

*Formation professionnelle  
Concepteur / Développeur*

# Développement de services Web JAX-RS pour l'import de données voirie et transport en commun



**Bertrand GUERRERO**

Responsables de stage : Christophe LAPIERRE / Julien LESBEGUERIES

Date de soutenance : 17.07.2015



# Remerciements

Je tiens tout d'abord à remercier mon responsable de stage Christophe Lapierre qui m'a permis de réaliser mon stage dans les meilleures conditions possibles. Je le remercie pour ses conseils, le temps qu'il m'a consacré et pour ses commentaires sur mon travail.

Je remercie mon maître de stage Julien Lesbegueries pour m'avoir intégré dans son projet, m'avoir permis au cours du stage de découvrir des technologies telles que Dropwizard, merci pour sa pédagogie et pour toutes les bonnes pratiques de développement qu'il m'a transmises.

Merci à mes deux « super » formateurs Gilles Vanderstraeten (Java SE) et Laure Bouquety (Java EE), pour leurs cours et leur support. Ils m'ont appris et fait aimer la programmation orienté objet et plus spécialement le langage Java.

Merci à Florian pour les nombreuses pauses à discuter, et aussi à Wafi pour ses jolis dessins de réseaux ;-).

Enfin, je voudrais remercier particulièrement Frédéric Schettini de m'avoir donné cette opportunité de stage qui rentre pleinement dans mon projet professionnel, tant par les aspects techniques que par leurs domaines d'application.



# Résumé

Au sein de MobiGIS <sup>1</sup>, société éditrice de logiciel SIG-Transport et société de service en géomatique, j’ai intégré le projet de Recherche & Développement « MobiSAAS ».

Mon sujet de stage porte sur le développement de web services permettant d’exposer des fonctionnalités d’importation de données voirie et transport en commun, pour la constitution automatique de réseaux de transports multi-modaux.

L’objectif de mon travail a été d’exposer des fonctionnalités codées initialement en langage SQL et/ou Python via une API REST en langage Java (JAX-RS). J’ai donc dans un premier temps pris en main les chaînes de traitements (Python) et appris à manipuler les données (Postgis/SQL) via le projet « DataWizard ». En parallèle, afin d’implémenter ces web services j’ai intégré le projet « MobiSAAS » et ainsi appris à développer avec le framework « Dropwizard ».

L’application développée dans ce projet présente en mode SAAS <sup>2</sup> les principales fonctionnalités du logiciel Desktop MobiAnalyst développé par l’entreprise. C’est dans le cadre de l’interface d’administration de l’application (backend) que je développe des fonctionnalités d’upload de données de transport (GTFS <sup>3</sup>) sur le serveur. Les perspectives de ce stage seront de généraliser les classes et méthodes utilitaires que j’ai développées, afin que l’application supporte plusieurs autres formats de données de transport (données vectorielles « Shapefile » par exemple). Ce projet devra à plus long terme exposer les fonctionnalités du logiciel DataWizard en mode SAAS.

A l’occasion de ce stage j’ai pu travailler sur plusieurs projets : MobiSAAS, DataWizard, Crislab, etc. J’ai donc participé à différentes phases du cycle de vie d’un projet de développement logiciel : depuis la conception R&D (MobiSAAS), au développement de code métier (DataWizard), jusqu’à la livraison au client (Crislab).

---

1. <http://www.mobigis.fr/>

2. [https://en.wikipedia.org/wiki/Software\\_as\\_a\\_service](https://en.wikipedia.org/wiki/Software_as_a_service)

3. <https://developers.google.com/transit/gtfs/>

# Abstract

At MobiGIS<sup>4</sup>, GIS and Transport software publisher and geomatics services company, I joined the Research & Development « MobiSAAS » project.

My internship subject focused on developing web services that exposes functionalities of road and transit data import, for the automatic build of multi-modal transport networks.

The purpose of my work was to expose functionalities originally coded in SQL and/or Python languages via a REST API in Java (JAX-RS). I therefore managed the business treatment (Python) and learned to handle data (PostGIS/SQL) via the « DataWizard » project. In the same time, in order to implement these web services I joined the « MobiSAAS » project and thus learned to develop with the « Dropwizard » framework.

The developed application of this project is similar to a SAAS<sup>5</sup>, the main features of MobiAnalyst Desktop software. It is in the context of the administration interface (backend) that I developed features to upload transit data (GTFS<sup>6</sup>). The prospects of my work will be to generalize the utility classes and methods I have developed so that the application may support multiple data formats of transport data (vector data « Shapefile » for example).

During this internship I have worked on several projects : MobiSAAS, DataWizard, Crislab, etc. This way, I participated in various phases of software development lifecycle : from the design (R&D) to development of business code, until the delivery to the customer.

---

4. <http://www.mobigis.fr/>

5. [https://en.wikipedia.org/wiki/Software\\_as\\_a\\_service](https://en.wikipedia.org/wiki/Software_as_a_service)

6. <https://developers.google.com/transit/gtfs/>

# Sommaire

<b>Sommaire</b>	<b>vi</b>
<b>Table des figures</b>	<b>1</b>
<b>1 Introduction</b>	<b>2</b>
<b>2 Présentation de Mobigis</b>	<b>4</b>
2.1 Domaine d'application . . . . .	4
2.2 Présentation de l'entreprise . . . . .	5
2.3 Activités . . . . .	6
2.4 Historique . . . . .	6
2.5 Ressources de l'entreprise . . . . .	8
<b>3 Contexte du stage</b>	<b>10</b>
3.1 Environnement de travail . . . . .	10
3.2 Outils utilisés . . . . .	10
3.3 Gestion de projets . . . . .	11
3.4 Difficultés rencontrées . . . . .	11
<b>4 Les Projets</b>	<b>13</b>
4.1 MobiSAAS . . . . .	13
4.1.1 Cahier des charges . . . . .	13
4.1.2 Eléments de spécifications fonctionnelles . . . . .	16
4.1.3 Eléments de spécifications techniques . . . . .	18
4.1.4 Réalisations . . . . .	24
4.1.5 Perspectives . . . . .	29
4.2 DataWizard . . . . .	31
4.2.1 Cahier des charges . . . . .	31
4.2.2 Eléments de spécifications fonctionnelles . . . . .	31
4.2.3 Eléments de spécifications techniques . . . . .	32
4.2.4 Réalisations . . . . .	33
4.2.5 Perspectives . . . . .	33
4.3 Crislab . . . . .	34
4.3.1 Présentation . . . . .	34
4.3.2 Réalisations . . . . .	35

<b>5</b>	<b>Bilan</b>	<b>37</b>
5.1	Bilan professionnel . . . . .	37
5.2	Bilan personnel . . . . .	37
<b>6</b>	<b>Conclusion</b>	<b>38</b>
<b>7</b>	<b>Glossaire et définitions</b>	<b>39</b>
<b>8</b>	<b>Annexes</b>	<b>43</b>
8.1	Redmine . . . . .	43
8.2	Classe Upload.java . . . . .	44
8.3	Classe UploadDAO.java . . . . .	46
8.4	Extrait du log dans Eclipse . . . . .	47
8.5	Structure d’une application Web . . . . .	49
8.6	Requête/Réponse . . . . .	50



# Table des figures

2.1	Offres de services Mobigis . . . . .	5
2.2	Exemples de clients Mobigis . . . . .	6
4.1	Interface de MobiSAAS (frontend) . . . . .	13
4.2	Offres du produit MobiAnalyst . . . . .	14
4.3	Architecture globale de « MobiAdmin » . . . . .	16
4.4	Contenu des données GTFS . . . . .	17
4.5	Extrait du fichier POM, versions des dépendances utilisées . . . . .	19
4.6	Organisation conseillée d'un projet Dropwizard . . . . .	21
4.7	Organisation du module mobi-admin . . . . .	21
4.8	Réponse Json renvoyée . . . . .	23
4.9	Rapport Json de validation des données GTFS . . . . .	24
4.10	Table des uploads sur le SGBD PostgreSQL . . . . .	25
4.11	Classes des objets GTFS (couche modèle) . . . . .	26
4.12	Modèle relationnel des données GTFS (couche modèle) . . . . .	26
4.13	Interface et exemple de requête « GET » SoapUI . . . . .	29
4.14	Exemple d'abstraction et d'héritage de méthode . . . . .	30
4.15	Métadonnées de réseau de transports v3.0 . . . . .	32
4.16	Fonction Python pour exporter des métadonnées . . . . .	34
4.17	Architecture globale de l'application Crislab . . . . .	35
8.1	Structure typique d'une application Web (Laure Bouquety ©) . . . . .	49

# Introduction

---

Ce stage s'inscrit dans le cadre de la formation "Concepteur / Développeur Informatique" délivrée par BGE Haute-Garonne et s'est déroulé durant 3 mois au sein de l'entreprise Mobigis. Les objectifs à l'issue de cette formation sont de savoir concevoir et développer des applications informatiques en utilisant le langage Java/JavaEE, dans un environnement professionnel.

Dans ce mémoire sera présenté le travail que j'ai réalisé, avec par exemple les résultats directs de mon action. Mais aussi, les tâches que j'ai effectuées et les moyens utilisés pour les accomplir : langages, frameworks, logiciels,...

Cette présentation s'organise de la façon suivante :

- Une première partie sera dédiée à la présentation du contexte du stage. Dans cette partie seront présentés le domaine d'application de ce stage : les Systèmes d'Informations Géographiques (SIG), et l'entreprise MobiGIS.
- Dans la deuxième partie, l'environnement de travail du stage sera expliqué : outils, gestion de projets, difficultés rencontrées,...
- Une troisième partie présentera la méthodologie de mon travail (conception/développement) et des exemples de réalisations (codes)...
- Dans la dernière partie de ce mémoire seront présentés un bilan professionnel et un bilan personnel de cette nouvelle expérience.
- Pour conclure, la liste des compétences du référentiel qui sont couvertes par cette expérience sera présentée.

Par la suite, les termes en **gras** seront définis dans le glossaire en fin du mémoire.

# **Première partie**

# Présentation de Mobigis

---

## 2.1 Domaine d'application

Les Systèmes d'Informations Géographiques (SIG) sont des outils informatiques permettant de représenter et d'analyser toutes les choses qui existent sur terre ainsi que tous les événements qui s'y produisent. Les SIG offrent toutes les possibilités des bases de données (telles que requêtes et analyses statistiques) et ce, au travers d'une visualisation unique et d'analyse géographique propres aux cartes. Ces capacités spécifiques font du SIG un outil unique, accessible à un public très large et s'adressant à une très grande variété d'applications. Les enjeux majeurs auxquels nous avons à faire face aujourd'hui (environnement, démographie, santé publique...) ont tous un lien étroit avec la géographie. De nombreux autres domaines tels que la recherche et le développement de nouveaux marchés, l'étude d'impact d'une construction, l'organisation du territoire, la gestion de réseaux, le suivi en temps réel de véhicules, la protection civile... sont aussi directement concernés par la puissance des SIG pour créer des cartes, pour intégrer tout type d'information, pour mieux visualiser les différents scénarios, pour mieux présenter les idées et pour mieux appréhender l'étendue des solutions possibles. Les SIG sont utilisés par tous ; collectivités territoriales, secteur public, entreprise, écoles, administrations, états utilisent les SIG. La création de cartes et l'analyse géographique ne sont pas des procédés nouveaux, mais les SIG procurent une plus grande vitesse et proposent des outils sans cesse innovant dans l'analyse, la compréhension et la résolution des problèmes. L'avènement des SIG a également permis un accès à l'information à un public beaucoup plus large (ex : Google Maps). Aujourd'hui, les SIG représentent un marché de plusieurs milliards d'euros dans le monde et emploient plusieurs centaines de milliers de personnes. Le SIG appliqué aux transports, peut être utilisé pour gérer et analyser certaines informations essentielles :

- Planification et modélisation des transports
- Planification et analyse des itinéraires
- Localisation et suivi automatiques des véhicules
- Inventaire des arrêts de bus et des infrastructures, gestion des installations ferrées
- etc.

## 2.2 Présentation de l'entreprise

MobiGIS<sup>1</sup> se présente comme une entreprise au coeur de l'innovation, dont l'activité est d'apporter des réponses sur-mesure aux besoins de ses clients en matière de **géomatique**, notamment lorsqu'elle est appliquée aux enjeux de transport et de mobilité. MobiGIS est une société innovante éditrice de solutions dans le domaine des Systèmes d'Information Géographique (SIG). Elle intervient dans les thématiques : de l'environnement et du développement durable ; de la mobilité des personnes ; du transport et de la logistique (Fig. 2.1).

Ses équipes font de la conception, mise en œuvre et déploiement d'architectures SIG, développement d'applications de cartographie web et mobiles, études de transports, solutions de prévention des risques, etc. L'entreprise est compétente dans l'édition de logiciels SIG, le conseil et les services en SIG, la R&D liée aux NTIC<sup>2</sup>.



