Rapport de stage industriel

Étienne CHANARON
2e année, filière informatique

✦

le 25 octobre 2014

Table des matières

Introduction

Le présent rapport décrit les trois mois que j'ai passés dans l'entreprise *Applicatour* en tant que stagiaire, le travail et les solutions techniques retenues par l'entreprise, ainsi que celui que j'y ai effectué.

1 Présentation de l'entreprise

1.1 L'entreprise

Applicatour est une entreprise bordelaise ayant une antenne à Paris, composée d'un vingtaine de personnes qui, depuis à peu près 10 ans, fournit des solutions logicielles destinées à l'organisation de voyages touristiques (« voyage à la carte et sur mesure »[1]) de grande ampleur. Ses clients sont bien sûr majoritairement des *tour-operator*, mais pas seulement : un organisateur de pèlerinages ou une association organisatrice de séjours linguistiques à destination d'élèves de collège ou de lycée ont fait appel aux services d'*Applicatour*.

Elle est composée majoritairement de développeurs, mais aussi d'un service commercial et d'une direction de projets. Le faible nombre permet à tout le monde de se rencontrer facilement, ce qui est un atout pour la résolution des problèmes et la mise en commun des idées.

1.2 Mon au rôle au sein de l'entreprise

Dans le cadre mon stage, il m'a été demandé de participer au développement de plusieurs logiciels sur mesure pour différentes entreprises. Comme *Applicatour* a de nombreux clients, le nombre de logiciels est assez conséquent. On peut même les considérer chacun comme un micro-projet. Mais tous s'articulent autour d'une base commune, qui constitue le gros projet actuel d'Applicatour. Les besoins de celui-ci sont exprimés en fonction des demandes de tous les clients qui, si elles sont souvent très semblables, différent suffisamment pour obliger *Applicatour* à ajuster son offre et développer des fonctionnalités supplémentaires. Ce mode de fonctionnement sera expliqué plus en détails ultérieurement.

Au sein de chaque petit projet, j'ai la plupart du temps été chargé du développement du même module : celui qui se charge de calculer les devis et factures relatifs au voyage. À titre d'illustration, imaginons que l'entreprise X a chargé *Applicatour* de lui développer un logiciel de gestion. Quand X organise un voyage, elle crée une base de données répertoriant les passages, les dépenses, les activités etc. X doit envoyer un devis à l'entité qui finance le voyage (un lycée, des passagers, une entreprise, une famille…). Si le voyage s'est fait comme prévu, il faut le facturer au client. Pour des raisons d'efficacité, c'est le logiciel d'*Applicatour* qui se charge de rédiger le devis et la facture à partir de la base. Il lui faut donc effectuer quelques calculs financiers (conversion de devises, calculs de TVA, répartition du financement par participant en tenant compte de la qualité du participant…), ainsi que la mise en forme du document à produire, celle-ci étant susceptible de varier en fonction du contenu.

1. `http://www.applicatour.com/societe/qui-sommes-nous/`

J'ai donc été amené à lire plusieurs cahiers des charges et à m'y conformer en vue d'une évaluation par l'entreprise, puis par le client, celle-ci se faisant de manière quasi-continue par mail, par téléphone, ou même directement dans les locaux de l'entreprise.

2 Présentation de la mission technique et des solutions

2.1 La plateforme *Via System*

Applicatour développe *Via System*, logiciel de gestion de voyages organisé de manière très modulaire. En fonction des demandes, elle livre au client le logiciel et les modules appropriés. Comme les attentes des clients sont souvent assez différentes, il est nécessaire de développer ou d'adapter de nouveaux modules à chaque contrat. Parmi ces modules, il y en a toujours pour gérer les réservations des touristes (hôtel, moyens de transports etc.), leur emploi du temps au cours du voyage et les dépenses de l'instance organisatrice (professeurs, moniteurs, intendance…). Bien entendu, les modules ne sont pas étanches et indépendants, mais sont placés en interaction pour former un logiciel complet.

2.1.1 *Odoo* comme plateforme de développement de *Via System*

Via System se base lui-même sur *Odoo*[2], logiciel de gestion de manière plus générale[3] , utilisé ici comme plateforme de développement. Il se trouve que plusieurs de ces modules sont plus ou moins réutilisables, la gestion de voyages comportant des points communs avec la gestion d'entreprise, par exemple celui concernant la gestion de données.

