d'appliquer aux offres et consulter ses offres favoris.

Schéma 11 : Gestion des offres d'emploi

6. Gestion des entreprises

Nous allons présenter le cas d'utilisation **Gestion des entreprises** à partir du schéma 12 ci-dessous. Ce schéma présente les acteurs et les cas d'utilisation qui vont être décrits.

Description textuelle

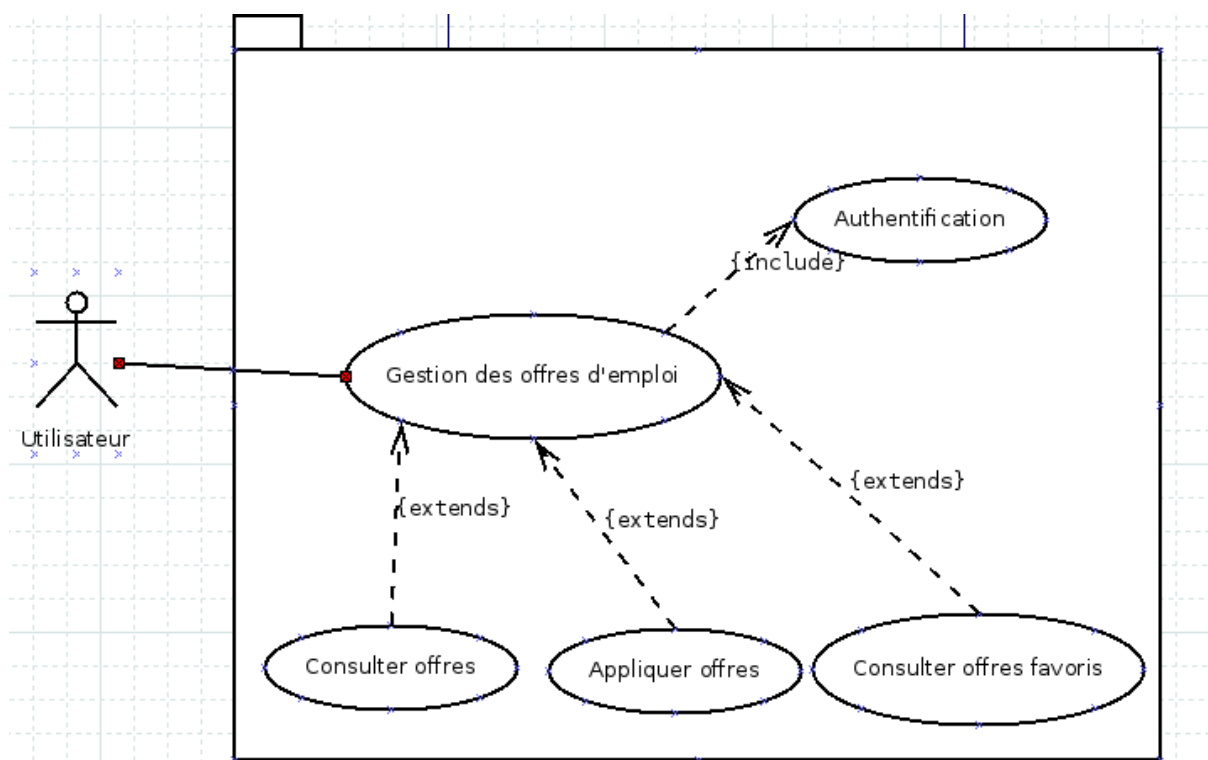


FIGURE 3.11 – Gestion des offres d'emploi

- **Acteurs** : Les recruteurs de l'application cliente sont les principaux acteurs de ce cas d'utilisation.
- **Description** : Comme dans les cas précédents les flèches **include** et **extends** représentent la même chose. De ce fait, après authentification, tout recruteur peut créer une entreprise afin d'enrichir notre annuaire et y effectuer des modifications, consulter l'annuaire des entreprises et aussi ses entreprises favoris.

Schéma 12 : Gestion des entreprises

7. Gestion du réseau social

Nous allons présenter le cas d'utilisation **Gestion du réseau social** à partir du schéma 13 ci-dessous. Ce schéma présente les acteurs et les cas d'utilisation qui vont être décrits.

Description textuelle

- **Acteurs** : Les utilisateurs de l'application cliente sont les principaux acteurs de ce cas d'utilisation.
- **Description** : Comme dans les cas précédents les flèches **include** et **extends** représentent la même chose. De ce fait, après authentification, tout utilisateur peut créer une connexion avec autant d'utilisateurs qu'il souhaite, consulter la liste des amis et effectuer des blocages. Tout utilisateur aussi a le droit de publier des articles ou images ou vidéos, de commenter les publications s'il a le droit, d'apporter des modifications à ses publications ou les supprimer.

Schéma 13 : Gestion du réseau social

8. Gestion du backend

Nous allons présenter le cas d'utilisation **Gestion du backend** à partir du schéma 14 ci-dessous. Ce schéma présente les acteurs et les cas d'utilisation qui vont être décrits.

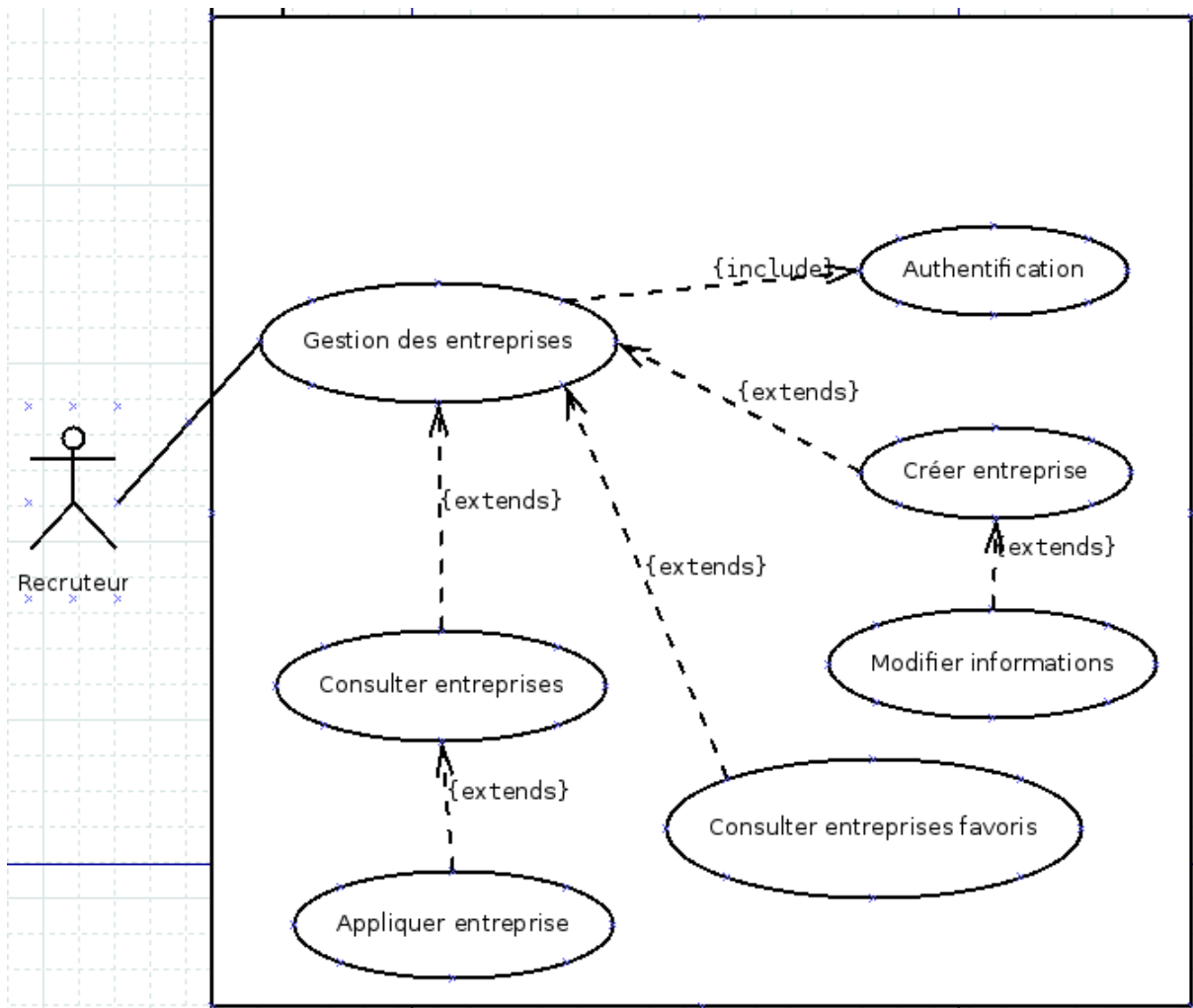


FIGURE 3.12 – Gestion des entreprises

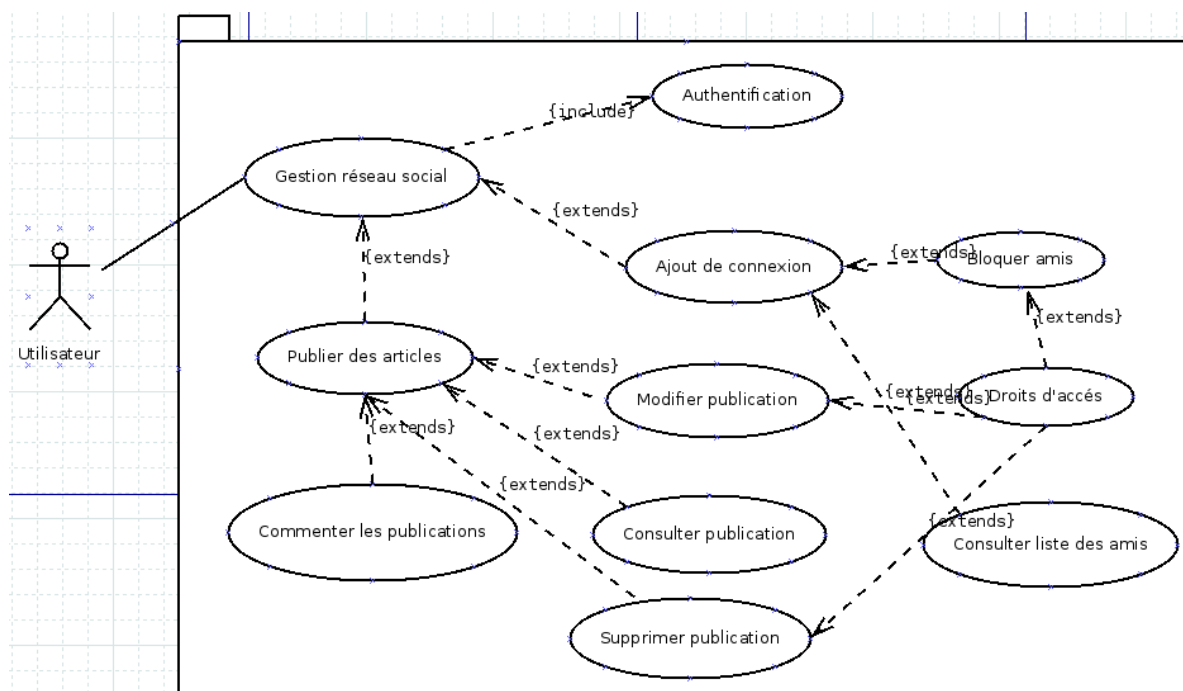


FIGURE 3.13 – Gestion du réseau social

Description textuelle

- **Acteurs** : L'administrateur du backend est le principal acteur de ce cas d'utilisation.
- **Description** : Comme dans les cas précédents les flèches **include** et **extends** représentent la même chose. De ce fait, après authentification, l'administrateur peut ajouter de nouvelles fonctionnalités, consulter les fonctionnalités existantes afin d'y apporter des modifications ou les supprimer.