FIGURE 2.1 – Offres de services Mobigis

1. site de l'entreprise <http://www.mobigis.fr/>

2. exemples de réalisations <http://www.mobigis.fr/realisations/>

## 2.3 Activités

L'activité de MobiGIS est centrée sur les services SIG et l'édition de logiciels. Les clients de MobiGIS sont par exemples des industries, la grande distribution, les collectivités locales, et les intégrateurs et société de services en ingénierie informatique (SSII) (Fig. 2.2). L'entreprise développe en particulier une activité d'édition de logiciels. Ceux-ci permettent de faire des analyses multiples : territoire, pollutions, démographie, etc., d'élaborer des plans de déplacement urbain et entreprise, d'étudier et de préparer la réorganisation des réseaux multimodaux et la création de nouvelles infrastructures de transport. Elle compte parmi ses clients des Autorités Organisatrices de Transports (ex : Tisseo), des bureaux d'études, des sociétés de consulting immobilier, des agences d'urbanisme, etc.



FIGURE 2.2 – Exemples de clients Mobigis

## 2.4 Historique

- 2007 - Création de la société MobiGIS par Frédéric SCHETTINI
  - Le projet de création de société a obtenu le titre de « projet de création d'entreprises innovantes » par le pôle innovation de la Chambre de Commerce et d'Industrie de Toulouse
  - La société MobiGIS est accompagnée par Oséo Midi-Pyrénées
  - MobiGIS obtient le statut de Jeune Entreprise Innovante (JEI)

- 2008 - Mise en place d'une démarche active de développement des activités de MobiGIS en Chine
  - Obtention du Prix TOTAL de l'Innovation IT
  - Phase intensive de Recherche Développement
  - Début de collaboration avec le CNRS/LAAS
  - Labellisation du projet POTIMART par la PREDIM
- 2009 - Emménagement sur le site de Grenade-sur-Garonne (31)
  - MobiGIS intègre le groupement CECILE, composé de PME Toulousaines spécialisées dans la géolocalisation
  - MobiGIS rejoint l'association JEInnov
- 2010 - MobiGIS accompagne le ministre des transports en Chine
  - Labellisation par la Plateforme de Recherche et d'Expérimentation pour le Développement de l'Information Multimodale (PREDIM) du projet CAMERA
  - MobiGIS recrute un Volontaire International en Entreprise (VIE) pour intensifier son développement en Chine
  - Soutien de la Région de Midi-Pyrénées pour un Contrat d'appui
- 2011 - Commercialisation du progiciel MobiAnalyst
  - Soutien de la CCIT et d'HGI Tech pour intensifier son développement commercial
  - Recrutement d'un chef de projets SI et d'un ingénieur commercial
- 2012 - L'équipe MobiGIS s'étoffe et compte désormais plus de 10 collaborateurs
  - Prix de l'innovation au Toulouse Space Show et lauréat du concours Open Data « Défi numérique Toulouse Métropole »
  - Ouverture d'un bureau à Paris
  - Adhésion à l'Aerospace Valley
  - Participation à 10 congrès dont l'ITS World à Vienne
- 2013 - Signature de nouveaux contrats avec le groupe Total, Carrefour China, et l'APEM
  - Lancement de la solution Anvio et de la V2.3 de MobiAnalyst

- 2014 - MobiAnalyst© remporte le prix de meilleure application de l'année !
  - Ouverture d'un bureau au Canada pour intensifier l'activité en Amérique du Nord
  - Premier Workshop MobiGIS organisé en septembre 2014

## 2.5 Ressources de l'entreprise

Les compétences humaines permettent à l'entreprise de mener à bien des projets SIG, l'implémentation d'architecture logicielle, de systèmes de gestion de bases de données spatiales, et enfin de développer des solutions bureautique, serveur, web et mobile. L'entreprise héberge également les compétences nécessaires pour mener à bien des missions de conseil, d'audit et de consulting. Elle peut établir des états des lieux, des analyses et des préconisations. Enfin, MobiGIS propose des formations sur ses progiciels sur site ou à distance.

Les technologies maîtrisées par MobiGIS sont les logiciels SIG propriétaires, les SIG libres, et les langages de programmation logiciel, web et mobile. Parmi les logiciels SIG propriétaires il y a en premier lieu ESRI ArcGIS<sup>3</sup>, mais aussi MapInfo, GeoConcept et Google Maps sont également présents. Actuellement, l'entreprise s'inscrit dans la tendance de nombreux éditeurs, de ne plus seulement proposer des solutions desktop (qui imposent d'installer des logiciels sur son poste de travail, etc...) mais aussi de proposer des solutions à distance, plus souples (et moins onéreuses pour l'acheteur), notamment sur le modèle d'ArcGIS Online<sup>4</sup>. Les SIG libres sont aussi très présents dans les projets dont notamment les logiciels QGIS, PostGIS, OpenLayers, ou encore GeoServer.

Les langages "objets" maîtrisés par les développeurs et géomaticiens de l'entreprise sont divers dont notamment Java, C++, C#, Python... Ils permettent de programmer les progiciels de l'entreprise. L'équipe MobiGIS pratique également les langages web du moment : HTML 5, JavaScript, PHP, CSS 3, etc. Ils développent sur les principaux supports mobiles : IOS et Android, qui prennent aujourd'hui une importance croissante dans le monde des SIG en raison de l'évolution des pratiques liées à l'utilisation des smartphones et des tablettes.

---

3. <http://www.esrifrance.fr/arcgis.aspx>

4. [http://www.esrifrance.fr/ArcGIS\\_Online\\_1.aspx](http://www.esrifrance.fr/ArcGIS_Online_1.aspx)



## **Deuxième partie**

## Contexte du stage

---

### 3.1 Environnement de travail

Durant le stage j'ai travaillé sur un pc dont le système d'exploitation est Windows 8.1 Professionnel (machine hôte). L'entreprise travaille avec de nombreuses machines virtuelles hébergées ou distantes afin de disposer d'environnement de tests, de développements, et de production. J'avais donc à ma disposition une machine virtuelle de développement via VirtualBox dont le système d'exploitation était Windows Server 2012 R2 Standard. Ma machine était pré-configurée avec tous les outils nécessaires pour développer, (le clone de cette machine virtuelle de développement a servi à d'autres stagiaires), ainsi l'administrateur système de MobiGIS maîtrise la configuration logicielle des machines et l'utilisateur perd moins de temps à la configuration.

### 3.2 Outils utilisés

Quotidiennement j'ai utilisé 2 environnements de développement intégré (IDE) : **Liclipse** (pour développer en langage Python) dans le projet DataWizard, et **Eclipse** version Mars (pour développer en langage Java/Java EE) pour les projets MobiSAAS et Crislab.

Dans les projets plusieurs **SGBD** sont manipulés : MongoDB (MobiSAAS), Oracle (Crislab), mais j'ai principalement travaillé avec **Postgresql** afin de gérer les données de mon module. De plus, Postgresql permet de gérer les données géographiques dans le projet « DataWizard » grâce à sa cartouche spatiale **Postgis**.

Les logiciels suivants ont été utilisés quotidiennement : Maven, DropWizard, Hibernate, Jackson, Postgresql, Pgadmin, **SoapUI**, Java JDK 1.7 et 1.8. Certains de ces outils seront décrits dans la suite de ce mémoire [4.1.3](#). J'ai également produit des schémas pour la documentation avec des logiciels de « modélisation » comme : ArgoUML, DBVisualizer, Enterprise Architect, yEd.

### 3.3 Gestion de projets

Dans les projets auxquels j'ai participé, l'entreprise utilise des outils de gestion : planning, suivi de bugs, outils de mutualisation, gestion de versions. Ainsi, j'ai utilisé l'outil Trello pour la gestion des tâches, Redmine pour la gestion des projets (cf. Annexe 8.1), et SVN pour la gestion des codes sources. L'entreprise met à disposition de ses salariés un intranet avec de nombreux outils collaboratifs sous la plateforme eGroupware (feuille de temps, etc...).

J'ai effectué mon travail en étroite relation avec le chef de projet. A partir des spécifications techniques et fonctionnelles, les développements ont progressé et évolué tout le long de la période de réalisation. Sans pour autant pratiquer une méthode «Agile» au sens strict, plusieurs ajustements (mode itératif) ont été effectués et le dialogue quotidien a permis de toujours rendre mon travail en adéquation aux besoins.

Un point de suivi informel était effectué plusieurs fois par semaine avec mon responsable afin de présenter le travail effectué, les résultats intermédiaires, et le travail planifié pour la semaine suivante. Un bilan à la mi-stage a été effectué afin de réajuster les priorités, et arriver à produire un livrable satisfaisant en fin de stage.

### 3.4 Difficultés rencontrées

L'environnement "équipe" et les compétences de chacun ont été propices à les éviter. En effet, j'avais de chaque côté de mon bureau les 2 personnes les plus à même de débloquent des situations, ou de me renseigner sur une question.

Pour le projet MobiSAAS, j'ai eu des "difficultés" qui sont communes à chaque évolution de version d'un **framework**. En effet, au milieu du stage il a fallu "recoder" dans plusieurs parties du code pour s'adapter aux évolutions lors du passage de la version 0.7.1 à la version 0.8.1 de Dropwizard. Pour le projet DataWizard et d'une manière générale, une des difficultés a été de répondre aux experts et analystes en réseaux de transport et de produire des résultats (Métadonnées) exploitables et conformes à leurs attentes. Cela a demandé du dialogue pour s'accorder sur la terminologie à employer (ex : EdgeType, PTMode,...). Tous ces termes courants pour les experts ne sont pas intuitifs pour les développeurs.

## **Troisième partie**

# Les Projets

## 4.1 MobiSAAS

### 4.1.1 Cahier des charges

#### Présentation des besoins

Le projet « MobiSAAS » : MobiAnalyst as a Service, s'inscrit dans la démarche d'entreprise de proposer des solutions en mode **SAAS**. Le projet consiste d'une manière générale à exposer les fonctionnalités de la solution Desktop du produit « MobiAnalyst<sup>1</sup> ». C'est pour ce projet que j'ai effectué mon travail de stage.

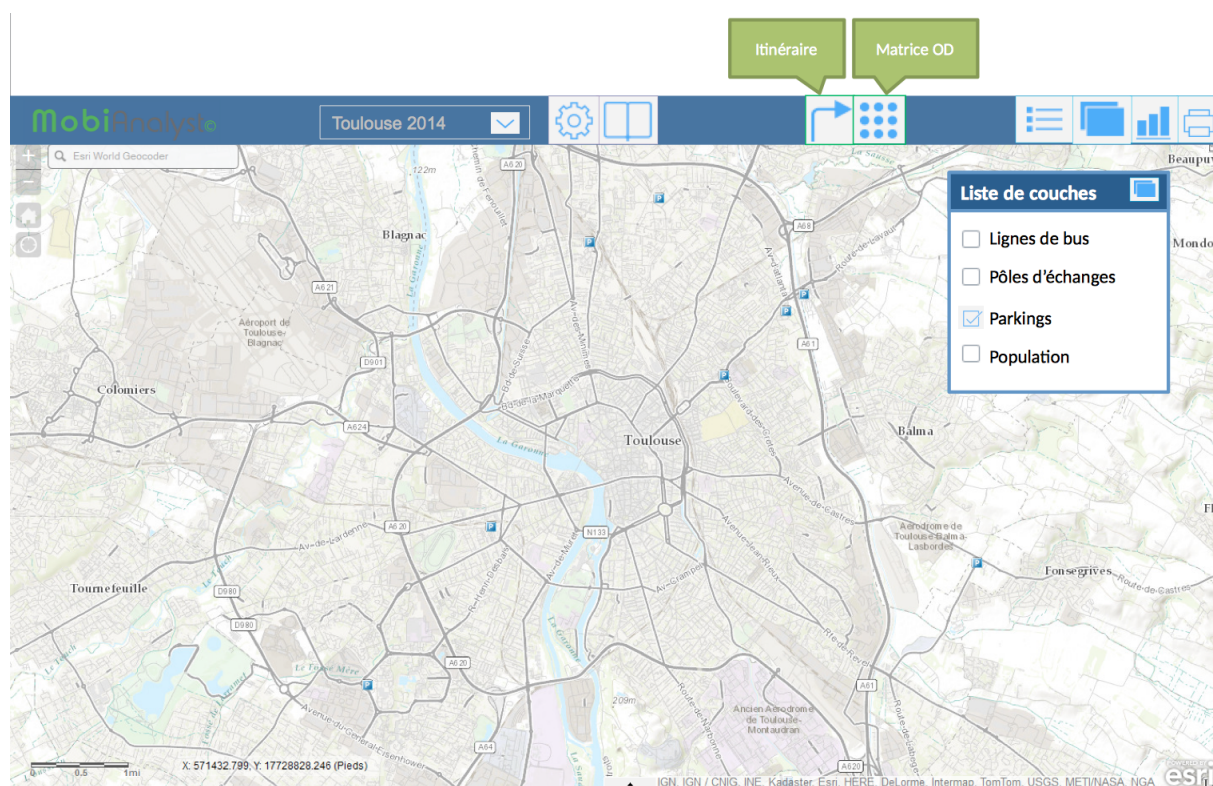


FIGURE 4.1 – Interface de MobiSAAS (frontend)

1. <http://www.mobianalyst.fr/>

Mon sujet de stage porte sur le développement de **web services** permettant d'exposer des fonctionnalités d'importation de données voirie et transport en commun, pour la construction automatique de réseaux de transports multi-modaux (graphes)(Fig. 4.2).