2.1.2 Structure d'un module

Via System étant réalisé à l'aide d'une suite de modules *Odoo*, il peut lui-même être étendu par d'autres modules *Odoo*.

Tout d'abord, il faut savoir qu'*Odoo* est programmé en majorité en langage *Python*. L'accès et les modifications se font via un navigateur web.

Ensuite, la structure est toujours la même (cf. figure 1[4].) :

— Une partie web, écrite en *XML*, qui décrit l'interface de communication avec le serveur où sont stockées les données de la BD (dans le répertoire *static*).
— Une partie gérant la communication avec le serveur (dans le répertoire *security*).
— Trois fichiers, *__init.py__*, *__openerp.py__* et *nom_du_module.py*, qui constituent le cœur du module.

On remarquera que c'est le langage *Python* qui fait « tourner » le module, donc par extension *Odoo* tout entier.

2. Anciennement *OpenERP*.
3. Selon `https://www.odoo.com/`, « logiciel libre de gestion d'entreprise ». La gestion d'entreprise est facilitée par le nombre de modules dédiés à cet effet développés par les concepteurs du logiciel (peut-être pour des raisons publicitaires), mais ce n'est nullement l'unique possibilité qu'offre *Odoo*.
4. Tiré de `http://thierry-godin.developpez.com/openerp/`

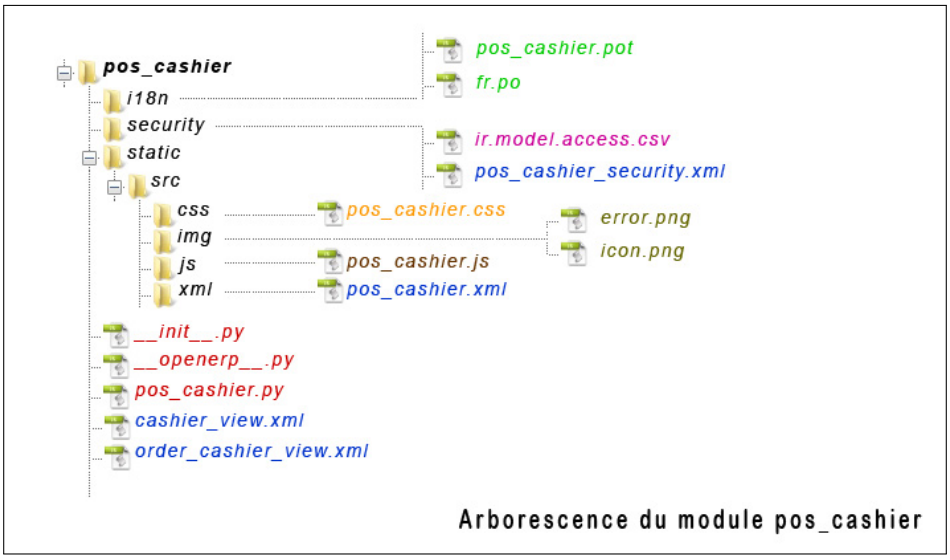

FIGURE 1 – Structure d'un module *Odoo*

Le fichier *nom_du_module.py* contient la définition de la classe `nom_du_module`. Cette classe comporte entre autres attributs `_columns`, un dictionnaire d'entrées de tous types (des booléens, des char...), qui se retrouveront comme champs dans une table `nom_du_module` de la base de données. La gestion de la base se fait de manière totalement transparente, le programmeur du module n'ayant à coder que du *XML* pour les interfaces et du *Python* pour tout le reste.

Concrètement, l'utilisateur aura à sa disposition une interface web voulue conviviale lui permettant de gérer son module, ici de gestion gestion de voyages. Il pourra créer un nouveau voyage, lui ajouter des utilisateurs et des activités, faire des réservations d'avions ou des locations de voitures, créer un devis...

2.2 Le module de production des devis : de *Mako* à *QWeb*

Elle dépend des clients. Parfois, elle est suffisamment complexe pour être développée en tant que module à part entière, parfois, les fonctions offertes par *Odoo* suffisent pour n'y consacrer qu'une partie d'un module.