Schéma 14 : Gestion du backend

3.3 Conception de l'application cliente

3.3.1 Architecture de notre application.

Dans cette partie, nous allons présenter l'architecture que nous proposons. La figure ci-dessous est décomposée en deux parties :

- **backend** : dans cette partie nous avons implémenté la partie web services REST ;
- **frontend** : dans celle-ci nous avons implémenté des fonctionnalités qui permettent d'utiliser les ressources du backend.

Nous avons choisi de passer à un style d'architecture REST afin de corriger les anomalies de l'architecture existante. Ce style d'architecture prend en compte tous les besoins des applications clientes à développer. Elle est le socle de la refonte de l'existant car elle nous permet d'avoir un système évolutif coté backend tout en n'impactant pas la partie frontend.

Le schéma 15 ci-dessous présente en détails les deux parties (**backend** et **frontend**). Notre architecture dispose d'un ensemble de ressources au niveau du backend. Toutes les fonctionnalités de notre application

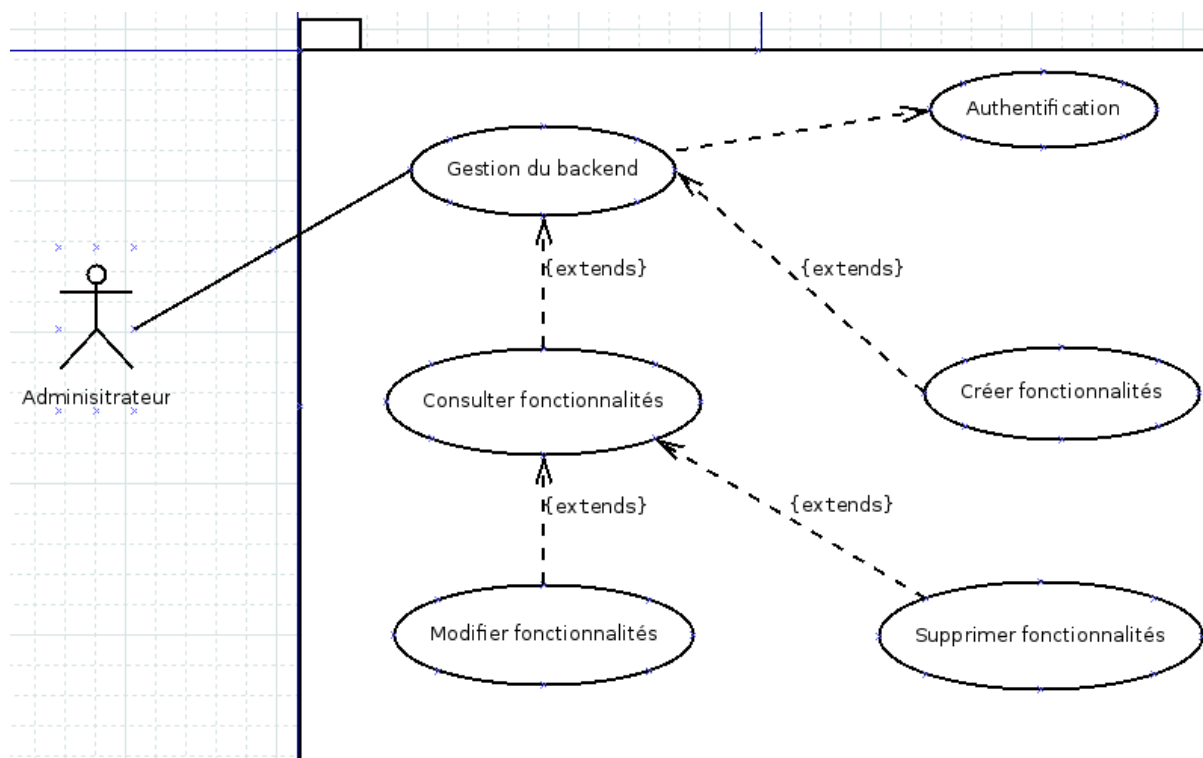


FIGURE 3.14 – Gestion du backend

ainsi que les contraintes utilisateurs seront définies dans cette partie. Les mêmes informations se trouvant de ce côté vont être utilisées par toutes les applications clientes. Ceci nous permettra de garantir l'intégrité de nos données.

En effet, imaginons qu'on est à développer une version de l'application en **IOS**, il suffirait tout simplement de se connecter à une interface du backend pour récupérer toutes les informations nécessaires. Ce qui rendra la tâche facile aux développeurs. Ces ressources sont accessibles via des routes. Par exemple les utilisateurs sont identifiés en tant que ressources par un schéma d'URI du type : **http://127.0.0.1:8000/api/user** où l'ensemble des utilisateurs est référencé. Ce qui est intéressant dans notre architecture est que les utilisateurs finaux n'ont pas un accès direct aux ressources. Ils sont directement branchés sur le frontend où ils effectuent toutes leurs opérations (**GET, POST, PATCH, PUT, DELETE**) et ainsi une communication est établie entre le frontend et le backend. De ce fait, pour des normes de sécurité, nos données ne sont pas exposées à n'importe quel utilisateur. Nous avons néanmoins une application plus ou moins fluide avec une séparation des tâches.

Étant donné que le backend a été élaboré, nous allons présenter le frontend. Il est constitué de deux clients (web et mobile) qui seront utilisés par les utilisateurs finaux afin de faire leurs transactions. Le frontend ne traite aucune requête cliente, il récupère les demandes utilisateurs qu'il envoie au backend et ce dernier est chargé de les traiter puis il retourne la réponse contenant l'en-tête, le code du message, le format d'encodage ainsi que l'information demandée. Imaginons qu'un utilisateur veut faire une recherche sur une entreprise intitulée **wutiko** via le client web comme l'illustre le schéma ci-dessous, la demande est envoyée au backend qui se charge de la transmettre au moteur de recherche. Ainsi le processus inverse est enclenché jusqu'au frontend avec les informations nécessaires.

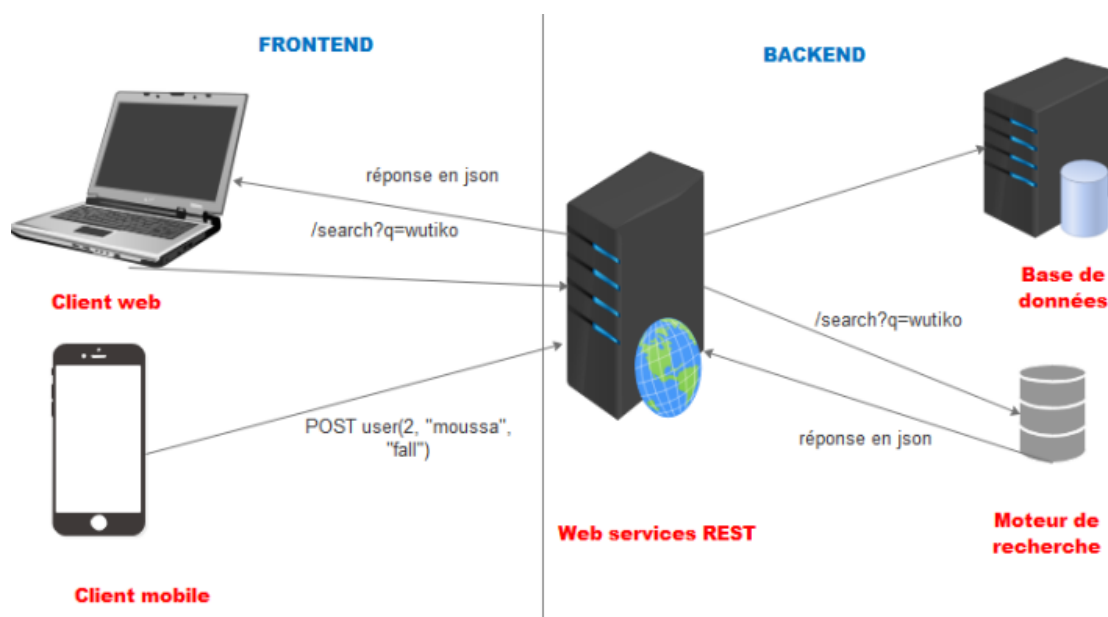


FIGURE 3.15 – Architecture de notre application

Architecture de notre application

Schéma 15 : Architecture de notre application

3.4 Conception détaillée (Diagramme de classes et dictionnaire)

3.4.1 Diagramme de classes.

Le diagramme de classes est considéré comme le plus important de la modélisation orientée objet, il est le seul obligatoire lors d'une telle modélisation. Le diagramme de classes montre la structure interne du système. Il permet de fournir une représentation abstraite des objets du système qui vont interagir ensemble pour réaliser les cas d'utilisation. Il s'agit d'une vue statique car on ne tient pas compte du facteur temporel dans le comportement du système. Les principaux éléments de cette vue statique sont les classes et leurs relations : association, généralisation et plusieurs types de dépendances, telles que la réalisation et l'utilisation. Une classe-association possède les caractéristiques des associations et des classes : elle se connecte à deux ou plusieurs classes et possède également des attributs et des opérations. Une classe-association est caractérisée par un trait discontinu entre la classe et l'association.

Une classe est une description d'un groupe d'objets partageant un ensemble commun de propriétés (les attributs), de comportements (les opérations ou méthodes) et de relations avec d'autres objets (les associations et les agrégations). Le diagramme de classes ci-dessous illustré par le schéma 16 décrit les principales classes de notre système. Chaque classe est composée du nom de la classe et de ses attributs. A partir de ce diagramme, on dégage les entités de la base de données ainsi que les différents relations qui existent entre eux correspondant dans l'application à développer.