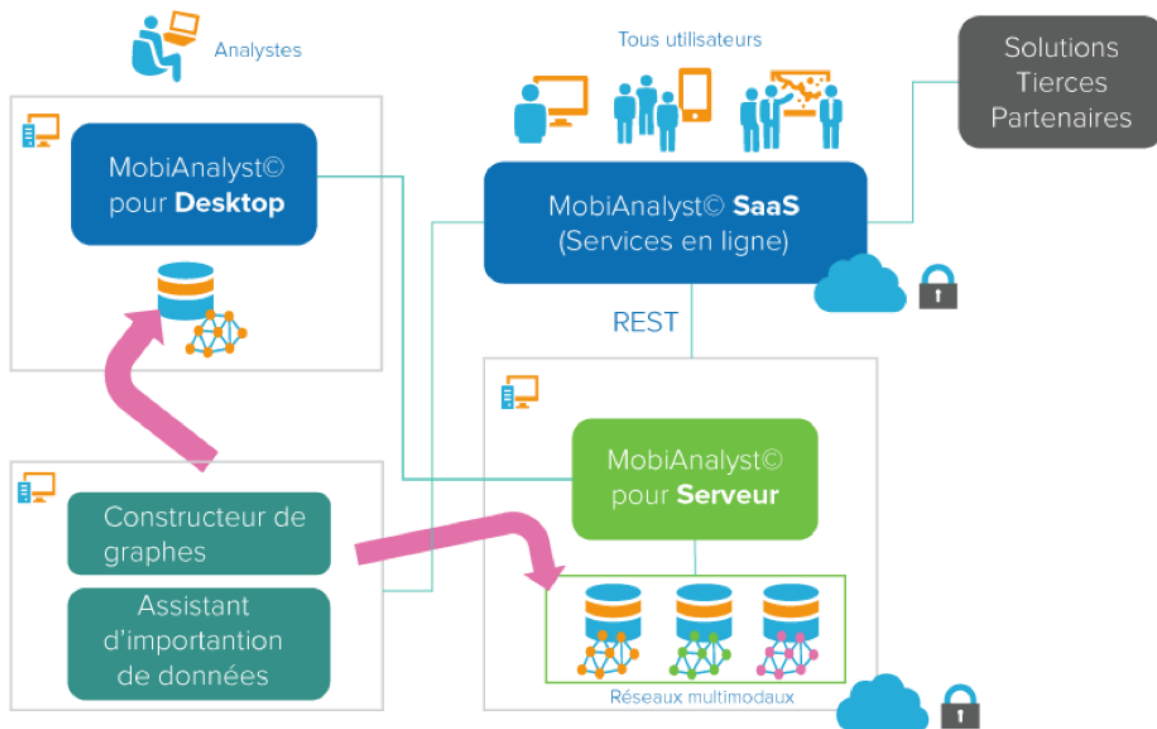


FIGURE 4.2 – Offres du produit MobiAnalyst

Les serveurs MobiAnalyst propose une **API** de services web de type **REST** avec des fonctions bas niveau :

- Calcul d'un itinéraire multi-modal
- Calcul d'un isochrone multi-modal
- Calcul d'une MOD
- Géotraitements d'analyse réseaux

C'est dans le cadre de l'interface d'administration de l'application (backend) que je devais développer des fonctionnalités d'upload de données de transport (GTFS<sup>2</sup>) sur le serveur.

L'objectif de mon travail a été d'exposer des fonctionnalités (en vert sur Fig. 4.2) codées initialement en langage SQL et/ou Python via une API REST en langage Java (JAX-RS).

2. <https://developers.google.com/transit/gtfs/>

J'ai donc dans un premier temps pris en main les chaînes de traitements (Python) et appris à manipuler les données (Postgis/SQL) via le projet « DataWizard ». En parallèle, afin d'implémenter ces web services j'ai intégré le projet « MobiSAAS » et ainsi appris à développer avec le framework « Dropwizard ».

### Une API REST ?

J'ai tout d'abord découvert ce qu'est une API REST et surtout comment la concevoir (maquetter). Je me suis inspiré de l'article suivant :

<http://blog.octo.com/designer-une-api-rest/> dont voici les grands principes à retenir :

- REST est un style d'architecture orienté ressource (ROA).
- L'abstraction clé de l'information en REST est une « Ressource ». N'importe quelle information qui peut être nommée est une ressource : une image, un document, un service temporel (la météo d'aujourd'hui à Toulouse), une collection d'autres ressources, un objet physique (une personne) etc. En d'autres termes, n'importe quel concept qui peut être la cible d'une référence d'un lien hypertexte doit s'adapter dans la notion de ressource.
- L'API doit donc proposer des web services sur ces ressources qui répondent aux requêtes du protocole HTTP (GET, POST, PUT, DELETE).
- Le concept de représentation (XML/JSON) est important. Nous avons choisi de ne représenter les résultats de requêtes (Response) que via le format d'échange de données JSON (JavaScript Object Notation).
- Un dernier concept fondamental sont les réponses de ces services. Le WS peut renvoyer des données (JSON), et des codes HTTP ([https://fr.wikipedia.org/wiki/Liste\\_des\\_codes\\_HTTP](https://fr.wikipedia.org/wiki/Liste_des_codes_HTTP)). Il est préconisé d'utiliser les codes de retour HTTP (500, 404, 200), de manière appropriée, sachant qu'il existe un code pour chaque cas d'utilisation courant. Ces codes sont connus de tous.

### Architecture actuelle

La figure 4.3 détaille l'ensemble des briques logicielles nécessaires au fonctionnement de la plate-forme MobiSAAS :

- **AGS** : ArcGIS Server. Le serveur de Système d'Information Géographique (SIG) permettant de publier et d'administrer des services cartographiques (MapService). Dans notre cas, les MapServices déployés sont des réseaux routiers et de transport en commun (TC), accompagnés de tables horaires (TimeTable) contenant les horaires des TC.

- **SOE** : Server Object Extension. Nous utilisons le mécanisme d'extension « SOE » pour déployer sur un MapService un service spécifique à MobiSAAS. Un SOE est une fonctionnalité qui se déploie sur un MapService et qui expose en Web Service (REST ou SOAP) les solveurs (algorithme de résolution d'itinéraires) de MobiAnalyst.
- **MobiAdmin** : Serveur REST d'administration MobiSAAS. Ce serveur (basé sur Java) est le point d'entrée des utilisateurs des services REST de MobiAnalyst. Il redirige les requêtes métiers vers le bon SOE déployé, il trace ces requêtes et enrichit la base de données client de MobiAnalyst. L'authentification qui est faite dans les requêtes utilise les comptes d'AGS créés au préalable.
- **Postgres** : Système de gestion de base de données. Cette base contient les données clients de MobiAnalyst : profil, traces d'utilisation de MobiSAAS.
- **MongoDB** : Système de gestion de base de données. Cette base contient les logs (statistiques mesurées en temps réel) correspondant aux services REST de MobiSAAS.

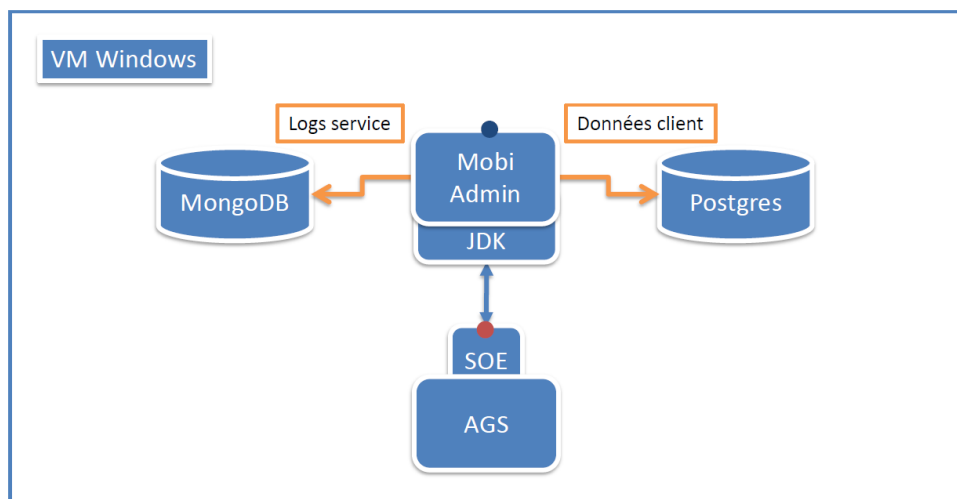


FIGURE 4.3 – Architecture globale de « MobiAdmin »

#### 4.1.2 Éléments de spécifications fonctionnelles

Mon stage et les développements demandés concernent le composant d'administration de l'infrastructure : « MobiAdmin » (cf. Fig. 4.3). Dans l'objectif de gérer les données du client, je dois proposer un web service pour la gestion des données de transport dans l'espace privatif du client. La fonctionnalité principale à développer est donc un web service d'upload de données et plus particulièrement l'upload de données de transport public au format GTFS<sup>3</sup>.

3. <https://developers.google.com/transit/gtfs/reference>



Les données GTFS se présentent le plus souvent dans une archive contenant au minimum les 8 fichiers .txt suivants :

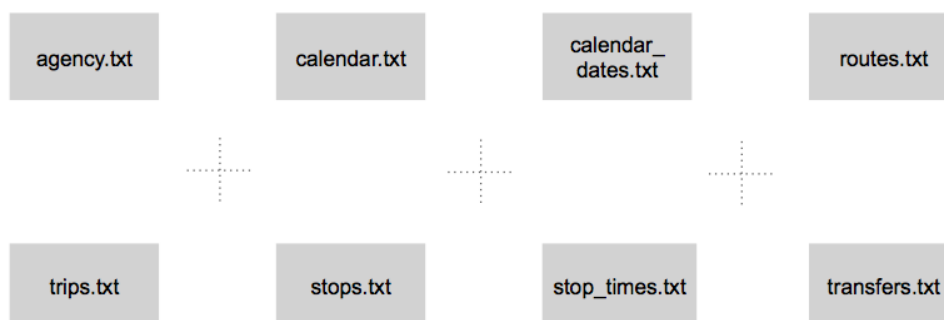


FIGURE 4.4 – Contenu des données GTFS

L'objectif de ce service d'import de données GTFS est de pouvoir manipuler ces données (fichier au format zip), et de récupérer des métadonnées. Par exemple : l'extension géographique des données, le nom de l'agence, le nombre de lignes, le mode de transport, etc. L'objectif final est de publier un réseau de transport en commun via un service en ligne. Actuellement, ces fonctionnalités sont réalisées en amont du logiciel MobiAnalyst via le logiciel Desktop Data-Wizard (cf. [4.2 DataWizard](#)).

Une archive ou « Upload » de données GTFS peut contenir un seul ou plusieurs jeu(x) de données (ensemble de fichiers .txt). Il faut donc récupérer et renseigner les métadonnées de chaque Upload sur le serveur :

- Nom - UUID (identifiant unique)
- Date d'upload
- Status d'upload (SUCCESS, FAILED, LOADING, INIT)
- Nom de l'archive (source)
- Chemin vers le rapport de validation GTFS
- Chemin vers la données

Les résultats attendus sont le stockage sur le serveur (système de fichier) des données envoyées par le client, la trace de l'opération dans un SGBD, et enfin la production de réponses HTTP (format JSON) pour chaque requête sur le service.

Les étapes successives du traitement seront donc : le téléchargement de l'archive sur le serveur, la validation des données, et enfin le chargement en base.

Pour les besoins de cette API nous avons spécifié des codes d'erreurs propres à l'application et au contexte : 601, 602, 603, respectivement Upload Failed, Validation Failed, Loading Failed.

Ce service dédié aux données GTFS est très spécifique. Une attention particulière a été faite afin de développer de manière à abstraire et rendre au maximum générique la plupart des composants afin de proposer d'autres format de données à l'importation sur le serveur REST « MobiAdmin » (utilisation d'interfaces et de classes abstraites pour la mise en place de l'héritage).

### 4.1.3 Éléments de spécifications techniques

Pour le composant « MobiAdmin » (serveur REST) qui va héberger les services, le choix technologique majeur est d'utiliser le framework DropWizard (cf. 4.1.3). Ce framework orienté microservices nous permet de fournir notamment un serveur embarqué HTTP « Jetty ». Et le framework open source Jersey pour la partie webservice REST qui est l'implémentation de référence de la spécification JAX-RS. Ou encore la librairie Jackson « King of JSON » pour la sérialisation/désérialisation du JSON.

L'environnement de développement est donc un projet Java EE Maven (cf. 4.1.3) composé des éléments de Dropwizard. Les **WS** exposés sont à priori « lourds », sachant qu'un jeu de données peut faire jusqu'à plusieurs centaines de Mo, les opérations de téléchargement, validation, traitement, stockage peuvent être long à répondre au client après chaque requête. Les WS à développer sont donc asynchrones. Chaque requête renvoie une réponse « immédiatement » pendant que le processus (thread) continu de s'effectuer afin de produire un résultat. A l'aide d'une Map (UploadMap), chaque requête GET (by ID) renvoie le status de l'opération : « en cours », ou « ok ». Si l'opération ne se déroule pas complètement une exception est levée et renvoie un des 3 codes d'erreurs. Le client peut par exemple effectuer des requêtes simultanément (GET) afin de demander le status de son upload. De plus, une contrainte supplémentaire est que le service doit supporter plusieurs requêtes simultanées, il faudra donc s'orienter vers un développement en mode « programmation concurrente ». Pour cela, on utilise le framework « executor » avec notamment l'interface « Callable » et un objet « newFixedThreadPool » (cf. 4.1.4).