La principale difficulté conceptuelle, assez analogue à ce qui ce rencontre en programmation web, vient de ce qu'il faut associer de manière élégante la récupération et le traitement d'informations provenant d'une base de données à la mise en forme et l'affichage final. C'est cette difficulté qui induit de nombreux changements au cours du développement d'*Odoo*. L'observation des différentes solutions retenues au fil du temps est assez intéressante et donne un aperçu de la réflexion des développeurs.

2.2.1 Avec *Mako*

Dans un premier temps, à mon arrivée, la technologie *Mako* était utilisée. En voilà le fonctionnement global [5] :

5. cf. `http://www.makotemplates.org/`

Le code *Python* (opérations arithmétiques, boucles...) est directement intégré au code *HTML* entre des balises `<%` et `%>` (code multiligne éventuellement).

Le code *Python* ayant un impact sur l'affichage *HTML* (par exemple l'affichage conditionnel) est lui aussi directement intégré au code, mais après le symbole `%` seul.

```
<%
    # Ceci est du Python pur...
    n = 0
    n += 1
%>
<!-- et ceci est du HTML avec du pseudo code Python. -->
<p>
    Ceci est un texte dans un paragraphe :
    % if n != 0:
        valeur nulle
    % else
        valeur non nulle
    % endif
</p>
```

produira l'affichage : « Ceci est un texte dans un paragraphe : valeur non nulle ».

Enfin, pour afficher des données écrites sur la base, il faut utiliser `${}`, par exemple :

```
<p>Affichage de la donnee : ${data}</p>
```

L'avantage du *Mako* est la possibilité d'utiliser la puissance du *Python* et de bénéficier de sa syntaxe claire. On voit (l'usage le confirme) que cette manière de procéder nous conduit à écrire du code lourd et difficilement lisible où il faut faire un effort pour séparer les traitements des affichages. Voilà pourquoi l'équipe de développement d'*Odoo* a effectué un revirement et a remplacé *Mako* par *QWeb* comme langage de sortie pour produire des documents de tournure professionnelle.

2.2.2 Avec *QWeb*

QWeb est un langage entièrement basé sur *HTML*, dont il partage certaines balises. Parmi les balises spécifiques au langage, on notera `<t></t>`, qui sert a donner des « directives » [6] dont le type dépend de l'argument fourni.

Ainsi, l'affichage conditionnel se fait avec `<t t-if="[condition]"></t>` de la manière suivante :

```
<t t-if="condition">
    <h1>Condition vraie</h1>
</t>
```

6. La documentation d'*Odoo* précise : « A directive will usually control or alter the output of the element it is set on. »

Dans cet exemple, si la condition est vérifiée, le texte « Condition vraie » s'affichera sous la forme d'un titre de niveau 1.

De manière semblable, on peut effectuer des boucles ou un parcours de données dans la base via `<t t-foreach="[condition de parcours]"></t>`.

Une autre balise très importante est `<span />` qui permet d'afficher des données de différents types provenant ou non de la base de données. Un large éventail d'options permet de formater assez finement les données à afficher (horaires suivant les habitudes de différentes langues, monnaies et unités SI...)

Par exemple, si l'on veut afficher une date de la base sur le document, on fera appel à

```
<span t-field="[data]" />
```

Pour une heure écrite en langue naturelle,

```
<span t-field="[time]"
      t-field-options='{"widget":"duration",
                        "unit":"hour"}' />
```

Remarquons que les options sont fournies sous forme de dictionnaire *Python*, ce qui illustre bien les liens forts entre *Python* et *Odoo*.

Profitons-en d'ailleurs pour donner un aperçu de la manière dont les données sont gérées dans *Odoo* : à chaque entrée dans la table correspond une variable en *Python* etc.

Avec *QWeb*, l'on dispose d'un langage de description de document peut-être moins puissant que *Mako* mais suffisant pour une utilisation courante, et en tout cas beaucoup plus esthétique conceptuellement et certainement plus lisible, donc en définitive plus efficace.

2.3 La gestion des données : de *PostgreSQL* à *Neo4j*

Parler de gestion implique nécessairement un grand nombre de données à gérer (pour se donner une idée de l'ordre de grandeur, les voyages organisés par les clients peuvent impliquer des milliers de personnes). Il est donc nécessaire de faire appel à un *SGBD*. Il se trouve qu'*Odoo* propose des solutions pour gérer une base de manière transparente, ce que *Via System* utilise largement, par le biais de *PostgreSQL* et du standard *SQL*.