Schéma 16 : Diagramme de classes

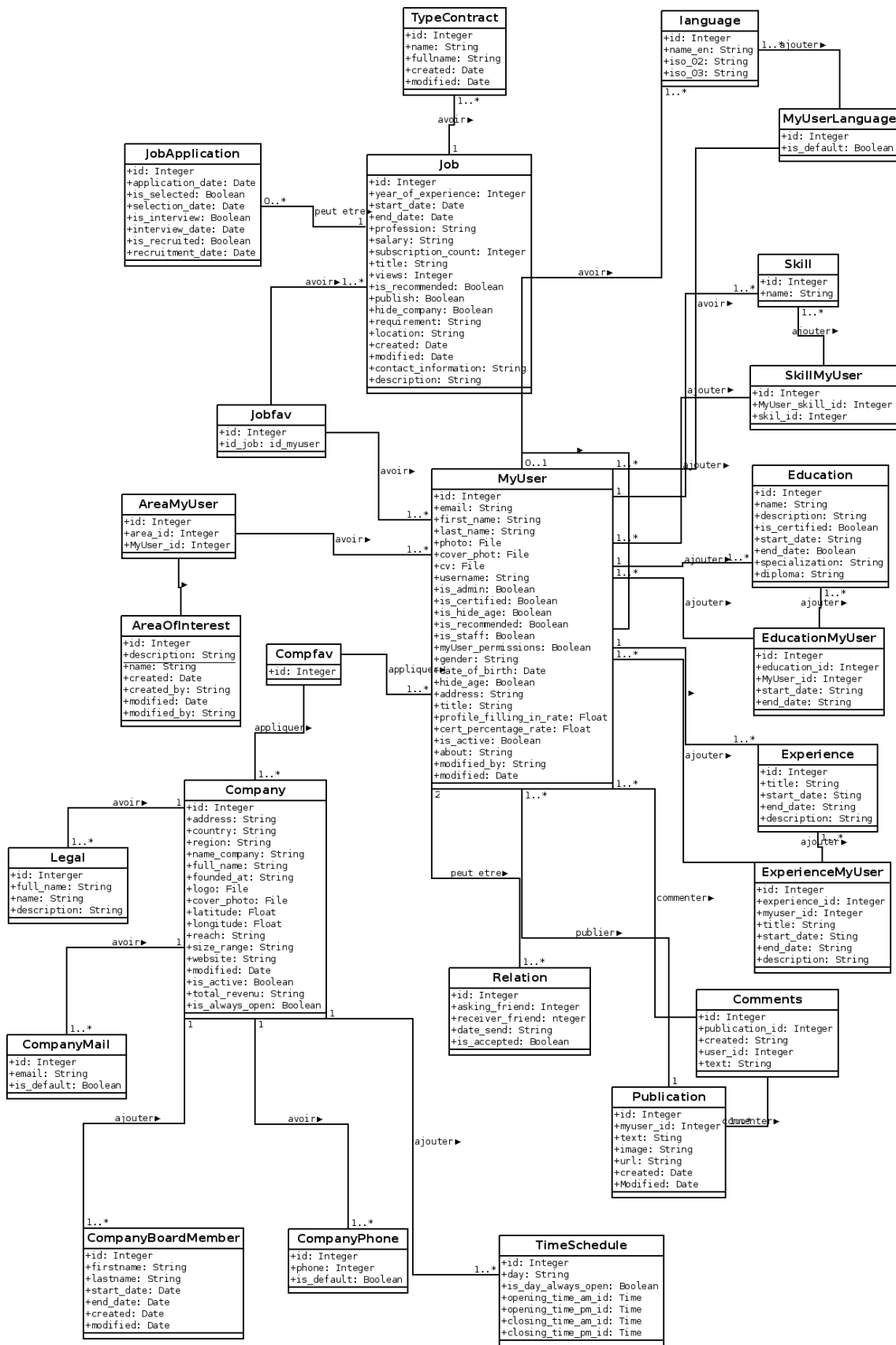


FIGURE 3.16 – Diagramme de classes

3.4.2 Dictionnaire du diagramme de classes.

Le diagramme de classes ci-dessus présente l'ensemble des classes de notre système ainsi que les relations qui existent entre eux. Nous allons décrire chacune des classes de ce diagramme.

Classes	Description	Relation
jobfav	Cette classe décrit les emplois favoris des utilisateurs.	C'est une classe d'association entre job et MyUser.
TypeContract	Cette classe décrit le type de contrat(CDD,CDI, stage...) lié à un emploi.	Elle a une relation de un à plusieurs avec la classe job.
jobApplication	Cette classe décrit la liste des emplois appliqués.	Elle a une relation de un à plusieurs avec la classe job.
language	Cette classe permet aux utilisateurs d'ajouter ses langues afin de mettre à jour leur cv.	Elle a une relation de un à plusieurs avec la classe MyUser.
Skill	Cette classe permet aux utilisateurs d'ajouter ses compétences afin de mettre à jour leur cv.	Elle a une relation de un à plusieurs avec la classe MyUser.
MyUserLanguage	Cette classe décrit la liste des langues choisies par l'utilisateur.	C'est une classe d'association entre les classes Language et MyUser.
SkillMyUser	Cette classe décrit la liste des compétences choisies par l'utilisateur.	C'est une classe d'association entre les classes Skill et MyUser.
Experience	Cette classe permet aux utilisateurs d'ajouter ses expériences professionnelles afin de mettre à jour leur cv.	Elle a une relation de un à plusieurs avec la classe MyUser.
ExperienceMyUser	Cette classe décrit la liste des utilisateurs ainsi que les expériences professionnelles effectués.	C'est une classe d'association entre les classes Experience et MyUser.
Education	Cette classe permet aux utilisateurs d'ajouter les établissements fréquentés afin de mettre à jour leur cv.	Elle a une relation de un à plusieurs avec la classe MyUser.
EducationMyUser	Cette classe décrit la liste des utilisateurs ainsi que les établissements fréquentés.	C'est une classe d'association entre les classes Education et MyUser.
job	Cette classe permet aux utilisateurs d'ajouter des emplois.	Elle a une relation de un à plusieurs avec les classes TypeContract, jobApplication, et company. Elle a une relation d'association aussi avec la classe MyUser.

Classes	Description	Relation
AreaOfInterest	Cette classe permet d'ajouter des centres d'intérêt pour les utilisateurs et les compagnies.	Elle a une relation de un à plusieurs avec MyUser et Company.
AreaMyUser	Cette classe permet de lister les centres d'intérêt des utilisateurs .	C'est une classe d'association entre les classes AreaOfInterest et MyUser.
MyUser	Cette classe permet aux utilisateurs d'ajouter leurs filiations(nom, prénom, adresse...).	Elle a une relation de un à plusieurs avec language, skill, education, experience et AreaOfInterest. Et aussi une relation de classe d'association avec job, education, skill et language.
Legal	Cette classe décrit le nom légal de l'entreprise.	Elle a une relation de un à plusieurs avec la classe company.
Company	Cette classe décrit les filiations des compagnies(nom, adresse, pays, région..).	Elle a une relation de un à plusieurs avec job, legal, place, compfav, CompanyBoardMember, CompanyPhone, TimeSchedule, CompanyMail.
Place	Cette classe décrit les autres places des compagnies.	Elle a une relation de un à plusieurs avec company.
Compfav	Cette classe décrit les compagnies favoris des utilisateurs.	C'est une classe d'association entre company et MyUser.
CompanyMail	Cette classe décrit les e-mails envoyés à une compagnie.	Elle a une relation de un à plusieurs avec company.
CompanyBoardMember	Cette classe décrit les utilisateurs capable d'administrer une compagnie comme par exemple : avoir le droit d'ajouter un emploi.	Elle a une relation de un à plusieurs avec company.
CompanyPhone	Cette classe décrit les numéros de téléphone de chaque compagnie.	Elle a une relation de un à plusieurs avec company.
TimeSchedule	Cette classe décrit les horaires d'ouverture et de fermeture de chaque compagnie.	Elle a une relation de un à plusieurs avec company.
Relation	Cette classe décrit les relations d'amitiés entre les utilisateurs.	Elle a une relation de deux à plusieurs avec MyUser.
Publication	Cette classe décrit les publications faites par les utilisateurs	Elle a une relation de un à plusieurs avec MyUser.
Comments	Cette classe décrit les e-mails envoyés à une compagnie.	C'est une classe d'association entre Publication et MyUser.

Schéma 17 : Dictionnaire du diagramme de classes

Chapitre 4 : Implémentation et Présentation de l'application

4.1 Implémentation du backend

1. Scénario des utilisateurs

Nous allons implémenter un service Web REST dans notre backend. Celui-ci permet de rechercher des emplois en ligne. Deux types d'utilisateurs ont été mis en place : l'un des utilisateurs est le recruteur. Il est chargé de créer des compagnies, des emplois et consulter l'ensemble des profils des utilisateurs. l'autre est chargé d'utiliser les ressources de la plate-forme (Web). Les utilisateurs qui utilisent les ressources de la plate-forme vont créer des comptes. Une fois leur compte actif ils peuvent se loguer pour mettre à jour leur profil. Ils peuvent consulter l'ensemble des compagnies enregistrées dans notre base de données. Et aussi ils leurs seraient proposés l'ensemble des emplois disponibles. Chaque utilisateur peut postuler à autant d'emplois qu'il désire. De ce fait, ils auront accès à l'ensemble des emplois qu'ils ont appliqué afin de faire le suivi. Les recruteurs peuvent créer une entreprise et y ajouter des administrateurs. Chacun de ces administrateurs peut ajouter des emplois pour le compte de l'entreprise. Ils peuvent aussi avoir le profil recruteur pour créer des emplois. Toutes ces activités auront des interfaces pour être accessible. Mais en guise d'illustration pour notre mémoire, nous allons utiliser le modèle **Utilisateur** pour créer un compte (voir toutes les autres fonctionnalités au niveau de la présentation de l'application).

2. Choix du langage de programmation

Il existe des dizaines de langages de programmation qui sont capables de mettre en place un service Web REST. Mais notre choix est porté sur **python**.

(a) Pourquoi python ?

Python est le langage le plus adapté pour notre projet de par sa simplicité de mise en œuvre. Sa portabilité est un facteur clé pour le déploiement de notre projet car disponible sous toutes les plate-formes (Unix et Windows). Ce fut notre choix du fait aussi de son aspect orienté objet avec la définition de classe, héritage multiple, introspection (consultation du type, des méthodes proposées), ajout/retrait dynamique de classes, de méthode, compilation dynamique de code, délégation ("duck typing"), passivation/activation, surcharge d'opérateurs, etc.

Il dispose de nombreux frameworks comme **Django** (pour le développement web), **Django REST Framework** (boîte à outils de création d'architecture REST avec Django)... Dans notre cas, comme nous voulons mettre en place une application utilisable par plusieurs terminaux (web et mobile), nous allons travailler avec Django REST Framework du fait qu'il hérite de toutes les caractéristiques de python.

(b) Qu'est ce que Django REST framework ?

Django REST framework est une boîte à outils de création d'architecture REST avec Django. Il est utilisé pour développer des applications Restfull (respectant l'architecture REST). il permet de créer facilement des ressources disponible sous plusieurs formats (XML, JSON...).

3. Principes d'implémentation avec Django REST Framework

Django REST Framework présente un développement modulaire. Le schéma 18 ci-dessous décrit l'architecture générale de Django REST Framework. Il est constitué de cinq parties dont chacune

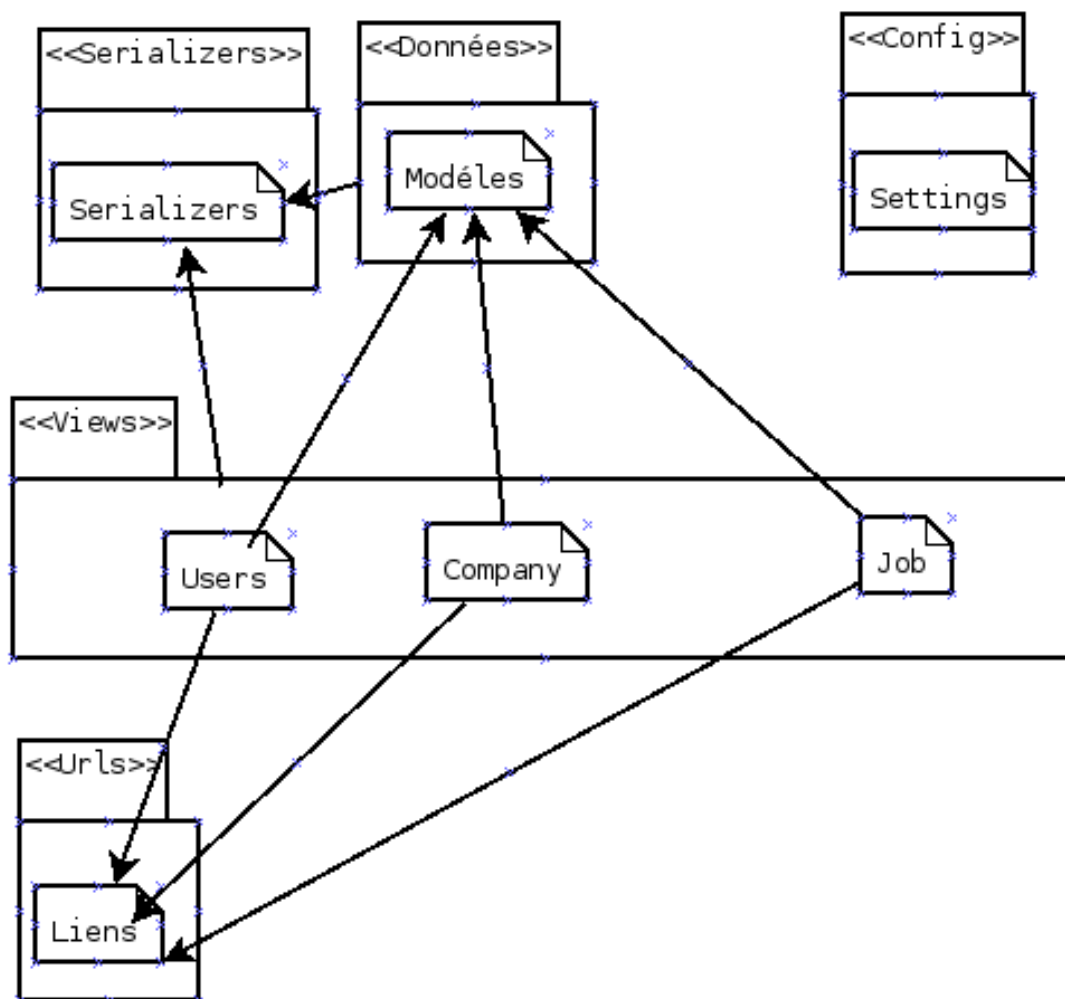


FIGURE 4.17 – Différents processus pour l'implémentation

d'elles effectue une tâche bien précise. Ainsi, nous allons énumérer chacune des parties et les liens qui existent entre eux.