## Technologies utilisées

### Apache Maven :



C'est un outil pour la gestion et l'automatisation de production des projets logiciels Java en général et Java EE en particulier. L'objectif recherché est comparable au système Make sous Unix : produire un logiciel à partir de ses sources, en optimisant les tâches réalisées à cette fin et en garantissant le bon ordre de fabrication.

Il est semblable à l'outil Ant, mais fournit des moyens de configuration plus simples, eux aussi basés sur le format XML. Maven est géré par l'organisation Apache Software Foundation. Précédemment Maven était une branche de l'organisation Jakarta Project.

Maven utilise un paradigme connu sous le nom de Project Object Model (**POM**) afin de décrire un projet logiciel, ses dépendances avec des modules externes et l'ordre à suivre pour sa production. Il est livré avec un grand nombre de tâches pré-définies, comme la compilation de code Java ou encore sa modularisation.

Un élément clé et relativement spécifique de Maven est son aptitude à fonctionner en réseau. Une des motivations historiques de cet outil est de fournir un moyen de synchroniser des projets indépendants : publication standardisée d'information, distribution automatique de modules « jar ». Ainsi en version de base, Maven peut dynamiquement télécharger du matériel sur des dépôts logiciels connus. Il propose ainsi la synchronisation transparente de modules nécessaires.

Dans le projet MobiSAAS, Maven est très utilisé car c'est un projet multi-modules avec un module parent. Maven me permet donc de télécharger et de gérer les dépendances du projet. Par exemple, voici un extrait du fichier **POM** du module utilitaire « mobi-admin-util » que j'ai développé (Fig. 4.5).

```
<artifactId>com-mobigis-util</artifactId>

<properties>
  <onebusaway.version>1.1.4-SNAPSHOT</onebusaway.version>
  <slf4j.version>1.5.5</slf4j.version>
  <geotools.version>8.6</geotools.version>
  <jts.version>1.8</jts.version>
  <opengis.version>3.0.0</opengis.version>
  <postgres.version>9.1-901.jdbc4</postgres.version>
  <hibernate.version>4.1.7.Final</hibernate.version>
  <hibernate.validator.version>4.1.0.Final</hibernate.validator.version>
  <hibernate.annotation.version>3.2.0.Final</hibernate.annotation.version>
  <junit.version>4.8.1</junit.version>
</properties>
```

FIGURE 4.5 – Extrait du fichier POM, versions des dépendances utilisées

**Dropwizard :**

C'est un framework Java léger adapté au développement rapide de micro-services REST et ne nécessitant pas de serveur d'application comme environnement d'exécution. Cela dit, au delà du framework, c'est surtout un assemblage habile de composants spécialisés parmi les meilleurs de l'écosystème Java :

- **Jetty**, un serveur HTTP et un moteur de servlet
- **Jersey**, l'implémentation de référence de la spécification JAX-RS (web services REST)
- **Jackson**, une librairie de sérialisation/dé-sérialisation JSON
- **Hibernate Validator**, l'implémentation de référence de l'API Bean Validation (JSR 303)
- **SLF4J** et **Logback** pour la gestion des traces
- **Metrics** pour le monitoring
- **jDBI** pour l'interfaçage rapide à une base de données relationnelle. Cette librairie est de bien plus bas niveau que JPA ou Hibernate et présente peu d'abstraction ce qui rend sa prise en main aisée

Packagée sous la forme d'un jar autonome contenant toutes ses dépendances, l'unité de déploiement n'a pas besoin de serveur d'application pour être exécutée (le conteneur Jetty est embarqué dans le jar). Avec ses 10 Mo tout au plus (dépendances comprises) l'empreinte mémoire d'une application Dropwizard est donc incomparablement plus faible qu'un Web Service SOAP déployé dans un serveur d'application Java EE (jusqu'à plusieurs centaines de Mo). En conséquences, le temps de démarrage d'une application Dropwizard est de quelques secondes quand il faut parfois plusieurs minutes pour un serveur d'application.

On peut considérer ce projet comme une alternative crédible aux serveurs d'applications Java EE perçus comme lourds, compliqués et gourmands en ressources. Le champ des applications couvert par Dropwizard est en réalité plus vaste que celui des microservices (il est tout à fait possible de développer une IHM web) mais l'aisance avec laquelle on développe et déploie un service REST en fait une solution très adaptée à ce type d'usage (à l'instar de Spring Boot de Pivotal ou Spark avec lesquels il est en compétition).

La structure d'une application Dropwizard de type API RESTful selon les préconisations d'organisation de projet du framework Dropwizard (cf. Fig 4.6) est :

- `com.example.myapplication`:
  - `api`: Representations.
  - `cli`: Commands
  - `client`: Client implementation for your application
  - `core`: Domain implementation
  - `jdbi`: Database access classes
  - `health`: Health Checks
  - `resources`: Resources
  - `MyApplication`: The application class
  - `MyApplicationConfiguration`: configuration class

FIGURE 4.6 – Organisation conseillée d'un projet Dropwizard

A partir de cette organisation, nous avons structuré le module « MobiAdmin » comme le montre la figure ci-dessous (cf. Fig 4.7). Plusieurs packages spécifiques ou de configuration supplémentaires apparaissent (auth, dao, jsonable, thread,...).

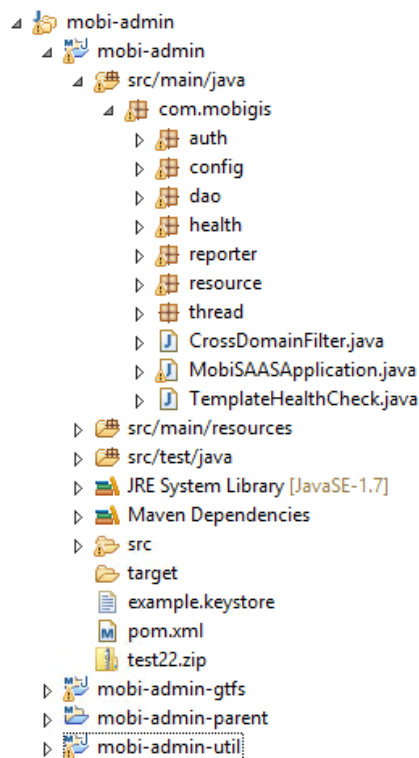


FIGURE 4.7 – Organisation du module mobi-admin

### Hibernate :



Dans les langages objet, les données étant le plus souvent stockées dans des bases de données relationnelles ainsi l'utilisation d'un framework de mapping Objet/Relationnel est recommandé pour assurer la rapidité, l'évolutivité et la maintenabilité des développements. Hibernate, issu de la communauté Open Source, répond à ce besoin et connaît depuis quelques années un vif succès. Ce

succès s'explique notamment par son architecture parfaitement adaptable à tout type de développements et le support de la majorité des bases de données du marché.

Afin de développer les classes pour la gestion des données clients (Uploads) du module «mobi-admin». J'ai mis en place une table « mobiuploads » sur le SGBD PostgreSQL (classes Upload.java, UploadDAO.java). Grâce au framework Dropwizard, nous avons utilisé Hibernate comme ORM (Object Relationnal Mapping). J'ai pu ainsi manipuler mes objets (opérations CRUD) pour la création, la lecture, la mise à jour et la suppression des objets dans la base. Egalement, j'ai développé le chargement de ces données métiers (GTFS) dans le SGBD, j'ai mis en place le modèle de données et développer les classes avec les annotations Hibernate (@Entity, @Table, @JoinColumns,...). Ainsi, si le schéma n'existe pas il se génère automatiquement grâce à Hibernate. Toutes les requêtes sont écrites dans les classes java en HQL, langage propre à Hibernate grâce aux annotations @NamedQueries.

### Jackson :

Jackson est une API JSON, elle est simple, bien documentée, et elle répond aux besoins suivants tels que :

```

/><FASTER
/><FASTER
><FASTER:
<FASTER:X
FASTER:XM
ASTER:XML
STER:XML
TER:XML /
ER:XML />

```

- Capable de sérialiser et désérialiser des arbres JSON sans adhérence aux beans modèles.
- Pouvant travailler directement sur des flux.
- Capable de tenir une charge conséquente, donc stable et performante.
- Avec le minimum possible de dépendances.

Afin que le service puisse renvoyer des réponses convenables et standards nous avons utilisé Jackson via Dropwizard<sup>4</sup>. Grâce à la librairie Jackson nous pouvons transformer les « beans » et convertir les propriétés java des objets en éléments Json, ainsi on renvoie par exemple le nom de l'upload, la date d'upload et surtout son status si l'opération c'est bien déroulée ou non (Fig. 4.8).

De la même manière, chaque donnée GTFS, passe par un traitement métier de validation. Chaque Upload, possède donc une propriété appelée « reportValidationPath » qui indique le chemin vers ce rapport de validation au format json. Ici, Jackson va permettre la production du rapport contenant les métadonnées et les anomalies pour chaque jeu de données GTFS (Fig. 4.9).

4. <http://www.mkyong.com/java/how-to-convert-java-object-to-from-json-jackson/>



FIGURE 4.8 – Réponse Json renvoyée

### SLF4J et Logback :



SLF4J est une couche d'abstraction pour les API de journalisation Java. Le principe est à peu près similaire à celui de Jakarta Commons Logging. Les avantages de l'utilisation d'une telle couche d'abstraction permettent de s'abstraire de l'implémentation utilisée. Ainsi, il est possible de changer facilement d'implémentation de journalisation sans avoir à toucher la base de code. Au plus, la configuration de l'implémentation de journalisation doit être modifiée. Et enfin, dans le cas de la conception d'une librairie,

cela permet de laisser à l'utilisateur de cette librairie le choix du système de journalisation.

Logback est un framework de logging. Logback est l'implémentation native de SLF4J, alors que son implémentation pour Log4J (ancêtre de Logback) est « wrappée ». De ce fait, il offre des fonctionnalités supplémentaires à Log4J.

Pour la gestion des traces (logs), j'ai donc pour chaque partie du code documenter et produit une trace intelligible à plusieurs niveaux d'informations : INFO, DEBUG, ERROR, etc... Encore une nouvelle fois, via Dropwizard l'utilisation d'un logger est facilité (cf. Extrait du log dans Annexe 8.2).

### JUnit :



JUnit est un framework de test unitaire pour le langage Java, il s'intègre à l'IDE Eclipse. JUnit définit deux types de fichiers de tests. Les TestCases sont des classes contenant un certain nombre de méthodes de tests. Un TestCase sert généralement à tester le bon fonctionnement d'une classe. Une TestSuite permet d'exécuter un certain nombre de TestCase déjà définis. Pour notre cas, nous n'avons utilisé que des TestCase, pour tester de manière unitaire des fonctionnalités. Cet outil a été complémentaire des tests réalisés avec SoapUI.

JUnit est un framework de test unitaire pour le langage Java, il s'intègre à l'IDE Eclipse. JUnit définit deux types de fichiers de tests. Les TestCases sont des classes contenant un certain nombre de méthodes de tests. Un TestCase sert généralement à tester le bon fonctionnement d'une classe. Une TestSuite permet d'exécuter un certain nombre de TestCase déjà définis. Pour notre cas, nous n'avons utilisé que des TestCase, pour tester de manière unitaire des fonctionnalités. Cet outil a été complémentaire des tests réalisés avec SoapUI.