2.3.1 Pourquoi utiliser *Neo4j* ?

L'équipe a remarqué que de nombreux problèmes demandés par les clients peuvent se réduire à des problèmes classiques de graphes, par exemple de plus court chemin. Précisément, les clients de type *tour-operator* stockent les données relatives aux voyages et aux voyageurs. Pour des raisons commerciales, le *tour-operator* peut se poser les questions suivantes en vue de rentabiliser son activité et de proposer à ses

propres clients des solutions adaptées :

— Qui a voyagé avec qui ?
— Quelle est la distance entre deux personnes (distance au sens du nombre de connaissances intermédiaires) ?
— ...

Concrètement la manière naturelle de modéliser le système est de stocker les voyages et les voyageurs sous forme de nœuds comportant des attributs tels que le nom, l'âge, la destination etc, puis de relier voyages et passagers par des arêtes du type « a voyagé dans ».

Pour répondre à ce problème, il existe des *SGBD* dits « orientés graphes », concept en plein expansion. *Applicatour* a retenu celui nommé *Neo4j*. Ceux-ci sont plus efficaces que les *SGBD* traditionnels dans le cas où les requêtes peuvent être modélisées par des problèmes de parcours de graphes.

2.3.2 L'utilisation de *Neo4j*

Neo4j est un produit tout récent qui a été retenu par Applicatour : la version 1.0 de Neo4j est sortie en 2010. Néanmoins, le développement ayant commencé depuis de nombreuses années (2000 précisément), le logiciel comprend une documentation complète qui possède l'avantage d'être très à jour. De plus, le tutoriel officiel est très bien conçu.

Neo4j est codé en Java et le code est réduit (quelques *.jar*), ce qui assure la portabilité des applications. Pour donner une idée de sa puissance, le site officiel assure que *Neo4j* peut gérer des millions de nœuds [7]. Bien sûr, les propriétés ACID sont respectées [8].

Il est aussi très facile d'utiliser un serveur local pour effectuer des tests : il suffit de lancer un petit programme et d'ouvrir `http://localhost:7474/browser/`.

L'accès au serveur se fait de manière graphique via un navigateur web quelconque (testé sans problèmes sous Firefox et Chrome). L'interface est très réactive et esthétique. Elle est constituée de deux parties :

— une invite de commandes permettant d'interagir avec le serveur sous forme de requêtes en *Cypher*.
— une visualisation particulièrement bien réalisée des résultats sous forme de listes (une visualisation qui rappelle les SGBD classiques) ou de graphes (une autre très puissante et concise).

Enfin, une API *Java* (donc orientée objet) permet d'utiliser *Neo4j* dans un logiciel.

2.3.3 Le langage de requêtes : *Cypher*

Ce langage, nommé *Cypher*, a été conçu pour faciliter la transition avec le traditionnel *SQL*. Ainsi les deux syntaxes, quoique différentes, sont de la même famille.

SQL	*Cypher*
`SELECT * FROM base;`	`MATCH (n) RETURN n;`

7. « up to several billion nodes/relationships/properties »
8. cf. `http://www.neo4j.org/learn/neo4j`.

Étant donné le caractère très visuel des graphes, les requêtes en *Cypher* ont été conçues pour évoquer au premier coup d'œil les nœuds et leurs relations, de manière semblable à l'« art ASCII ». Ainsi, le nœud n est représenté par `(n)` tandis que l'arête r se note `-[r]->`. Afin d'illustrer la conception intuitive de *Cypher*, pour indiquer au SGBD les nœuds n et p reliés par l'arête r allant de n à p, on utilisera la syntaxe `(n)-[r]->(p)`.

Par exemple, pour obtenir la totalité des nœuds de type `Toto` vers lesquels pointent exactement deux autres du même type, on aura recours à

```
MATCH (:Toto)-->(p:Toto)<--(:Toto)
RETURN p;
```

Remarquons encore comme preuve de la clarté du langage qu'il n'est pas nécessaire de préciser de nom aux arêtes si on ne désire pas s'en servir.

Bien évidemment, comme les graphes constituent le cœur de *Neo4j*, le SGBD peut effectuer des calculs de plus courts chemins [9], en donner la longueur ainsi que d'autres opérations classiques.