Schéma 18 : Différents processus pour l'implémentation

(a) **Modèles**

Le modèle est une source d'information unique et définitive à propos de nos données. Il contient les champs et le comportement essentiels des données que nous stockons. Généralement, chaque modèle correspond à une seule table de la base de données.

Les bases d'un modèle sont :

- Chaque modèle est une classe Python qui hérite de **django.db.models.Model**.
- Chaque attribut du modèle représente un champ de base de données.
- Avec tout ça, Django REST Framework nous permet de générer automatiquement la base de données.

Imaginons qu'on est à créer des utilisateurs pour l'application **wutiko**. Nous avons besoin de mettre en place le modèle des utilisateurs. Ce modèle va hériter de toutes les fonctions et méthodes de **django.contrib.auth.models** pour la création et l'authentification des utilisateurs. Le modèle MyUser ci-dessous représente la table des utilisateurs dans notre base de

données utilisateurs. Il est composé de trois parties : la première partie représente le nom de la classe, la seconde représente les attributs avec le type de chacun d'eux et enfin la dernière partie est destinée pour la définition des fonctions liées à cette classe.

```
65 class MyUser(abstractBaseUser):
66     email = models.EmailField(unique=True)
67     first_name = models.CharField(max_length=100, blank=True)
68     last_name = models.CharField(max_length=100, blank=True)
69     photo = models.FileField(upload_to=get_upload_file_name, default="upload_files/default.png")
70     cover_photo = models.FileField(upload_to=get_upload_file_name, default="upload_files/default.png")
71     cv = models.FileField(upload_to=get_upload_file_name, blank=True)
72     #password = models.CharField(max_length=100)
73     username = models.CharField(max_length=100, unique=True)
74     is_admin = models.BooleanField(default=False)
75     is_certified = models.BooleanField(default=False)
76     is_hide_age = models.BooleanField(default=False)
77     is_recommended = models.BooleanField(default=False)
78     is_staff = models.BooleanField(default=False)
79     MyUser_permissions = models.BooleanField(default=False)
80     gender = models.CharField(max_length=100, blank=True)
81     date_of_birth = models.DateField(auto_now_add=True)
82     hide_age = models.BooleanField(default=False)
83     address = models.CharField(max_length=100, blank=True)
84     title = models.CharField(max_length=100, blank=True)
85     profile_filling_in_rate = models.FloatField(default=0)
86     cert_percentage_rate = models.FloatField(default=0)
87     is_active = models.BooleanField(default=False)
88     about = models.CharField(max_length=100, blank=True)
89     modified_by = models.CharField(max_length=100, blank=True)
90     modified = models.DateField(auto_now=True)
91     objects = MyUserManager()
92     USERNAME_FIELD = 'email'
93     REQUIRED_FIELDS = ['username']
94     def __unicode__(self):
95         return self.email
96
97     def get_full_name(self):
98         return ' '.join([self.first_name, self.last_name])
99
100     def get_short_name(self):
101         return self.first_name
102
```

Exemple : Modèle des utilisateurs

(b) **Serializers**

Les serializers vont nous permettre de sérialiser les instances en JSON et transformer le JSON en instance python. Si nous suivons la logique de création d'utilisateurs comme illustré dans la partie modèle, les données liées aux utilisateurs vont être sérialisés via le processus de création ou de récupération de données. La classe MyUserSerializer ci-dessous représente la classe qui permet de sérialiser les objets de la classe MyUser défini dans le modèle précédent. La sérialisation présente trois parties : la première est le nom du serializer, la seconde est de définir le modèle et ses attributs à sérialiser et enfin de définir les fonctions liées au sérialiser.

```
class MyUserSerializer(RelationModelSerializer):
    password = serializers.CharField( required=True)
    confirm_password = serializers.CharField( required=False)
    class Meta:
        model = MyUser
        fields = '__all__'

    def create(self, validated_data):
        return MyUser.objects.create(**validated_data)

    def update(self, instance, validated_data):
        instance.username = validated_data.get('username', instance.username)
        #instance.tagline = validated_data.get('tagline', instance.tagline)

        instance.save()

        password = validated_data.get('password', None)
        confirm_password = validated_data.get('confirm_password', None)

        if password and confirm_password and password == confirm_password:
            instance.set_password(password)
            instance.save()

        update_session_auth_hash(self.context.get('request'), instance)

        return instance
```

Exemple : Serializers des utilisateurs

(c) Views

Une fonction de vue, ou vue, est une simple fonction Python acceptant une requête Web et renvoyant une réponse Web. Cette réponse peut contenir le contenu HTML d'une page Web, une redirection, une erreur 404, un document XML ou JSON, une image ... ou vraiment n'importe quoi d'autre. La vue elle-même contient la logique nécessaire pour renvoyer une réponse. Nous allons adopter la même démarche que précédemment pour la création d'utilisateurs. La vue MyUserView ci-dessous récupère les requêtes clientes, se charge de voir l'intégrité de toutes les données à ajouter dans la base de données. Ainsi, il fait appel à la fonction serializer de cette vue concernée pour retourner la réponse aux clients. La réponse sera le statut, le message, le format d'encodage... Comme toutes les autres parties énumérées précédemment, la vue présente trois parties : la première est de donner un nom à chaque vue, la seconde est de traiter la requête utilisateur en se basant sur le modèle et le serializer, et enfin de définir les fonctions qui vont se charger de récupérer les requêtes clientes et de retourner les réponses.

```
class MyAPIView(viewsets.ModelViewSet):
    lookup_field = 'username'
    queryset = MyUser.objects.all()
    serializer_class = MyUserSerializer
    def get_permissions(self):
        if self.request.method in permissions.SAFE_METHODS:
            return (permissions.AllowAny(),)

        if self.request.method == 'POST':
            return (permissions.AllowAny(),)

        return (permissions.AllowAny(),)

    def create(self, request):
        serializer = self.serializer_class(data=request.data)

        if serializer.is_valid():
            MyUser.objects.create_user(**serializer.validated_data)

            return Response(serializer.validated_data, status=status.HTTP_201_CREATED)

        return Response({
            'status': 'Bad request',
            'message': 'Account could not be created with received data.'
        }, status=status.HTTP_400_BAD_REQUEST)
```

Exemple : Vue des utilisateurs

(d) Urls

Pour les urls nous allons mettre en place les **routers** qui hérite de **REST framework**. Ils vont nous permettre d'ajouter des urls aisément sans avoir à les taper les unes aux autres. Nous aurons seulement besoin de spécifier la vue et le nom qu'on lui attribue et Django REST framework va s'occuper de créer les urls automatiquement. Nous allons avoir deux urls créés automatiquement pour la vue des utilisateurs :

- 'localhost :8000/api/user/' : pour la création ou la récupération des données utilisateurs.
- 'localhost :8000/api/user/pk' : pour la modification ou la suppression de données d'un utilisateur.

```
router = routers.SimpleRouter()
router.register(r'user', views.MyAPIView)
```

Exemple : Urls des utilisateurs

(e) Settings

C'est un fichier de réglage qui contient toute la configuration de Django REST framework.

4.2 Implémentation du frontend

1. Scénario des utilisateurs

Dans la partie cliente, nous allons essayer d'utiliser toutes les ressources disponibles sur le serveur backend. L'intérêt est de montrer comment une application cliente peut utiliser les ressources d'un service Web REST. Ainsi comme nous l'avons fait coté **backend**, nous allons travailler avec le modèle **MyUser** avec les deux types d'utilisateurs présentés précédemment dans l'implémentation du backend. Pour voir toutes les autres tâches ; veuillez consulter l'application.

2. Choix technologique

Il existe de nombreux framework capable de communiquer à un interface REST. Mais notre choix est porté sur Angularjs.

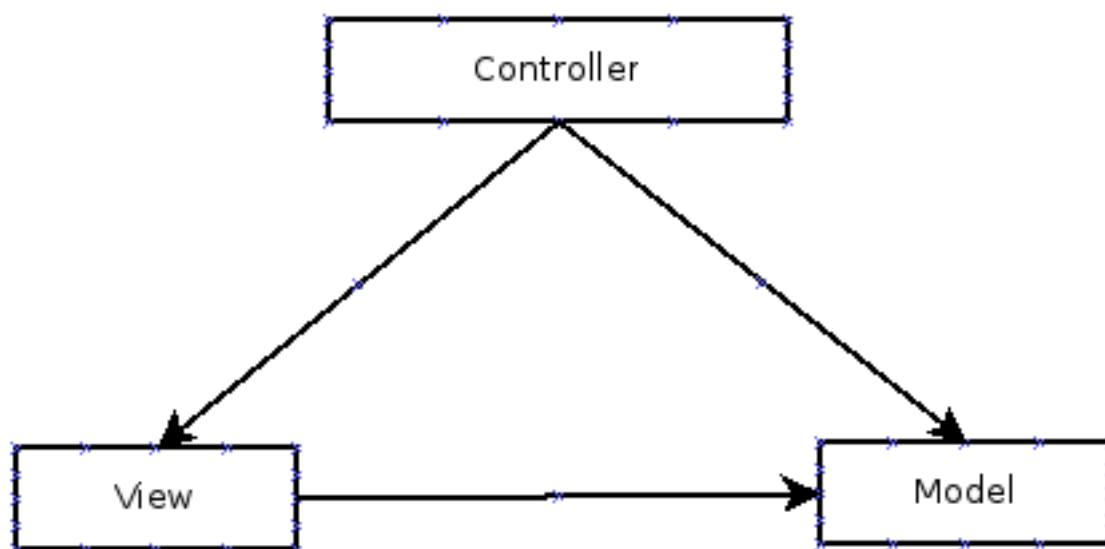


FIGURE 4.18 – Modèle MVC

3. Qu'est ce qu'AngularJS ?

AngularJS est un framework de développement Javascript côté navigateur comme JQuery. Il est utilisé notamment par Gmail et Google +. Il permet de faire des interfaces plus réactives et one page, c'est-à-dire sans rechargement de la page.

4. Pourquoi avons nous choisi d'utiliser AngularJS ?

Nous avons choisi AngularJS car il était la meilleure solution technique pour le projet que nous développons actuellement. Il présente un vrai gain de productivité pour le développement d'applications Web ergonomiques. Le point fort le plus marquant est la façon d'amélioration de la réactivité de l'interface utilisateur. Vu de l'internaute, le site ou logiciel va paraître plus fluide et plus ergonomique. En ce qui concerne les développeurs, il nous impose une modification de l'architecture de nos applications, à laquelle nous devons nous adapter.