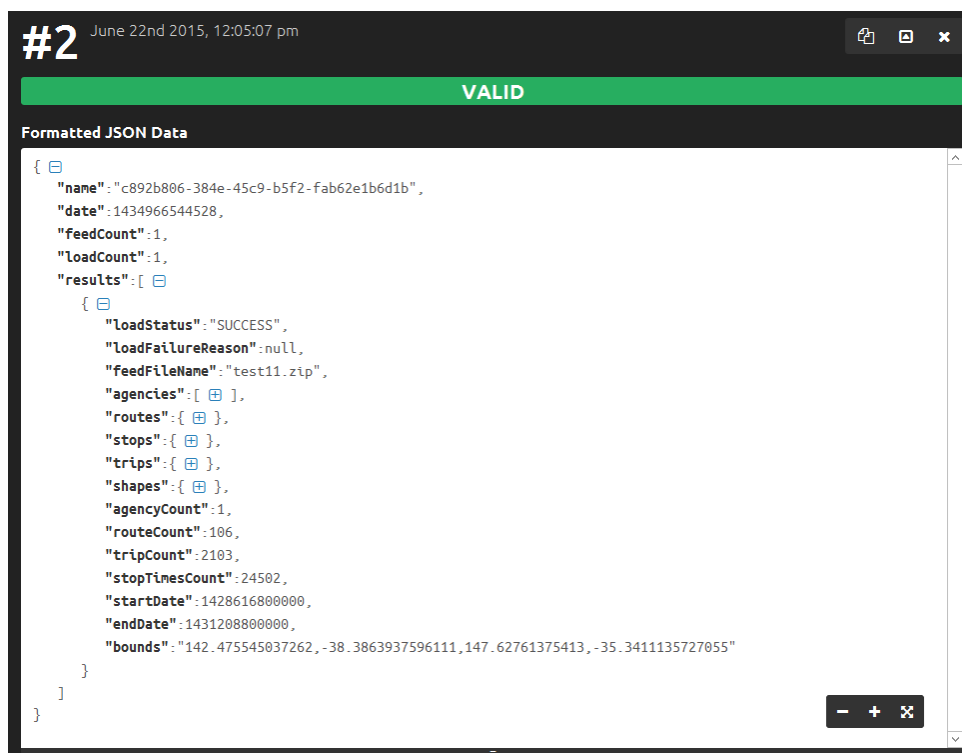


FIGURE 4.9 – Rapport Json de validation des données GTFS

#### 4.1.4 Réalisations

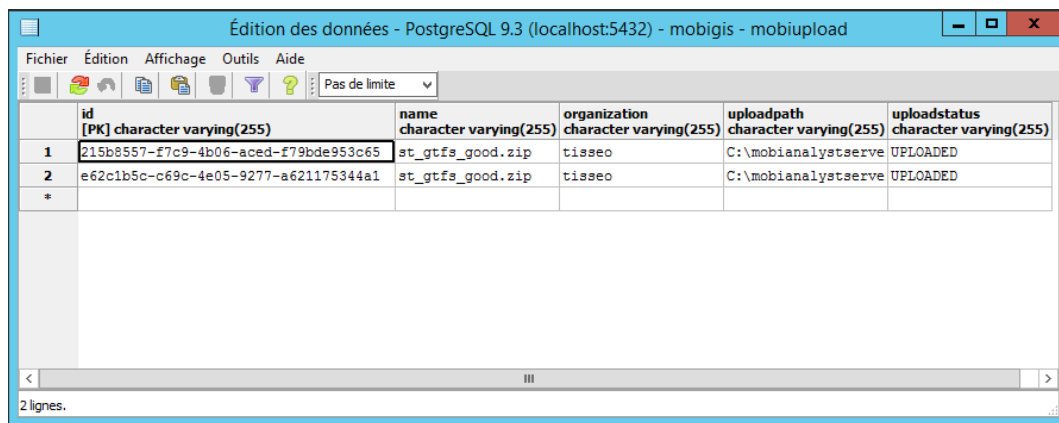
J'ai tout d'abord commencé par découvrir Maven (cf. 4.1.3), les projets, les modules, le désormais célèbre fichier **POM**, etc... Ensuite, je me suis documenté sur le code métier existant, et chercher des outils ou bibliothèques de professionnels du domaine (cf. Outils métiers 8.6). Enfin, après le « Getting Started » de Dropwizard<sup>5</sup>, une fois tout cela bien maîtrisé, j'ai pu commencé la conception et le développement de services web REST.

J'ai choisit de stocker les informations concernant ma partie de l'application dans une table PostgreSQL (Fig 4.10)

A chaque requête POST du client, je crée une instance d'objet Upload, les informations sont saisies dans la table de suivi, et je stocke les données envoyées sur le serveur. Pour chaque client ou utilisateur, pour chaque type de données, il y a un répertoire personnalisé, chaque upload est « taggé » d'un identifiant unique « UUID ». Le code métier qui a été implémenté est pour le moment le test sur le type de données envoyées, la validation des données et la production d'un rapport d'un validation (JSON), enfin l'extraction (récursive) des données contenues dans l'archive (cf. Annexe 8.2).

5. <https://dropwizard.github.io/dropwizard/getting-started.html>





	id [PK] character varying(255)	name character varying(255)	organization character varying(255)	uploadpath character varying(255)	uploadstatus character varying(255)
1	215b8557-f7c9-4b06-aced-f79bde953c65	st_gtfs_good.zip	tisseo	C:\mobianalystserve	UPLOADED
2	e62c1b5c-c69c-4e05-9277-a621175344a1	st_gtfs_good.zip	tisseo	C:\mobianalystserve	UPLOADED
*					

2 lignes.

FIGURE 4.10 – Table des uploads sur le SGBD PostgreSQL

A partir de ce service spécifique, dédié aux données GTFS, j’ai généralisé cette ressource (FileResource.java) afin de permettre d’upload n’importe quel fichier sur le serveur, en réutilisant les mêmes méthodes, notamment celle de UploadDAO.java et la méthode « métier » manageUpload().

Dans un deuxième temps, j’ai entamé le « chantier » de permettre la persistance des données GTFS au sein de la base de données Postgres. A partir des classes du modèle (package gtfs) (Fig 4.11), et du schéma relationnel, j’ai produit à l’aide des annotations JPA le code nécessaire à la génération du schéma gtfs dans la base et le chargement des données (txt) (Fig 4.12).

## Application multi-couches :

### Domaine métier

Les classes du domaine métier sont des POJOs (Plain Old Java Object) et sont persistés par les objets d’accès aux données (DAO : Data Access Object). L’acronyme POJO est principalement utilisé pour faire référence à la simplicité d’utilisation d’un objet Java en comparaison avec la lourdeur d’utilisation d’un composant EJB. Ainsi, un POJO n’implémente pas d’interface spécifique à un framework comme c’est le cas par exemple pour un composant EJB.

Voici l’exemple d’un objet (Upload.java) persisté par Hibernate :

Upload

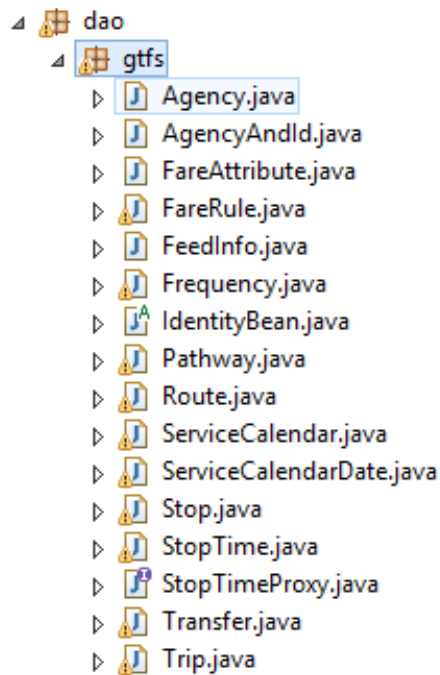


FIGURE 4.11 – Classes des objets GTFS (couche modèle)

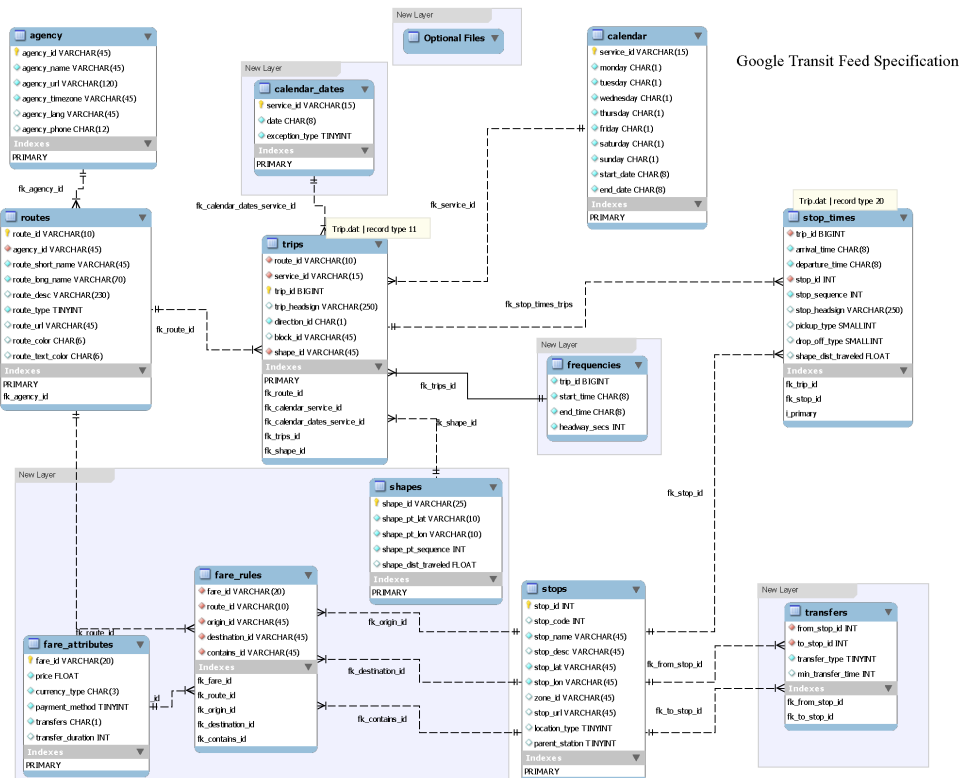


FIGURE 4.12 – Modèle relationnel des données GTFS (couche modèle)

Les annotations ne sont pas obligatoires et dépendent du framework de persistance utilisé. Dropwizard n'apporte pas de restriction sur le choix de la librairie utilisée et apporte par défaut JDBI (module dropwizard-jdbi) et Hibernate (module dropwizard-hibernate).

## DAO

La DAO UploadDAO étend la classe abstraite *AbstractDAO* de Dropwizard pour avoir accès à Hibernate au travers de ces méthodes.

Voici l'exemple d'une classe DAO :  
UploadDAO

## Service REST

La classe du service REST est autonome : il n'y a pas d'adhérence à Dropwizard. Elle utilise directement les annotations de Jersey : @Path, @Produces, @GET et quelques annotations de Dropwizard : @UnitOfWork, @LongParam...

Voici l'exemple d'une classe DAO :  
GtfsResource

La DAO UploadDAO est injectée via le constructeur. Cela a été effectuée dans la méthode run de la classe principale MobiSAASApplication de l'application Mobi-Admin via la ligne :  
environment.jersey().register(new GtfsResource(dao));

## Programmation concurrente

Un processus représente l'environnement d'exécution d'un programme. Il référence d'une part un espace mémoire permettant de stocker les données propres à l'application, et d'autre part un ensemble de threads permettant l'exécution du code qui manipulera ces données.

En Java, au démarrage de l'application, un thread initial est créé : le thread « main ». Son rôle est de localiser le point d'entrée de l'application (la méthode public static void main(String... args)) puis d'exécuter son code. Ce thread, comme tous les threads, exécute la séquence d'instructions qui lui est confiée de manière purement séquentielle. Si une instruction prend du temps

à compléter (par exemple, en attente de connexion à un serveur), toute l'application est paralysée. Pour éviter cela, il est souhaitable de confier l'exécution de ces portions bloquantes à des threads annexes, laissant ainsi le thread principal libre de continuer l'exécution de l'application.

### **Le framework « Executor »**

Disponible dans le package `java.util.concurrent`, il fournit un pool de threads robuste et hautement configurable, ainsi que les classes `Callable` et `Future` qui étendent les fonctionnalités des `Runnable`s (approche plus classique ?). Sa mise en œuvre doit systématiquement être préférée à la création manuelle de threads. Ce framework répond aux problématiques suivantes : 1> d'une part, chaque thread supplémentaire augmente la mémoire consommée, la complexité globale de l'application, et le risque de contention ; 2> d'autre part, pour de petites tâches, le coût de création d'un nouveau thread peut se révéler supérieur au coût d'exécution du traitement associé.

La méthode `newFixedThreadPool()` du service `Executor` permet de précharger en cache une réserve (pool) de threads et limite à un nombre fixe le nombre de requête simultanée (pour notre service le maximum est mis à 10). Ainsi, jusqu'à 10 requêtes, le service répondra et les traitements seront exécutés en parallèle, au-delà, la requête sera mise dans la file d'attente.

Afin de renvoyer un résultat et de savoir si le traitement lève une exception, le framework « Executor » propose l'interface `Callable<V>`, qui est une sorte de `Runnable` amélioré. Le type paramétré `<V>` définit le type du résultat produit par la méthode `call()`. Ainsi, un `Callable<String>` produira un `String`. (Fig. 4.14)

Une fois soumis à un pool de threads, un `Callable` est généralement exécuté de manière asynchrone ; le résultat produit ne sera sans doute pas disponible avant un certain temps. Du point de vue de l'appelant (celui qui soumet le traitement au pool), cela n'aurait aucun sens d'attendre ce résultat de manière synchrone : il perdrait tout le bénéfice du système ! Mais il lui faut tout de même un moyen de récupérer le résultat lorsqu'il aura été calculé.

Le framework « Executor » fournit donc la classe `Future<V>`, qui représente un résultat de type `V` dont la valeur est pour l'instant inconnue (puisque le traitement n'a pas encore été exécuté), mais qui sera disponible dans le futur. La méthode `get()` permet de récupérer le résultat immédiatement s'il est disponible (c'est-à-dire si le traitement a bien été exécuté par le pool de threads). Mais attention : si son calcul n'est pas terminé, la méthode est bloquante ! C'est donc une bonne pratique de vérifier la réelle disponibilité du résultat avec `isDone()` avant de le récupérer.