3 Ma réalisation

3.1 La réalisation de modules de calcul de devis

3.1.1 Nature de mon travail

Afin de m'habituer à *Odoo* et aux outils de l'entreprise, on m'a demandé d'abord de ne me préoccuper que de la partie visuelle du module de devis (le *CSS*) pour un client appelé *Vivalangues* (organisateur de voyages touristiques à visée d'apprentissage pour des élèves de l'enseignement secondaire). La partie description (*HTML*) et traitements (module *Python*) étant fournies. Ce travail n'étant pas particulièrement difficile, j'ai rapidement présenté une version qui a été soumise au client.

Bien entendu, ce dernier a demandé quelques modifications mineures ainsi que des ajouts de fonctionnalités (par exemple un menu avec des cases à cocher par le signataire, des tableaux de compatibilité...). J'ai donc été obligé de m'occuper de la structure du fichier *HTML* correspondant. Là, j'ai effectué quelques retouches puis, constatant que j'avançais avec peine à cause du non respect des conventions *HTML5* (majoritairement des insertions de mise en forme entrant en concurrence avec le *CSS*) et de choix de champs peu judicieux, j'ai décidé de reprendre à partir du début.

Là, je me suis rendu compte des motivations qui avaient amené à certains choix, que j'ai été obligé de réintégrer malgré leurs défauts. Finalement, j'ai pu présenter une version corrigée et correspondant du mieux possible aux exigences de *Vivalangues*. Comme ce client est exigeant et assez ignorant des problèmes informatiques, il demande souvent des changements faussement mineurs. C'est pourquoi tout au long du stage j'ai eu à retoucher ce module alors même que je ne m'en occupais plus depuis plusieurs semaines.

9. cf. documentation : c'est l'algorithme de Dijkstra qui est utilisé.

Par la suite, j'ai été intégré au développement de modules dont l'état d'avancement était moindre, ce qui m'a obligé à intégrer mes propres fonctions au module, par exemple pour des calculs de dates.

Enfin, j'ai dû m'attacher à comprendre la structure de la base de données afin de pouvoir utiliser son contenu, ce qui m'a pris plusieurs jours de rétro-ingéniérie étant donnée son immense étendue, l'absence totale de documentation par les développeurs d'*Applicatour* et les vacances du principal responsable. Pour cela, j'ai renoncé à lire le détail des tables pour me concentrer sur les classes *Python* à partir desquelles elles sont générées automatiquement.

3.1.2 Exemples de réalisations

Voici un fragment de définition de la classe qui gère les voyages pour *Vivalangues*. Au chargement du module, la base de données crée ou met à jour des attributs dont les noms sont décrits dans l'attribut `_columns`. Elle comprendra ainsi une table `via_trip` contenant des attributs `client_note`, `departure_date` etc.

```
class VIA_Trip(osv.Model):

    _name = 'via.trip'

    _columns = {
        'budget': fields.float('Budget'),
        'client_note': fields.text("Client note"),
        'color': fields.integer('Color Index'),
        'c_datetime': fields.date('Confirmation date'),
        'departure_date': fields.date('Departure date'),

# et caetera
```

Les pages qui suivent montrent des exemples de devis calculés par les modules auxquels j'ai contribué. J'ai essayé de me conformer autant que possible au cahier des charges (seule est faite ici l'économie des logos des entreprises sur ces exemples) fourni par le client, même si l'esthétique imposée est parfois discutable... J'ai proposé de petites améliorations visuelles, qui n'ont en général pas été acceptées par le client, en raison de la force des habitudes datant de l'époque où tout était géré de manière artisanale. En revanche, pour la structure, le modèle et les calculs, toute latitude nous était offerte. Comme cela engageait notre réflexion, c'était réellement la partie intéressante du travail (mais aussi la plus difficile).

Document édité le 05/08/14

N° de devis	**SO3**
Votre contact	**Administrator**
admin@example.com	

Du 24/07/2014 au 26/08/2014
Devis au nom de : Jacques Martin
3 participants :
Jacques Martin
Jean Dupont
Marie Dupuis

À l'attention de :

Administrator
Jetset

Devis

Hôtel

Du 24/07/2014 au 27/08/2014

☆☆☆☆☆
No passenger...

Dispo du jour
Montant : 0,00

Rome

Voiture

Du 24/07/2014 à 10,0
Au 26/08/2014 à 10,0

No passenger...

Dispo du jour
Montant : 0,00

Merci de noter que le client doit avoir plus de machin truc etc.