5. Principe d'implémentation avec Angularjs

Dans AngularJS le modèle **MVC** (Model View Controller) est implémenté dans JavaScript et HTML. La vue est définie dans HTML, tandis que le modèle et le contrôleur sont mises en œuvre en JavaScript. Comme l'illustre le schéma 19 ci-dessous, les différents composants ont une relation définie par leurs interactions. Le modèle est seulement conscient de lui-même. En général, la vue est au courant du modèle, car il est responsable du rendu des données contenues dans le modèle et en invoquant des actions (méthodes) sur le modèle. Le travail du contrôleur est de créer et remplir le modèle et le remettre à la vue. Le contrôleur a besoin de savoir sur le modèle et comment résoudre la vue et fournir le modèle.

Schéma 19 : Modèle MVC

Il y a plusieurs façons que ces composants peuvent être mis ensemble dans AngularJS mais la forme la plus simple commence par la vue.

(a) Vues

La vue dans une application AngularJS est créée avec HTML. Il y a des étapes à suivre pour utiliser AngularJS dans une vue :

- Référencé le cadre de AngularJS :

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.5/angular.min.js"/>
```

- Définir l'application AngularJS :

```
<div ng-app = "wutiko">
```

Lorsque ce script charge, il examinera la page et de chercher des directives comme par exemple **ng-app** qui permet d'amorcer l'application. La vue n'est pas seulement constitué de balises, elles a besoin d'un contrôleur et d'un modèle pour faire plus que juste quelques balises HTML. Le contrôleur sera responsable de la connexion entre la vue et le modèle. En guise d'exemple nous vous présentons la vue ci-dessous qui permet de lister les utilisateurs. En effet, au niveau de la ligne cent soixante quatre, nous avons déclaré notre contrôleur dans un div avec son domaine d'application. A la ligne cent soixante cinq, on applique une boucle **for** sur la liste d'utilisateur afin d'afficher les utilisateurs.

```

164 <div class="col-lg-12 col-md-12 col-sm-12 col-xs-12" ng-controller="UserController as uc">
165 <ul class="list-group list-of-members" ng-repeat="d in uc.user | filter:search">
166 <li class="list-group-item peoples">
167 <div class="col-lg-2 col-md-2 col-sm-2 col-xs-2 picture"></div>
168 <div class="col-lg-8 col-md-8 col-sm-8 col-xs-8">
169 <div class="people-name"><a href="#">{{ d.first_name }} {{d.last_name}} </a></div>
170 <div class="fonction"><span>{{d.title}} </span><span>at </span><span><a href="#"> </a></div>
171 <div class="education-list-of-members"><span> </span><span></span><span></span></div>
172 </div>
173 <div class="col-lg-2 col-md-2 col-sm-2 col-xs-2 people-address">
174 <div class="validation"><small> {{d.address}} </small> </small></div>
175 <div class="address">
176 <button class="btn btn-default" type="button">View this profile</button>
177 </div>
178 </li>
179 </ul>
180 </div>
181

```

(b) Contrôleurs

Le contrôleur contient un ou plusieurs modèles de données. Il contient des méthodes propres à la vue ainsi que des variables d'état ou d'affichage. Il sert également de proxy au modèle. En AngularJS, le contrôleur est résolu par un nom et par une directive appelé **ng-controller** illustré comme dans l'exemple de la vue ci-dessus. En guise d'exemple, nous allons présenter le contrôleur **UserController** ci-dessous qui permet de lister les utilisateurs. A la ligne six cent douze, nous avons déclaré notre contrôleur avec toutes ses dépendances. Et à la ligne six cent quinze, nous avons effectué une opération **Get** au niveau du **backend** pour récupérer l'ensemble des utilisateurs.