**Tests** Afin de tester les web services REST développés, j’ai utilisé fréquemment l’outil **SoapUI** (Fig. 4.13). Il permet de mettre en place une suite de tests qui peuvent être lancée d’une traite du côté client, permet de tester les services web en mode bouchon mais aussi d’effectuer des tests de charge. Il permet entre autre de fournir une hiérarchie des services web, de lister les différentes méthodes disponibles, les paramètres attendus, de réaliser des requêtes et de récupérer les réponses ... C’est l’un des meilleurs outils de test unitaires concernant les services web.

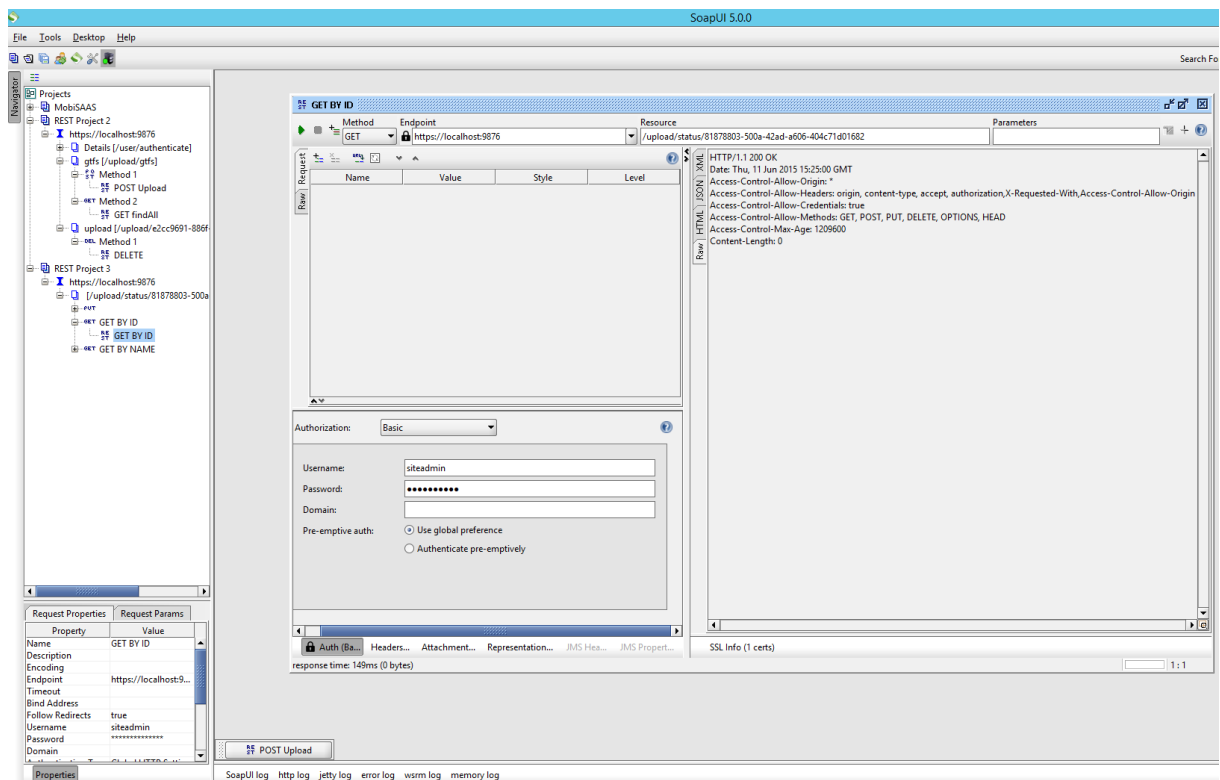


FIGURE 4.13 – Interface et exemple de requête « GET » SoapUI

### 4.1.5 Perspectives

L’idéal d’un programme développé dans un langage orienté objet est la généricité et la réutilisation d’un maximum de composants. Dans ce travail, l’objectif est clairement de produire un maximum de composants abstraits (classes et interfaces) et de méthodes réutilisables. Les perspectives du projet « MobiSAAS » sont nombreuses : gérer une architecture distribuée, augmenter les fonctionnalités SAAS notamment les fonctionnalités de « MobiAnalyst », étendre le projet « DataWizard » lui aussi en mode SAAS.

J'ai donc « maqueté l'application » et débuté le développement de ces composants abstraits (ex : Les ressources au sens DropWizard (Fig. 4.14). Grâce à cela on pourra donc « uploader » plusieurs types de données et réutiliser la méthode `manageUpload()`. Egalement, sur ce même principe j'ai développé une classe abstraite afin de faire proposer la gestion de l'asynchronisme utilisée dans tous les services que j'ai implémentés, limitant ainsi la duplication de codes dans l'application.

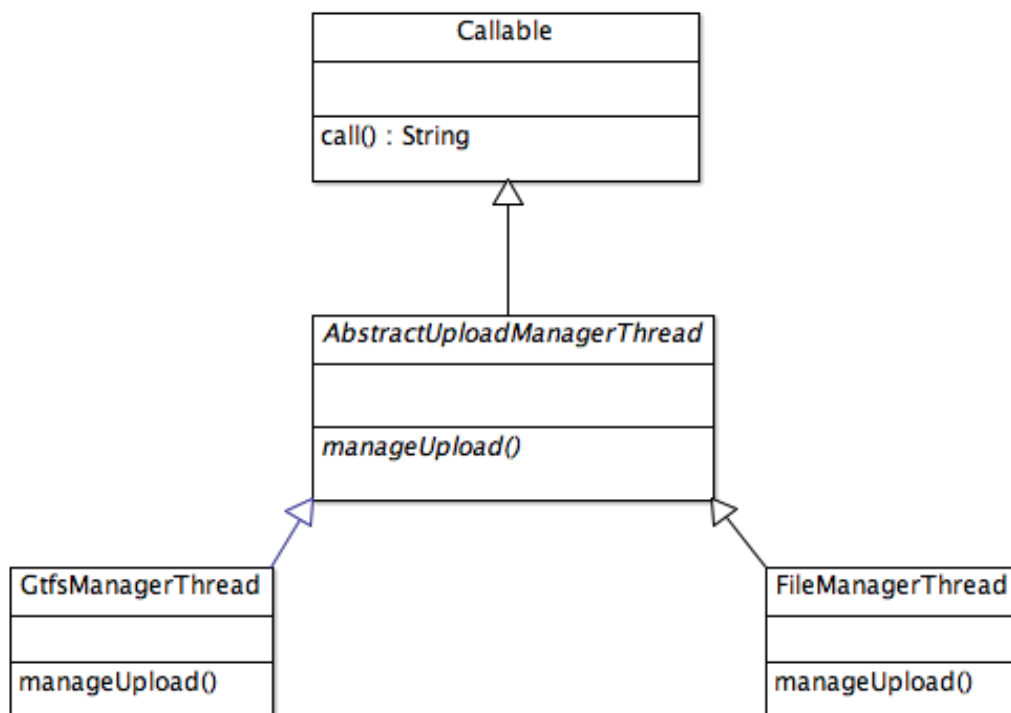


FIGURE 4.14 – Exemple d'abstraction et d'héritage de méthode

J'ai commencé un autre traitement de données spécifique au format GTFS. Une méthode de chargement des données dans la base de données Postgresql (méthode `loadData()`). J'utilise pour cela une librairie appelée OneBusAway<sup>6</sup> (OBA) spécialisée dans le transport en commun. Ce traitement réalise une désérialisation (lecture de fichiers csv) et une persistance des données via le framework Hibernate dans Postgresql. Ce qu'il reste à faire est de passer la méthode en service REST.

6. <http://onebusaway.org/developer-information/>

## 4.2 DataWizard

### 4.2.1 Cahier des charges

#### Présentation des besoins

Le projet « DataWizard », ensemble de scripts Python et SQL est une application dont le but est d'ingérer des données de transports, et de voirie dans une base de données spatiales ([PostGIS7](#)) afin de produire en sortie un réseau de transport (Network Dataset ou NDS).

Sur ce projet, les besoins correspondent à des corrections de bugs ainsi que des développements d'évolutions de fonctionnalités décrites sur le projet Redmine. Ainsi, j'ai réalisé de la production (nombreux réseaux), et j'ai fait évoluer l'outil notamment afin de lui permettre de produire des métadonnées.

### 4.2.2 Eléments de spécifications fonctionnelles

Le DataWizard permet de se lancer par étapes. Il y a 10 étapes successives, dont une étape préliminaire appelée étape 0, et une indépendante : l'étape 10.

- STEP 0 : Nettoyage et préparation de la base de données
- STEP 1 : Import des données transport en commun et conversion vers le schéma mobianalyst
- STEP 2 : Import des données de voirie
- STEP 3 : Import des données métro et conversion vers le schéma mobianalyst
- STEP 4 : Génération des données vitesses moyennes et horaires et connexion des réseaux de transport en commun et voirie
- STEP 5 : Calcul des horaires et des vitesses moyennes
- STEP 6 : Export des données vers des fichiers Shapefile (.shp)
- STEP 7 : Import des données dans la Géodatabase de sortie et changement du système de projection si nécessaire
- STEP 8 : Extraction des différents modes, génération du fichier xml servant à construire le réseau et de MobiNetwork.xml
- STEP 9 : Construction et compilation du réseau
- STEP 10 : Production de métadonnées pour le réseau

Afin de fournir, un réseau conforme aux attentes du logiciel MobiAnalyst « version 3.0 » et des experts en analyse de réseaux de transports, l'équipe me fournit une spécification de métadonnées à extraire des données en entrée du DataWizard (Fig. 4.15).

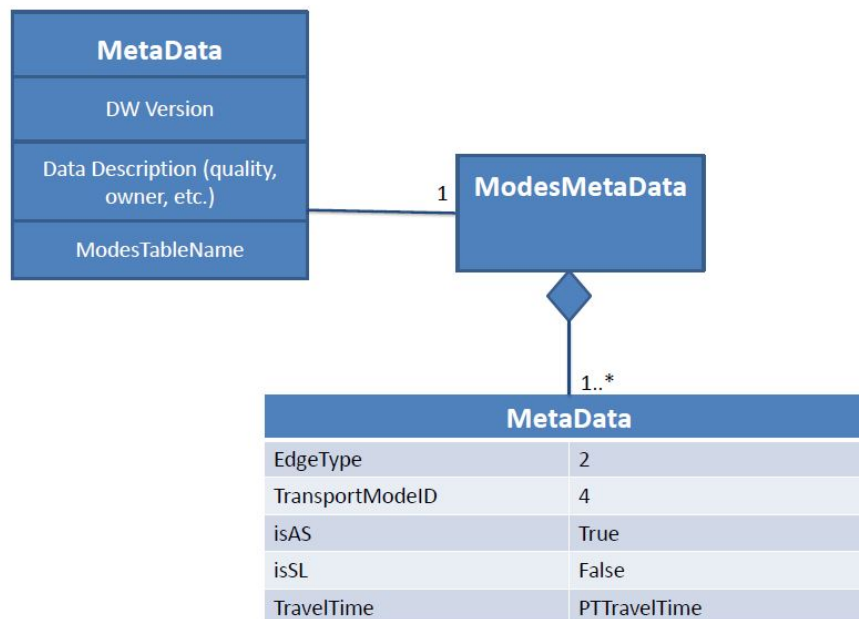


FIGURE 4.15 – Métadonnées de réseau de transports v3.0

### 4.2.3 Éléments de spécifications techniques

J'intègre tout d'abord le projet en tant qu'utilisateur, j'exploite des données et produits des réseaux de transport en commun (TC) (Bordeaux, Champagne-Ardennes, Montreal, Melbourne, etc.). Ensuite, petit à petit je suis le plan de charge issu du projet Redmine et corrige des bugs et des anomalies.

Mon premier travail a été de développer l'étape 10 (STEP 10 : Export Metadata Tables) de l'exécution du DataWizard. Pour cela j'ai dû extraire le nombre de modes de transport en commun (variable selon chaque projet), leurs codes (variable selon chaque projet) et enfin extraire des listes de constantes pour documenter le réseau produit par le DataWizard.

Un autre travail réalisé est le développement qui permet la gestion du système de projection des données géographiques pour les calculs du DataWizard. Cette modification impacte quasiment la totalité des requêtes SQL dites spatiales de l'application, et permet des résultats (calculs



de distances) plus réalistes (par exemple pour Melbourne, Australie située dans l'hémisphère Sud).

#### 4.2.4 Réalisations

Les principales difficultés concernent les données. Ces données GTFS ou voirie (données Here<sup>7</sup> (anciennement Navteq) ou OpenStreetMap<sup>8</sup>) sont complexes. La base de données « DataWizard » possède ainsi de nombreux schémas, et les requêtes SQL sont parfois très complexes mêlant fonctions, et requêtes géographiques.

Les résultats obtenus sont la production de nombreux réseaux avec leurs métadonnées pour nos analystes, et pour les projets de R&D nécessitant des réseaux récents (Moveazy, Mobilyse, etc...). J'ai ainsi amélioré la stabilité de l'application.

Exemple de code pour produire une table de métadonnées (Fig. 4.16) :

Dans ce projet, j'ai également développé des fonctions utilitaires : BOMConverter.py, searchAndReplaceFiles.py, etc... Ces méthodes permettent notamment de formater les données avant leur import dans la base de données. Cela peut certainement éviter les mauvaises surprises d'encodage ou de caractères spéciaux fréquemment rencontrés lors de l'exploitation de données GTFS.