Vol Paris-Rome

Du 24/07/2014 au 26/08/2014

3 adults

Dispo du jour
Montant : 407,58

Rome

Votre vol

Paris, France	**Rome, Italy**
24/07/2014 à 10,0	

Votre vol

Rome, Italy	**Paris, France**
26/08/2014 à 10,0	

Total :

DEVIS CONSTITUANT L'OFFRE PRÉALABLE en date du 15-07-2014

(à compléter et à renvoyer pour acceptation accompagné de la demande d'inscription, du programme validé, du coupon-assurances et d'un acompte de 30% ou bon de commande administratif dûment renseigné)

***ÉTABLISSEMENT* : lycée Jean-Jacques Rousseau**

Représenté par les soussignés *(à remplir obligatoirement)*

Nom : Jacques Martin	Fonction : intendant
Nom : Roger Duport	Fonction : chef d'établissement
Nom : Marie Duvent	Fonction : professeur

DESTINATION : Australie
Départ : jeudi 17 juillet 2014
Retour : vendredi 22 août 2014
Nombres de nuits sur place : 36

***CONDITIONS FINANCIÈRES* pour**	***PRIX GLOBAL***
séjour	0.0€
Aérien	0.0€
MONTANT ESTIMÉ *sauf application de l'article R211-10 du Code du tourisme*	0.0€
ACOMPTE dès réception de facture OU à la commande	30%
2ème ACOMPTE - 2 mois avant le départ	40%
SOLDE à payer (hors option supplémentaire) à la remise des documents	30%

RÈGLEMENT DU VOYAGE :

□ chèque □ virement
(copie d'état de mandatement à transmettre par fax)

NB. Le carnet de voyage ainsi que les dernières informations ne seront adressés qu'à réception du solde du séjour (article R211-8 du Code)

ITINÉRAIRE :

Cf. programme

INFORMATIONS PRÉALABLES :

SANTE et INFORMATIONS relatives au pays visité : www.diplomatie.gouv.fr - www.pasteur.fr
Chaque participant doit être muni de ma CEAM (Carte Européenne d'Assurance Maladie). Valable 1 an, elle est délivrée gratuitement par la CPAM.

FORMALITES / **POLICE** :
il appartient à l'établissement de vérifier la validité des documents ainsi que les formalités pour les ressortissants etrangers.

X Carte Nationale d'identité OU **X** Passeport et **X** Visa

KELEY consulting

Facture

Facture n° 2014 07 009 en date du 02/07/2014

Référence client :

MERIAL SAS
contact interne : Marie-Paule LADRAT
27, avenue Tony Garnier
69348 Lyon BP 7123

Dossier :

bon de commande n° 51465415614-2-4651
BD juin 2014

Montant HT	4565,00 €
TVA à 20 %	913,00 €
Total TTC	5478,00 €

En votre aimable règlement par chèque à l'ordre de PGCE ou par virement :

Banque	Guichet	Compte	Clé	IBAN
10207	00046	7865 1234 4520 127	33	FR76 4200 7895 1235 1234 7561 133

Adresse SWIFT : XXXXXXXX
Facture réglable [...]
n° de TVA intracommunautaire : AAXX XXX XXX XXX

Pas d'escompte par paiement anticipé [...]

3.2 L'intégration des bases de données orientées graphe

Deux faits s'opposent à cette intégration :

— Tout d'abord, nous avons vu dans la partie précédente que *Neo4j* était programmé en Java, ainsi que l'API officielle. Malheureusement, c'est plutôt le langage Python qui constitue le cœur d'*Odoo*. C'est pourquoi le seul moyen d'utiliser *Neo4j* dans *Odoo* est d'utiliser la bibliothèque *Py2neo*[10].

— Ensuite, *Odoo* est conçu pour interagir avec un SGBD relationnel « traditionnel » : PostgreSQL. L'utilisation de *Neo4j* conjointement avec *Odoo* nécessite donc un travail de conversion des bases loin d'être trivial.

On m'a demandé de trouver une solution à ces deux problèmes dans les derniers jours, tout en sachant que je ne m'aventurais pas dans un sentier battu, puisque le problème avait été très peu abordé par l'équipe d'*Applicatour* (ce qui m'obligeait d'ailleurs à me débrouiller à peu près seul, contrairement à la majeure partie de mon stage). À titre d'exemple, j'ai commencé par effectuer une retranscription sommaire (c'est-à-dire des principaux champs seulement) de la base *PostgreSQL* utilisée par *Via System*. Ensuite, j'ai essayé de comprendre comment agir sur cette base via l'API *Python* fournie par *Py2neo*.