```

612 function UserController($scope, server, $http, Authentication, server) {
613   var vm = this;
614
615   $http.get(server.url + '/api/user/').
616   success(function(data) {
617     vm.user = data;
618   });
619
620 }

```

(c) Modèles

Les modèles sont dédiés à un type de données. Ils contiennent les données à afficher ainsi que les données qui doivent être recueillies dans des champs d'entrées et toutes les méthodes de manipulation de cette dernière :

- Appels de services
- Ajout
- Sélection

Les modèles font appels aux services en cas de besoin de communication serveur. L'objet **vm** injecté dans le contrôleur ci-dessus peut être utilisé comme modèle directement.

4.3 Présentation de l'application

4.3.1 Création de compte.

Tout d'abord l'utilisateur doit créer un compte dans le système. Ainsi le schéma 20 ci-dessous décrit les champs à remplir pour la création de compte. Pour ne pas trop stresser l'utilisateur, nous avons voulu lui proposer quelques champs ainsi après connexion il pourra mettre à jour les autres informations.

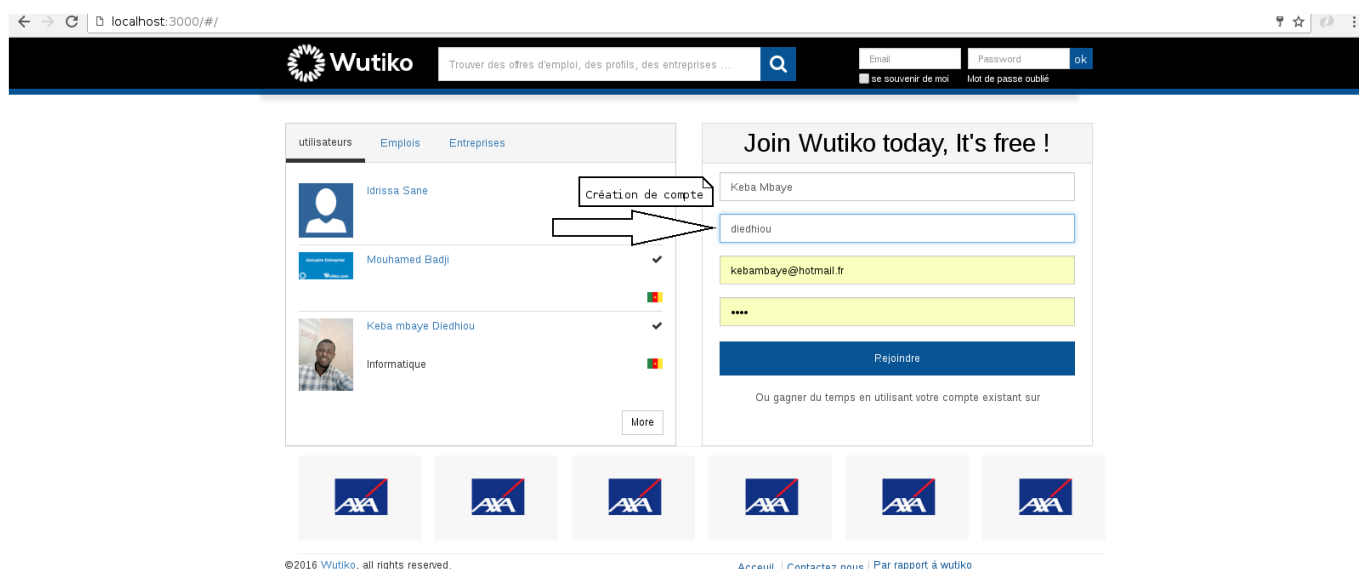


Schéma 20 : Création de compte

4.3.2 Authentification.

Après la création de compte, l'utilisateur pourra s'authentifier afin d'accéder au plate-forme. Le schéma 21 ci-dessous décrit les paramètres à utiliser pour pouvoir s'authentifier. Nous avons l'email au lieu du username car nous jugeons facile à l'utilisateur de capter son mail au lieu de son username mais aussi pour des règles de sécurité.

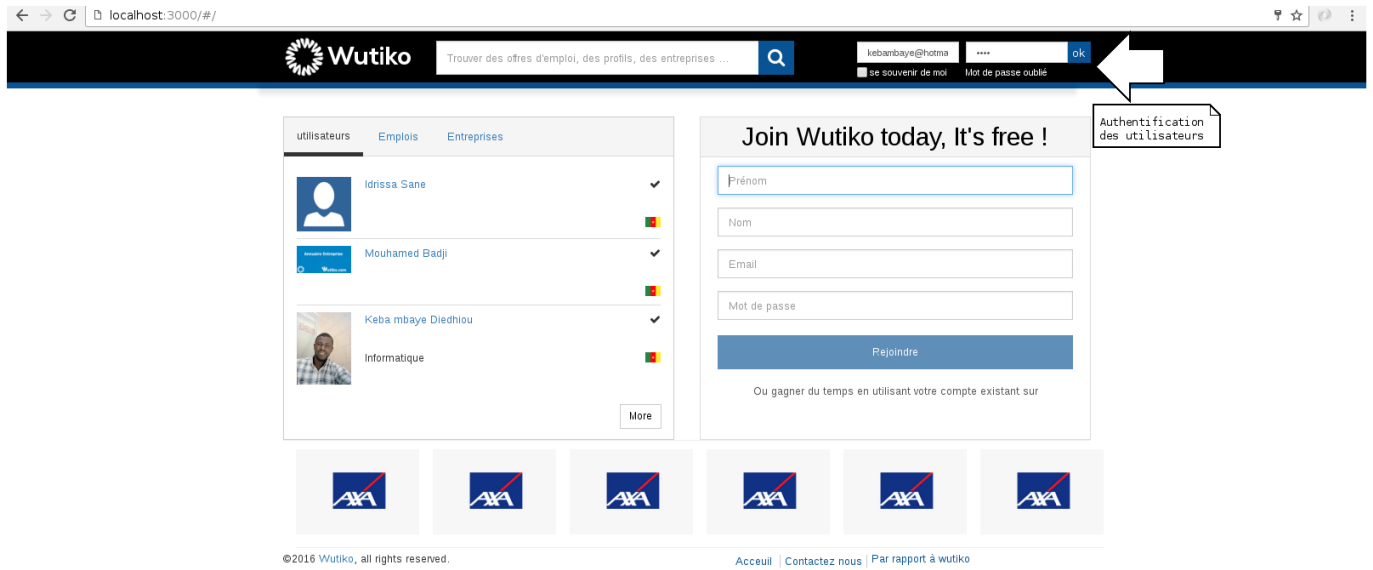
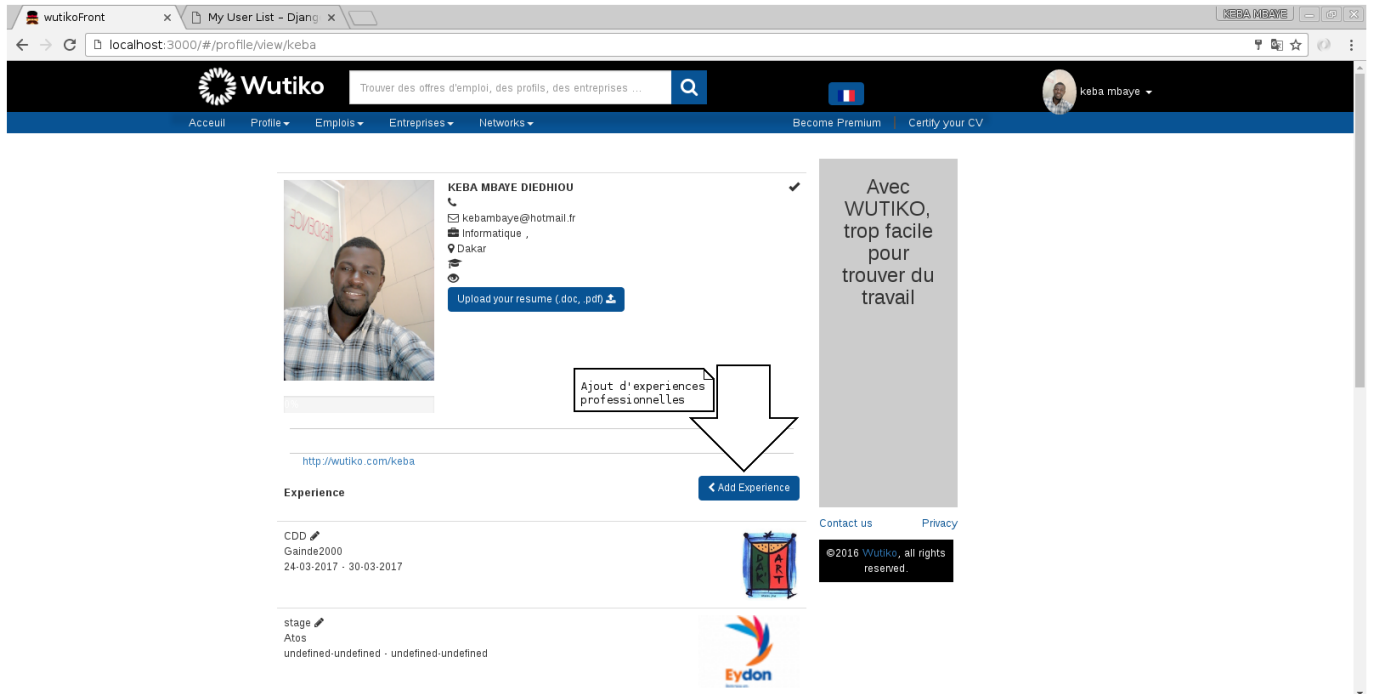


Schéma 21 : Authentification

4.3.3 Création du cv en ligne.

Comme le décrit le schéma 22 ci dessous, l'utilisateur a la possibilité de créer un curriculum vitae en ligne. Le curriculum vitae est composé de quatre parties qui sont l'expérience professionnelle, l'éducation(cursus scolaire de l'utilisateur), les langues parlées et écrites et ses compétences.



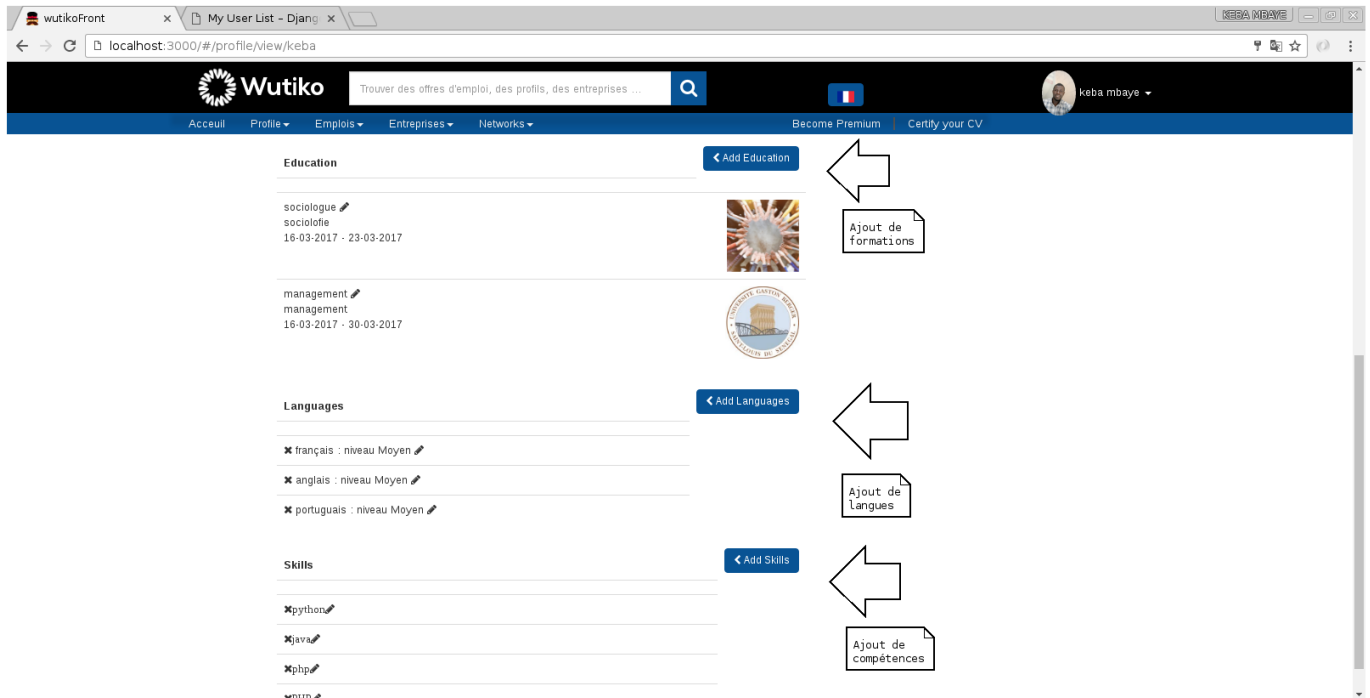


Schéma 22 : Création du cv en ligne

4.3.4 Mise à jour profil.

Dans cette partie, l'utilisateur a la possibilité de mettre à jour son profil à savoir de compléter les champs manquants lors de la création de compte. Ainsi le schéma 23 ci-dessous décrit les éléments à ajouter.

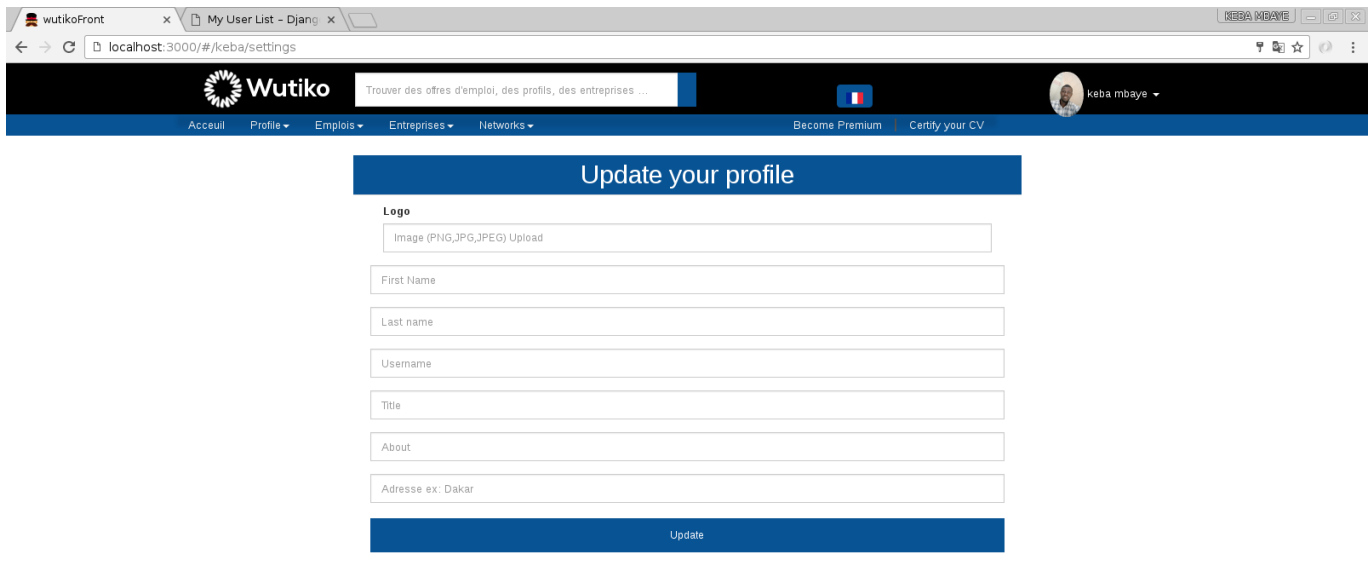


Schéma 23 : Mise à jour profil

4.3.5 Création d'une entreprise.

Tout recruteur a le droit de créer une entreprise afin de disposer d'une annuaire d'entreprises. De ce fait, le schéma 24 décrit les champs à remplir pour créer une entreprise.

The screenshot shows a web browser window with the URL `localhost:3000/#/company`. The page title is "Add company". The form contains the following fields:

- Name**: Enter here the name of the company or org
- Full name**: Enter here the full name of the company or
- Logo**: Image (PNG,JPG,JPEG) Upload
- Cover photo**: Image (PNG,JPG,JPEG) Upload
- Phone**: Enter phone number
- Mobile Phone**: Enter mobile phone number
- Email**: Enter the contact email
- Website**: Enter the website
- Size**: Dropdown menu
- Year of creation**: Dropdown menu
- Reach**: Dropdown menu
- Total revenue**: Dropdown menu
- Sector**: Dropdown menu
- Legal Form**: Dropdown menu
- Tags (Separated by commas)**: Text input field
- Board Members**: Section header

A callout box with an arrow points to the "Name" field, containing the text "Créer une compagnie".

The right sidebar shows a list of existing companies:

- Marketing
- santé
- Informatique
- wutiko (1 Jobs, 22 views, 0 likes)
- Atos (1 Jobs, 4 views, 0 likes)
- Gainde2000 (1 Jobs, 1 view, 1 like)

Schéma 24 : Création d'une entreprise

4.3.6 Liste des entreprises.

Les utilisateurs ont la possibilité de consulter la liste des entreprises. Comme le schéma 25 l'indique cette liste peut être filtrée par différentes paramètres afin d'accéder rapidement à l'information souhaitée.

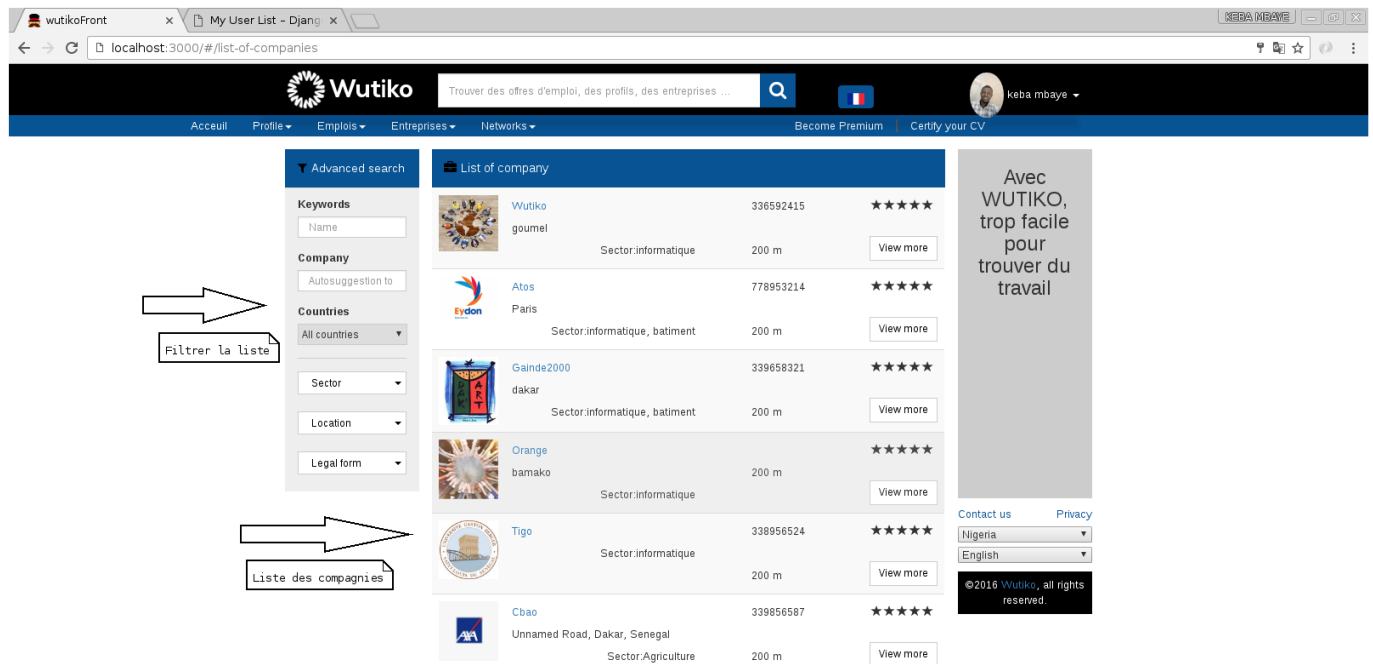


Schéma 25 : Liste des entreprises

4.3.7 Détails d'une entreprise.

Disposant d'une liste de entreprises, tout utilisateur peut accéder aux détails de ses dernières. Le schéma 26 décrit les informations d'une entreprise, les emplois postés, les administrateurs de cette compagnie, les autres lieux où la compagnie est implémentée et sa localisation via map.

The screenshot displays the Wutiko web application interface. The main content area shows the profile of a company named 'wutiko'. The profile includes a logo, contact information (phone number 336592415, website www.wutiko.com, and location in Senegal), and a 'Premium' badge. Below this is a summary table with the following data:

Field	Value
Full name	wutiko SA
Founded	14.01.1990
Revenue	2124577498-5158478
Industry	
Size	12642345-655656
Company type	SA
Company reach	international
Parent companies	Attijari Wafa Bank Group (MA)
Subsidiaries	CBIP (FR), Credit Du Sénégal (SN)

On the right side, there is a 'Jobs' section with a list of job categories: Marketing, santé, and Informatique. Below this, there are job listings for 'wutiko' (1 job), 'Atos' (1 job), and 'Gainde2000' (1 job). The interface also features a navigation menu at the top and a footer with the text '©2016 Wutiko, all rights reserved.'