#### 4.2.5 Perspectives

La liste des évolutions sur le projet Redmine est longue, il y a en effet énormément de perspectives pour le projet... Tout d'abord permettre l'import de plusieurs formats de données, comme l'import de données « Shapefile » qui n'est pas encore terminé, mais aussi de données de transports en commun « Trident », de données de trafic, etc. Augmenter encore la capacité de l'outil à rendre ses étapes indépendantes, nettoyer les fonctions obsolètes, revoir l'algorithme de certaines méthodes,... D'une manière générale les besoins pour l'application sont d'optimiser les étapes longues, et de développer des routines de simplification de données (SQL) afin par exemple de ne retenir que les données nécessaires au traitement au lieu de charger toutes les

---

7. <https://company.here.com/here/>

8. <http://openstreetmap.fr/>

```

from Exceptions import CommandError
import logging, sys, os, csv, datetime
from Config.settings import config
import Fonctions
import arcpy
from arcpy import env
import DataWizard
#####
##      This function aims to produces csv file for reference      ##
#####
def exportRefCodeGTFS(logger):

    STEP_10_ExportMetadataTables = "\n\n*****" \
        "\n** ETAPE : génération de tables de métadonnées **" \
        "\n*****\n"

    logger.info(STEP_10_ExportMetadataTables)

    #data = codes GTFS
    # https://developers.google.com/transit/gtfs/reference#routes_fields
    headings=['route_type','labelGTFS']
    RouteTypeGTFS = [{'route_type':0,'labelGTFS':'TRAMWAY'},
        {'route_type':1,'labelGTFS':'METRO'},
        {'route_type':2,'labelGTFS':'TRAIN'},
        {'route_type':3,'labelGTFS':'BUS'},
        {'route_type':4,'labelGTFS':'FERRY'},
        {'route_type':5,'labelGTFS':'TRAIN'},
        {'route_type':6,'labelGTFS':'GONDOLA'},
        {'route_type':7,'labelGTFS':'FUNICULAR'}]

    #On écrit les data dans le fichier
    with open(config['outputFolder'] + "\\\" + 'refCodesGTFS.csv', 'w') as f:
        writer = csv.DictWriter(f, fieldnames=headings, dialect='excel', quoting=csv.QUOTE_NONNUMERIC)
        writer.writeheader()
        for data in RouteTypeGTFS:
            writer.writerow(data)

    logger.info ("## Table de référence codes GTFS générée : " + str(datetime.datetime.now().time()) + " ##\n")

```

FIGURE 4.16 – Fonction Python pour exporter des métadonnées

données en amont, réalisé des tests de qualité sur les données d'entrée...

A long terme, l'idée est de proposer toutes les fonctionnalités du DataWizard via une application en mode SAAS.

## 4.3 Crislab

### 4.3.1 Présentation

Ce projet est dédié à la Cartographie des RISques de LABoratoire. Il consiste en une application web, proposant une interface cartographique des bâtiments afin de permettre la gestion des marqueurs et relevés de zones à risques. La vocation de l'application est un site intranet. Durant mon stage, j'ai eu de nombreux échanges avec le client afin de résoudre des bugs de l'application. J'ai donc participé au débogage et à la production de nouveaux livrables (.war).

Le projet est très complet, et présente une architecture assez typique d'une application web

(cf. Annexe 8.3). Ainsi, l'interface Homme-Machine (IHM) est codée en Javascript, il y a de nombreux formulaires d'affichage ou de saisie dans des fichiers JSP (Java Server Pages), la partie cartographique est encore une fois produite par un serveur ArcGIS dédié. Le serveur d'applications est un serveur Tomcat 6, adossé sur un SGBD Oracle 10g. Toute la persistance est gérée via le framework Hibernate.

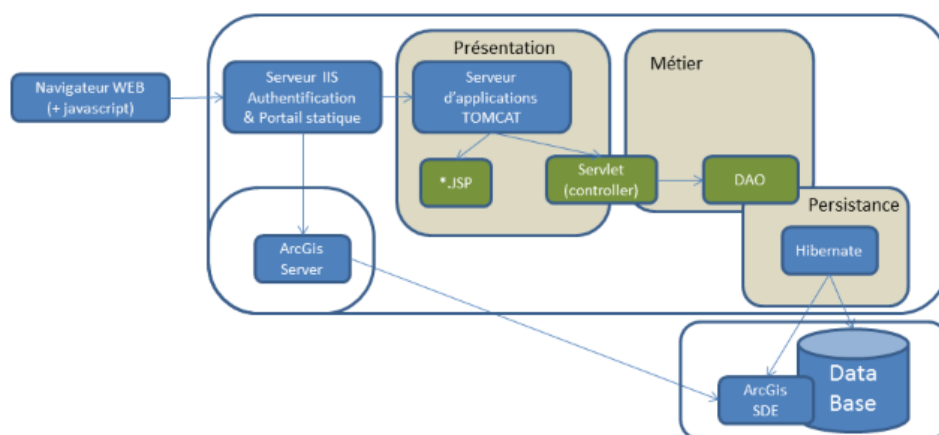


FIGURE 4.17 – Architecture globale de l'application Crislab

### 4.3.2 Réalisations

J'ai donc « administré » le projet sous Eclipse avec Maven, j'ai lancé l'application sur un serveur Tomcat, créer différents profils de configuration (développement, tests, production) avec des propriétés de connexion différentes à chaque profil). J'ai testé l'application avec le navigateur « client » Internet Explorer 11. Ce projet m'a permis de mettre en application mes connaissances client-serveur, sur toute la partie présentation d'une application web depuis l'affichage des JSP jusqu'aux actions déclenchées (couche service) interceptées par les contrôleurs.

Le debug a surtout consisté à reproduire les bugs de l'application en production. Du coup, j'ai dû importer le schéma de la base de données (Oracle) des clients, tester les requêtes de mon service sur leurs données. J'ai essentiellement passé mon temps à refaire les requêtes, et notamment traduire le HQL (langage de requête Hibernate) en SQL (langage de requête « standard »). Par exemple, le premier bug était dû à une mauvaise initialisation d'une variable. Un deuxième bug venait d'une mauvaise manipulation de l'opérateur qui avait provoqué la création d'un doublon dans la base (chose impossible à réaliser via l'interface cliente (IHM)).

## **Quatrième partie**

## **5.1 Bilan professionnel**

Grâce à ce stage au sein de l'entreprise MobiGIS, j'ai pu acquérir une expérience dans le développement de sites web dynamiques et utiliser la technologie Java EE. De plus, j'ai pu intégrer un projet innovant me permettant de découvrir énormément de choses : REST, Dropwizard, JSON, etc. Cette expérience est pour moi très valorisante tant d'un point de vue humain que technique, elle répond parfaitement à mes attentes.

De plus, la mise en pratique de mes connaissances théoriques en Java se sont parfaitement concrétisées au domaine d'application des Systèmes d'Information Géographique (SIG) dans lequel j'exerçais auparavant en tant que Géomaticien.

## **5.2 Bilan personnel**

Ces 3 mois de stage ont été très enrichissants car ils m'ont permis de découvrir le monde de l'industrie informatique et d'appliquer mes compétences passées de Géomaticien. En effet, après 7 années d'expériences dans le domaine de la recherche publique, j'ai pu intégré une équipe dynamique, des projets concrets, dans une entreprise privée. J'ai découvert un nouveau domaine en plein développement et propice à l'innovation, celui des « Transports ».

Enfin, ce stage a été pour moi l'occasion de pratiquer des langages tels que Python ou Java, langages largement répandus dans le monde industriel. J'ai aussi réalisé ce mémoire avec le langage  $\text{\LaTeX}$ .

# Conclusion

---

Pour conclure ce rapport, et d'un point de vue "académique" les 12 compétences suivantes peuvent être mises en avant suite à cette expérience :

- Pour l'activité "Développer des composants d'interface" :
  - Maquetter une application
  - Développer des composants d'accès aux données
  
- Pour l'activité "Développer la persistance des données" :
  - Concevoir une base de données
  - Mettre en place une base de données
  - Développer des composants dans le langage d'une base de données
  - Utiliser l'anglais dans son activité professionnelle en informatique
  
- Pour l'activité "Développer une application n-tiers" :
  - Concevoir une application
  - Collaborer à la gestion d'un projet informatique
  - Développer des composants métier
  - Construire une application organisée en couches
  - Préparer et exécuter les plans de tests d'une application
  - Préparer et exécuter le déploiement d'une application

N'ont pas été couvertes les 3 compétences suivantes :

- Développer une interface utilisateur
- Développer des pages web en lien avec une base de données
- Développer une application de mobilité numérique

## Glossaire et définitions

---

**API** : En informatique, une interface de programmation (souvent désignée par le terme API pour Application Programming Interface) est un ensemble normalisé de classes, de méthodes ou de fonctions qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels. Elle est offerte par une bibliothèque logicielle ou un service web, le plus souvent accompagnée d'une description qui spécifie comment des programmes consommateurs peuvent se servir des fonctionnalités du programme fournisseur.

Des logiciels tels que les systèmes d'exploitation, les systèmes de gestion de base de données, les langages de programmation, ou les serveurs d'applications comportent une interface de programmation <sup>1</sup>.

**Eclipse** : Eclipse est un environnement de programmation (IDE) pour le langage Java de la fondation Eclipse. Ce logiciel simplifie la programmation grâce à un certain nombre de raccourcis et notamment grâce à la possibilité d'intégrer de nombreuses extensions. Au fur et à mesure de l'avancement du code, Eclipse compile automatiquement le code et signale les problèmes qu'il détecte.

**Framework** : Appelé en français cadre d'applications, c'est un ensemble de classes d'objets, utilisables pour créer des applications informatiques. Le framework fournit au développeur des objets d'interface (bouton, menu, fenêtres, boîtes de dialogue), des objets de service (collections, conteneurs) et des objets de persistance (accès aux fichiers et aux bases de données) prêts à l'emploi. Le développeur peut donc s'appuyer sur ces classes et se concentrer sur les aspects métier de son application.

---

1. [http://fr.wikipedia.org/wiki/Interface\\_de\\_programmation](http://fr.wikipedia.org/wiki/Interface_de_programmation)

**Géomatique** : La géomatique est la combinaison syntaxique de deux mots : Géographie et Informatique. Le mot géomatique a été déterminé pour regrouper de façon cohérente l'ensemble des connaissances et technologies nécessaires à la production et au traitement des données numériques décrivant le territoire, ses ressources ou tout autre objet ou phénomène ayant une position géographique. La géomatique est un domaine qui fait appel aux sciences, aux technologies de mesure de la terre ainsi qu'aux technologies de l'information pour faciliter l'acquisition, le traitement et la diffusion des données sur le territoire (aussi appelées « données spatiales » ou « données géographiques »). La géomatique est étroitement liée à l'information géographique qui est la représentation d'un objet ou d'un phénomène localisé dans l'espace. Ainsi, la géomatique regroupe l'ensemble des outils et méthodes permettant de représenter, d'analyser et d'intégrer des données géographiques<sup>2</sup>.

**POM** : Chaque projet ou sous-projet est configuré par un POM « Project Object Model » qui contient les informations nécessaires à Maven pour traiter le projet (nom du projet, numéro de version, dépendances vers d'autres projets, bibliothèques nécessaires à la compilation, noms des contributeurs etc.). Ce POM se matérialise par un fichier pom.xml à la racine du projet. Cette approche permet l'héritage des propriétés du projet parent. Si une propriété est redéfinie dans le POM du projet, elle recouvre celle qui est définie dans le projet parent. Ceci introduit le concept de réutilisation de configuration. Le fichier pom du projet principal est nommé pom parent. Il contient une description détaillée de votre projet, avec en particulier des informations concernant le versionnage et la gestion des configurations, les dépendances, les ressources de l'application, les tests, les membres de l'équipe, la structure et bien plus.

**PostgreSQL** : PostgreSQL est un système de gestion de bases de données relationnelles objet (Manuel PostgreSQL). PostgreSQL est un outil Open Source et disponible gratuitement, compatible avec les systèmes d'opérations les plus connus (Linux, Unix (Mac OSX, Solaris etc.) et Windows). PostgreSQL propose des interfaces de programmations pour des langages de programmation comme Java, C++, Python etc. Le développement de PostgreSQL a débuté en 1986 (appelé à l'époque Postgres). En 1995, les développeurs ajoutent un interpréteur de langage SQL à l'outil. A partir de 1996, l'outil s'appelle PostgreSQL afin de souligner le lien entre Postgres et le langage SQL. PostgreSQL peut être facilement étendu par l'utilisateur en ajoutant de nouvelles fonctions, de nouveaux opérateurs ou même de nouveaux langages de procédure.

---

2. <http://www.sig-geomatique.fr/sig-geomatique.html>



**PostGIS** : PostGIS est une extension du système de gestion de base de données PostgreSQL qui permet de stocker des données (objets) géographiques dans la base de données. Cette extension permet d'utiliser une base de données PostgreSQL comme une base de données dans n'importe quel projet SIG. PostGIS est compatible avec de nombreux autres outils SIG comme par exemple QGIS, Mapserver, etc...