Malheureusement, le temps m'a fait défaut. Un mois supplémentaire m'aurait certainement permis de savoir dans quelle direction concentrer mes efforts afin d'implémenter un début de solution.

J'ai pu constater que cette nouvelle technologie, bien que peu présente actuellement dans les produits de l'entreprise, est sérieusement à l'étude, puisque c'est une des premières choses que l'on m'a fait découvrir à mon arrivée.

3.3 Difficultés rencontrées

Travaillant sur des logiciels encore en production et non commercialisés, les *bugs* étaient très fréquents lors des modifications de l'équipe, ce qui m'a obligé à changer d'activité périodiquement en attendant les corrections.

Un autre souci particulièrement pénible est venu des changements fréquents de version d'*Odoo*, malheureusement non compatibles entre elles pour profiter des toutes nouvelles fonctionnalités, très souvent instables (il se trouve en effet que ce sont les versions alpha d'*Odoo* qui ont été utilisées pendant l'été pour gagner du temps en attendant la sortie officielle). Chaque fois que je devais travailler pour un client différent, il fallait que j'installe la version utilisée au moment du développement. Cette instabilité m'a permis de trouver un certain nombre bugs, ce qui a sans doute aidé à améliorer la robustesse des produits.

Enfin, le logiciel étant encore au stade de développement, les messages d'erreur étant non explicites ou seulement compréhensibles par quelqu'un qui maîtrise la structure du code *Pyhon* (du type « tel attribut n'existe pas dans telle classe ») voire complètement ésotériques (« erreur de l'interpréteur de *JavaScript* »), il m'a été nécessaire de m'immerger largement dans le code, de suivre des tutoriels et de mieux comprendre le développement de modules dans *Odoo*. C'est ainsi par exemple que j'ai mieux progressé dans la compréhension de l'interaction entre le code *Python* des modules, *Odoo*

10. *Py2neo* a été programmée par un membre de la communauté d'utilisateurs.

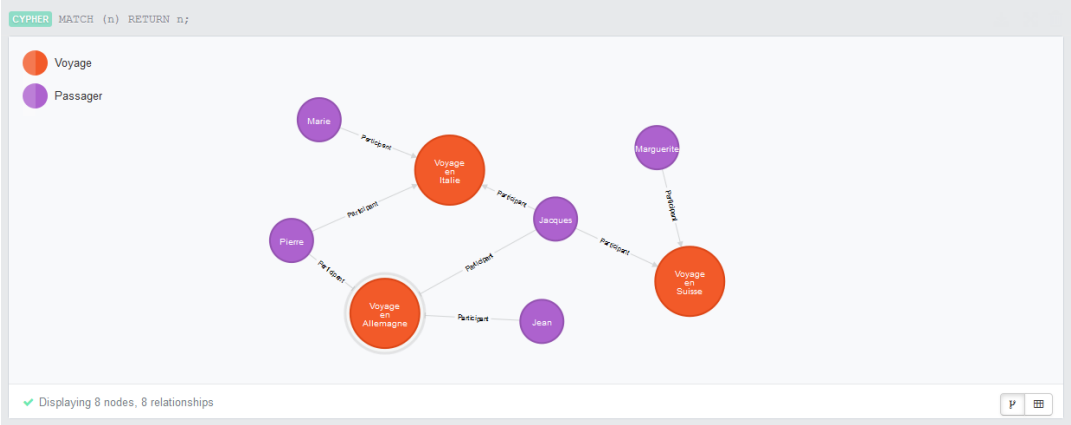

FIGURE 2 – Exemple de modélisation sous Neo4j

et la base de données. Quoiqu'il en soit, et comme l'affirme le site officiel, il faut au moins 6 mois pour maîtriser tous les ressorts du logiciel, ce qui implique qu'il me reste beaucoup de choses à découvrir du développement à l'aide de cet outil.

Finalement, je suis particulièrement satisfait d'avoir pu contribuer de manière utile à un projet réel, ce qui constitue pour moi une première expérience. Apparemment, mon travail a été suffisamment apprécié pour que la validation ne prenne pas une place disproportionnée dans développement.