Schéma 26 : Détails d'une entreprise

4.3.8 Créer un emploi.

Avec la concurrence très rude dans le secteur, nous avons décidé de laisser aux recruteurs de créer des emplois gratuitement pour un temps déterminé afin qu'il se familiarise avec notre application. Ainsi le schéma 27 décrit les champs à remplir pour créer un emploi.

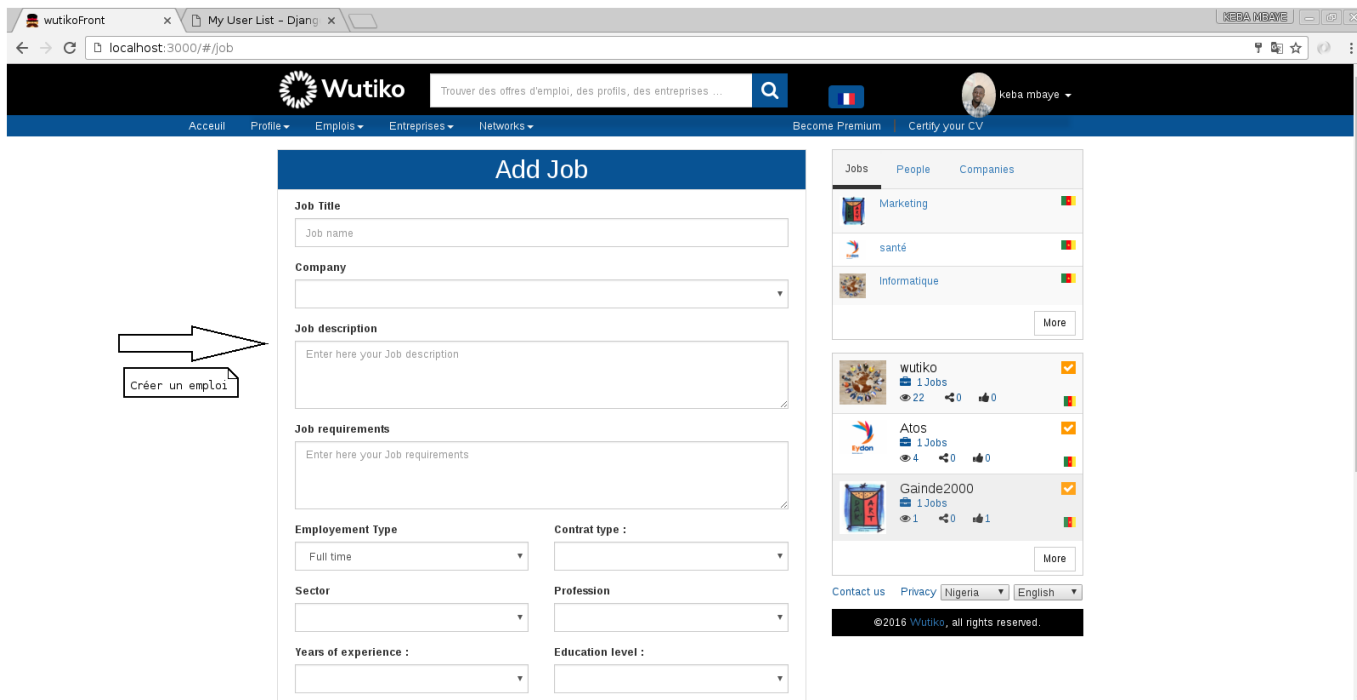


Schéma 27 : Créer un emploi

4.3.9 Liste des emplois.

Les utilisateurs ont la possibilité de consulter la liste des emplois. Comme le schéma 28 l'indique cette liste peut être filtrée par différentes paramètres afin d'accéder rapidement à l'information souhaitée.

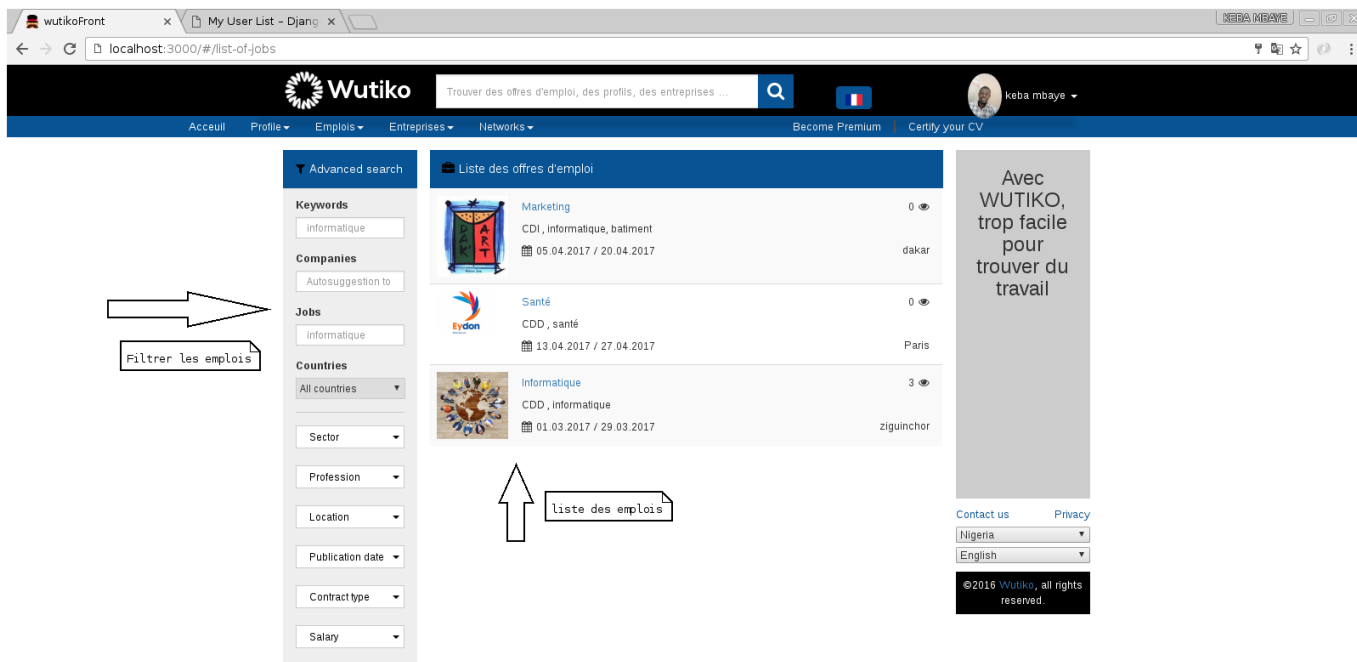
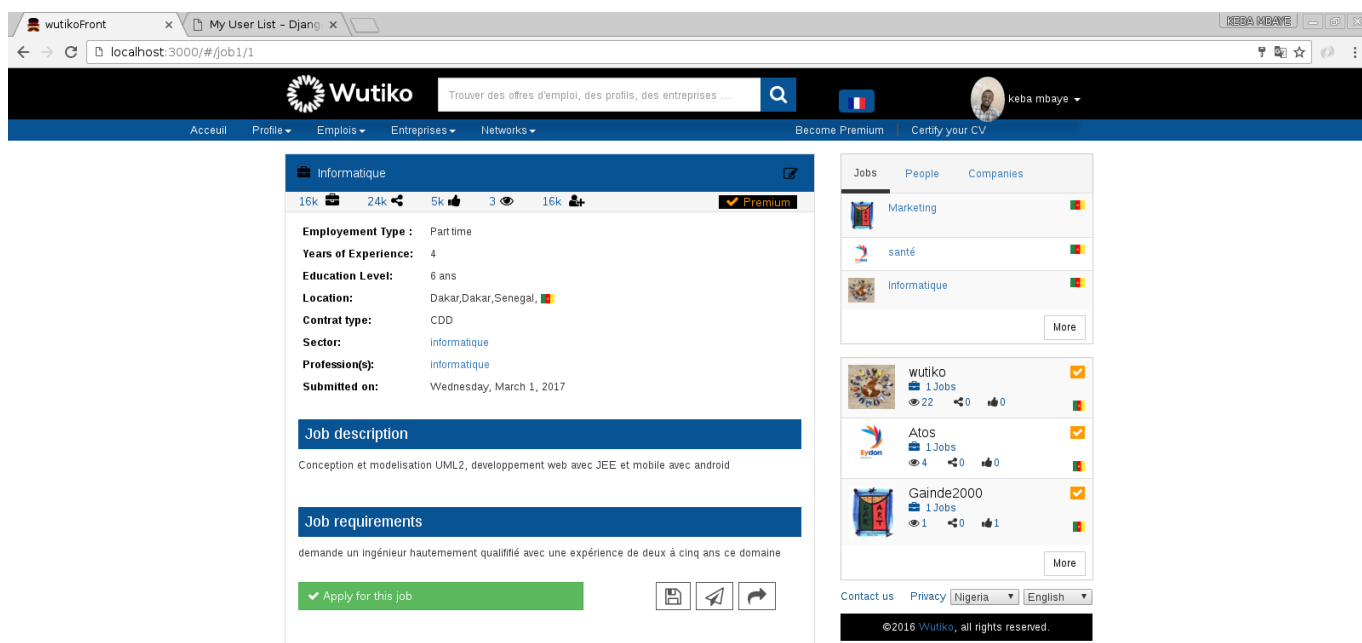


Schéma 28 : Liste des emplois

4.3.10 Détails d'un emploi.

Disposant d'une liste d'emplois, tout utilisateur peut accéder aux détails de cette dernière. Le schéma 29 décrit toutes les informations relatives à un emploi à savoir la description de l'emploi, les informations requises, le type de contrat...



The screenshot shows a web browser window displaying the Wutiko job details page. The browser address bar shows 'localhost:3000/#/job1/1'. The page header includes the Wutiko logo, a search bar with the text 'Trouver des offres d'emploi, des profils, des entreprises ...', and a user profile for 'keba mbaye'. The main content area is divided into two columns. The left column displays job details for 'Informatique', including statistics (16k views, 24k likes, 5k shares, 3 comments, 16k followers), a 'Premium' badge, and fields for Employment Type (Part time), Years of Experience (4), Education Level (6 ans), Location (Dakar, Dakar, Senegal), Contrat type (CDD), Sector (Informatique), Profession(s) (Informatique), and Submitted on (Wednesday, March 1, 2017). Below this is the 'Job description' (Conception et modélisation UML2, développement web avec JEE et mobile avec android) and 'Job requirements' (demande un ingénieur hautement qualifié avec une expérience de deux à cinq ans ce domaine). A green 'Apply for this job' button is at the bottom. The right column shows a list of related jobs under 'Jobs', 'People', and 'Companies' tabs, including 'Marketing', 'santé', 'Informatique', 'Wutiko', 'ATOS', and 'Gainde2000'. The footer contains 'Contact us', 'Privacy', 'Nigeria', 'English', and '©2016 Wutiko, all rights reserved.'

Schéma 29 : Détails d'un emploi

4.3.11 Emplois favoris.

Chaque utilisateur ayant appliqué à un emploi donné pourra faire le suivi des emplois appliqués via cette page. Le schéma 30 présente la liste des emplois appliqués par l'utilisateur connecté et un filtre sur la liste lui est proposé.

The screenshot shows the Wutiko web application interface. The browser address bar indicates the URL is localhost:3000/#/favorite-job. The page header includes the Wutiko logo, a search bar, and user information for 'keba mbaye'. The main content area is divided into three sections:

- Advanced search:** A sidebar with filters for Keywords, Companies, Countries, Location, and Contract type.
- My Favorites jobs:** A table listing favorite jobs with columns for Job name, Company, Contract type, Views, and Action.
- Promotional banner:** A grey box with the text 'Avec WUTIKO, trop facile pour trouver du travail' and contact information.

Job name	Company	Contract type	Views	Action
Informatique		CDD	800	
Marketing		CDI	800	
santé		CDD	800	

Contact us Privacy
Nigeria
English
©2016 Wutiko, all rights reserved.

Schéma 30 : Emplois favoris

4.3.12 Entreprises favorites.

Chaque utilisateur ayant appliqué aussi à une entreprise donnée pourra faire le suivi des entreprises appliquées via cette page. Le schéma 31 présente la liste des entreprises appliquées par l'utilisateur connecté et un filtre sur la liste lui est proposé.

The screenshot shows the Wutiko web application interface. The top navigation bar includes the Wutiko logo, a search bar with the text "Trouver des offres d'emploi, des profils, des entreprises ...", and a user profile for "keba mbaye". The main content area is titled "My Favorites companies" and displays a table of favorite companies. The table has columns for Logo, Company name, Secteur, Views, and Action. Two companies are listed: "wutiko" and "Orange", both in the "informatique" sector with 800 views. To the left of the table is an "Advanced search" sidebar with filters for Keywords, Companies, Countries, Location, and Sector. To the right is a promotional banner with the text "Avec WUTIKO, trop facile pour trouver du travail". At the bottom right, there are links for "Contact us" and "Privacy", a dropdown menu for "Nigeria", a dropdown menu for "English", and a copyright notice: "©2016 Wutiko, all rights reserved."

Logo	Company name	Secteur	Views	Action
	wutiko	informatique	800	
	Orange	informatique	800	

Schéma 31 : Compagnies favorites

4.3.13 Vue des recruteurs.

Cette vue permettra aux recruteurs de disposer d'une banque de curriculum vitae afin de voir les profils qui matchent avec leurs secteurs d'activité. Pour l'instant cette fonctionnalité est gratuite pour les mêmes raisons citées lors de la création d'emploi. Le schéma 32 ci-dessous présente la liste des utilisateurs ainsi que les filtres pour retrouver le profil souhaité rapidement.

The screenshot displays the Wutiko application interface. At the top, there is a navigation bar with the Wutiko logo, a search bar, and user information for 'keba mbye'. Below the navigation bar, there are several tabs: 'Accueil', 'Profil', 'Emplois', 'Entreprises', and 'Networks'. The main content area is divided into three sections:

- Left sidebar (Filtre):** Contains a list of filters such as 'Tous (1900)', 'CV ouverts (800)', 'Candidat(s) Retenus (500)', 'Candidats non retenus (410)', and 'Talents (s) Recrutés (90)'. Below this is an 'Advanced search' section with fields for 'Keywords', 'Companies', 'Countries', 'Education', 'Skills', and 'Location'. A callout box labeled 'Filtrer les utilisateurs' points to this section.
- Center (List of members):** Displays a list of user profiles. Each profile includes a name, location, function, education, and a 'View this profile' button. The profiles shown are:
 - Idrissa Sane (Dakar)
 - Mouhamed Badji (Thies)
 - Ibrahima Sidibe
 - Keba mbye Diedhiou (Dakar)
 - Mouhamed Badji (Louga)
 - Abdou Diop (Ziguinchor)
- Right sidebar:** Contains a promotional message: 'Avec WUTIKO, trop facile pour trouver du travail'. Below this is a copyright notice: '©2016 Wutiko, all rights reserved.' and a callout box labeled 'Banque de cv pour les recrute' pointing to the copyright notice.

Schéma 32 : Vue des recruteurs

4.3.14 Réseau social.

Cette vue permet aux utilisateurs de créer des liens d'amitiés ainsi que des groupes afin de les regrouper autour des thématiques. En effet tout utilisateur pourra faire des publications d'images, d'articles, de photos, de vidéos ou du texte. Le schéma 33 ci-dessous présente la page.

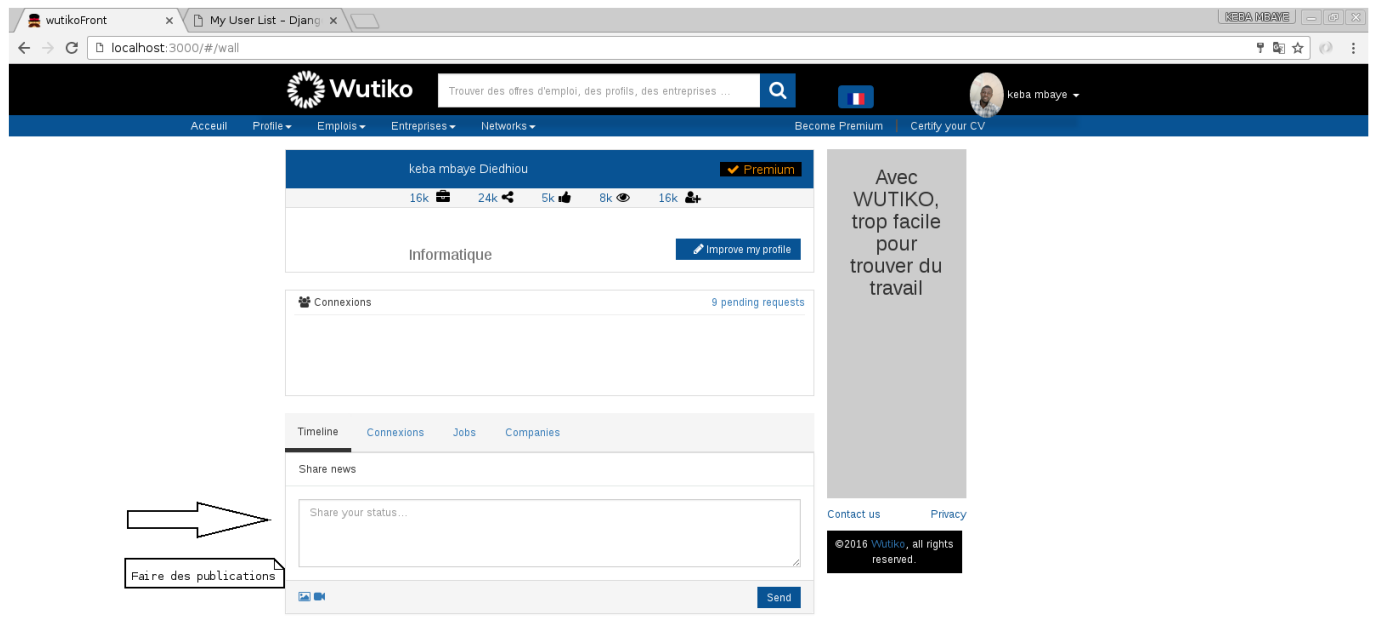


Schéma 33 : Réseau social

4.3.15 Barre de recherche.

cette barre permet de faire des recherches auto-complétées sur les utilisateurs, les entreprises et les emplois disponibles. Le schéma 34 ci-dessous montre une recherche auto-complétée sur un utilisateur donné.



Schéma 34 : Barre de recherche

Conclusion et Perspectives

5.1 Conclusion

Un service Web REST est très simple à utiliser (partie cliente) et relativement simple à écrire (partie serveur) à partir du moment où l'on fait appel à des bibliothèques existantes (pour notre cas nous avons utilisé Django REST Framework). Le seul prérequis est la connaissance de HTTP. Ce type d'architecture connaît un grand succès en ce moment, confirmé par l'intérêt d'acteurs tels que Google, Yahoo! ou encore Amazon. Une des raisons principales tient à l'importance croissante des navigateurs en tant que plateforme d'exécution sur les postes clients fixes mais également sur les terminaux mobiles. Les applications web sont en train de passer d'un mode document (le serveur retourne une mise en forme du résultat) à un mode desktop (le client charge la partie interface de l'application, les requêtes au serveur ne font plus que charger les données) essentiellement grâce à l'utilisation de JavaScript. C'est dans ce sens que nous avons porté notre choix sur ce type d'architecture afin d'apporter des solutions aux limites de l'architecture existante, ainsi nous avons procédé à la mise en place d'une nouvelle architecture, puis nous avons développé la partie web de l'application. Mais certaines fonctionnalités n'ont pas été développées et seront définies en guise de perspectives.

5.2 Perspectives

Dans le futur, nous comptons ajouter de nouvelles fonctionnalités afin de satisfaire les besoins des utilisateurs. Ainsi nous allons travailler sur :

- l'accès au niveau du backend par token pour les utilisateurs ;
- la disponibilité des ressources en xml ;
- l'ajout des fonctionnalités de social networking ;
- l'étude sur les données pour passer de services Web à sémantique Web.
- l'aspect sécurité ;
- le développement de la partie mobile est envisagé après le lancement de l'application web.

References

- [All10] Subbu Allamaraju, *Restful web services cookbook : solutions for improving scalability and simplicity*, " O'Reilly Media, Inc.", 2010.
- [Com] Licence Creative Commons, *Les services web*.
- [Fel10] Oliva Felipe, *Design and development of a rest-based web service platform for applications integration*, Master's thesis, Universitat Politècnica de Catalunya, 2010.
- [Fie00] Roy Thomas Fielding, *Architectural styles and the design of network-based software architectures*, Ph.D. thesis, University of California, Irvine, 2000.
- [Grö11] Benjamin Gröhbiel, *Rest engineering on the server-and client-side*, Software Engineering Group Department of Informatics University of Fribourg Switzerland (2011).
- [KCQ⁺06] S Krakowiak, T Coupaye, V Quéma, L Seinturier, J Stefani, M Dumas, and MC Fauvet, *Intergiciel et construction d'applications réparties*, Ecole d'été **13** (2006).
- [Pot11] Pavan Kumar Potti, *On the design of web services : Soap vs. rest*.
- [RR08] Leonard Richardson and Sam Ruby, *Restful web services*, " O'Reilly Media, Inc.", 2008.
- [urla] <https://trends.google.com>.
- [urlb] <https://www.openclassrom.com>.
- [urlc] <https://www.slack.com>.
- [urld] <https://www.google.com>.
- [urle] <https://www.youtube.com>.