**REST** : REST « Representational State Transfer » est un style d'architecture logicielle comprenant des lignes directrices et des meilleures pratiques pour la création de services Web évolutifs. REST est un ensemble coordonné de contraintes appliqué à la conception de composants dans un système hypermédia distribué qui peut conduire à une architecture plus performante et maintenable.

REST a gagné sa réputation à travers le web comme une alternative plus simple à SOAP et des services basés sur un WSDL. Les systèmes « RESTful » peuvent généralement, mais pas toujours, communiquer avec les verbes du protocole HTTP (GET, POST, PUT, DELETE, etc.) utilisés par les navigateurs Web pour récupérer des pages Web et envoyer des données à des serveurs distants.

**SAAS** : SAAS « Software as a Service » Le logiciel en tant que service est un modèle d'exploitation commerciale des logiciels dans lequel ceux-ci sont installés sur des serveurs distants plutôt que sur la machine de l'utilisateur. Les clients ne paient pas de licence d'utilisation pour une version, mais utilisent généralement gratuitement le service en ligne ou payent un abonnement récurrent<sup>3</sup>.

**SGBD** : Système de gestion de base de données - Un ensemble de programmes qui permettent l'accès à une base de données<sup>4</sup>.

**SoapUI** : SoapUI est outil graphique qui permet de tester des services web basés sur diverses technologies. Il est disponible en deux versions : une version gratuite et open source et seconde version payante. Il est également disponible sous forme de plugin pour les IDE Netbeans, IntelliJ IDEA et Eclipse. SoapUI est développé entièrement en Java et utilise Java Swing pour son GUI, il fonctionne donc sur la plupart des systèmes d'exploitation et en plus il est disponible sous licence GNU. L'outil gère respectivement les services web basés sur les technologies telles que le HTTP (S), HTML, SOAP (WSDL), REST, AMF, JDBC et JMS.

---

3. [http://fr.wikipedia.org/wiki/Logiciel\\_en\\_tant\\_que\\_service](http://fr.wikipedia.org/wiki/Logiciel_en_tant_que_service)

4. <http://www.futura-sciences.com/fr/definition/t/informatique-3/d/sghd-2525/>

**SVN** : SVN ou Subversion est un système de gestion de version, conçu pour remplacer CVS. Concrètement, ce système permet aux membres d'une équipe de développeur de modifier le code du projet quasiment en même temps. Le projet est en effet enregistré sur un serveur SVN et à tout moment, le développeur peut mettre à jour une classe avant de faire des modifications pour bénéficier de la dernière version et a la possibilité de comparer deux versions d'un même fichier.

**WS** : Acronyme de « Web Service » ou Service Web. Un WS est un programme informatique de la famille des technologies web permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués. Il s'agit donc d'un ensemble de fonctionnalités exposées sur internet ou sur un intranet, par et pour des applications ou machines, sans intervention humaine, de manière synchrone ou asynchrone. Actuellement, le protocole de transport est essentiellement HTTP(S)<sup>5</sup>.

---

5. [http://fr.wikipedia.org/wiki/Service\\_web](http://fr.wikipedia.org/wiki/Service_web)

## 8.1 Redmine

Redmine est considéré comme l'un des outils de gestion de projets collaboratifs Open Source parmi les plus aboutis. Il recouvre un ensemble de fonctionnalités dont un aperçu est donné ci-dessous, comme la gestion multi-projets, la gestion des demandes d'évolution et des bugs, la gestion et l'indexation des documentations techniques, mais aussi la gestion des droits et des profils des différents intervenants. Il propose aussi une console de suivi de l'état d'avancement des projets, des tâches, des recettes... sous forme de diagramme de Gantt.

Redmine est une forge logicielle sous licence GPL. Ses principaux concurrents se nomment Trac, Retrospectiva, Django Projector ou encore InDefero. Notons que Jira ne trouve pas sa place ici, bien qu'il soit le concurrent le plus sérieux de Redmine, parce que Jira est sous Dual licence (l'utilisation commerciale nécessite une licence payante non-Open Source).

Redmine offre un ensemble de fonctionnalités comme par exemples :

- Prise en charge de plusieurs projets
- Contrôle d'accès avec un modèle flexible de rôles
- Gestion avancée des tickets
- Diagramme de Gantt et calendrier
- Publication de news, documents et gestionnaire de fichiers
- Notifications par emails et flux ATOM
- Wiki et forums par projet
- Outil de suivi du temps
- Champs personnalisables pour les tickets, suivi de temps, projets et utilisateurs
- Intégration avec plusieurs SCM : SVN, CVS, Git, Mercurial, Bazaar et Darcs
- Une communauté active et un ensemble d'outils connexes

Plus de détails est disponible sur le wiki du projet : <http://www.redmine.org/projects/redmine/wiki>.

## 8.2 Classe Upload.java

```
// Empty constructor
public Upload (){
}

// Constructor
public Upload(String id, String name, String uploadStatus, String
    organization, String uploadPath){
    super();
    this.id=id;
    this.name = name;
    this.uploadStatus = uploadStatus;
    this.organization = organization;
    this.uploadPath = uploadPath;
}

// Getters & Setters
public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getUploadStatus() {
```

```
        return uploadStatus;
    }

    public void setUploadStatus(String uploadStatus) {
        this.uploadStatus = uploadStatus;
    }

    public String getOrganization() {
        return organization;
    }

    public void setOrganization(String organization) {
        this.organization = organization;
    }

    public String getUploadPath() {
        return uploadPath;
    }

    public void setUploadPath(String uploadPath) {
        this.uploadPath = uploadPath;
    }

    // suite des Getters & Setters
```

### 8.3 Classe UploadDAO.java

```
// Create
public String create(Upload up)
{
    System.out.println("Upload instance created : "+up.getId()+" | "+up.getName()
        +" | "+up.getUploadStatus()+" | "+up.getOrganization()+" | "+up.
        getUploadPath());
    return persist(up).getId();
}

// Read one
public Upload findByID(String id)
{
    //return get(id);
    Query query = namedQuery("com.mobigis.dao.Upload.findByID");
    query.setParameter("id", id);
    List<Upload> res = list(query);
    if(res != null && res.size()>0)
    {
        return res.get(0);
    }
    return null;
}
```

## 8.4 Extrait du log dans Eclipse

```
INFO [2015-06-11 15:31:52,883] com.mobigis.thread.AbstractUploadManagerThread:
    GTFS validation JSON report done !
file unzip : agency.txt
file unzip : calendar.txt
file unzip : calendar_dates.txt
file unzip : fare_attributes.txt
file unzip : fare_rules.txt
file unzip : frequencies.txt
file unzip : routes.txt
file unzip : shapes.txt
file unzip : stop_times.txt
file unzip : stops.txt
file unzip : trips.txt
Extract Done
Clean directory Done
upload status = UPLOADED
INFO [2015-06-11 15:31:52,920] com.mobigis.thread.AbstractUploadManagerThread:
    Process upload to Location = C:\mobianalystserver\download\tisseo\gtfs\215
    b8557-f7c9-4b06-aced-f79bde953c65
INFO [2015-06-11 15:31:52,920] com.mobigis.thread.AbstractUploadManagerThread:
    Upload file done ! Id = 215b8557-f7c9-4b06-aced-f79bde953c65 | Status =
    UPLOADED
C:\mobianalystserver\download\tisseo\gtfs\215b8557-f7c9-4b06-aced-
    f79bde953c65
INFO [2015-06-11 15:31:52,965] org.hibernate.engine.internal.
    StatisticalLoggingSessionEventListener: Session Metrics {
    36877 nanoseconds spent acquiring 1 JDBC connections;
    0 nanoseconds spent releasing 0 JDBC connections;
    358836 nanoseconds spent preparing 4 JDBC statements;
    6286839 nanoseconds spent executing 4 JDBC statements;
    0 nanoseconds spent executing 0 JDBC batches;
    0 nanoseconds spent performing 0 L2C puts;
```

```
0 nanoseconds spent performing 0 L2C hits;
0 nanoseconds spent performing 0 L2C misses;
24185654 nanoseconds spent executing 1 flushes (flushing a total of 2
    entities and 2 collections);
42650 nanoseconds spent executing 1 partial-flushes (flushing a total of 0
    entities and 0 collections)
}
127.0.0.1 -- [11/Jun/2015:15:31:50 +0000] "POST /upload/gtfs HTTP/1.1" 200 61 "-" "
    Apache-HttpClient/4.1.1 (java 1.5)" 2254
```



## 8.5 Structure d'une application Web

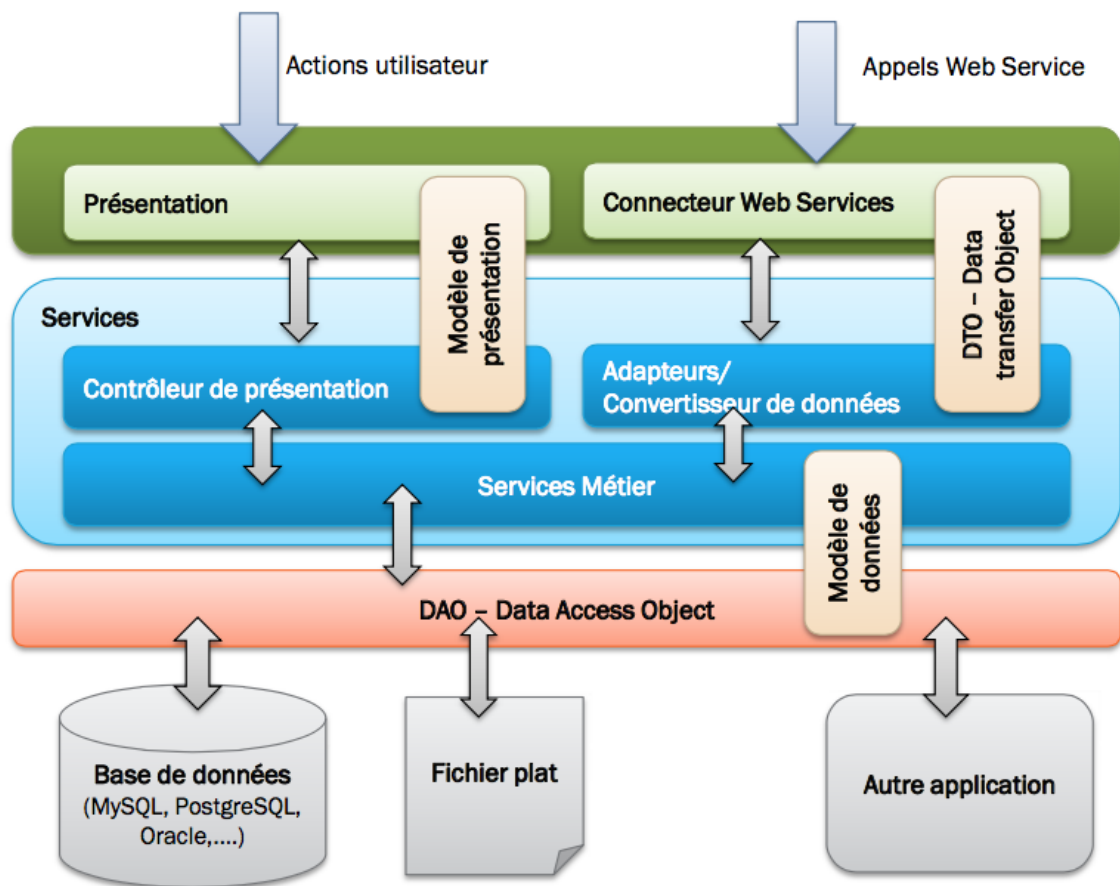


FIGURE 8.1 – Structure typique d'une application Web (Laure Bouquety ©)

## 8.6 Requête/Réponse

### Exemple de requête « GET BY ID »

<https://localhost:9876/upload/gtfs/status/a74f467e-4e77-4aad-9254-c3c3dad64b31>

### Exemple de Réponse JSON

```
{
  "id": "a74f467e-4e77-4aad-9254-c3c3dad64b31",
  "name": "st_gtfs_good.zip",
  "uploadStatus": "UPLOADED",
  "organization": "tiseo",
  "uploadPath": "C:\\mobianalystserver\\download\\tiseo\\gtfs\\a74f467e-4e77-4aad-9254-c3c3dad64b31",
  "publishedDate": 1435591474335,
  "reportPath": "C:\\mobianalystserver\\download\\tiseo\\gtfs\\a74f467e-4e77-4aad-9254-c3c3dad64b31\\st_gtfs_good\\validation_st_gtfs_good.json",
  "persisted": false
}
```

# Webographie

## **Sites sur les données/outils « métiers » :**

<https://developers.google.com/transit/gtfs/examples/gtfs-feed>

<https://developers.google.com/transit/gtfs/>

<https://github.com/google/transitfeed/wiki/FeedValidator>

<http://onebusaway.org/developer-information/>

<https://github.com/OneBusAway/onebusaway/wiki/Importing-source-code-into-Eclipse>

<http://developer.onebusaway.org/modules/onebusaway-gtfs-modules/current/apidocs/index.html>

## **Sites sur les frameworks et outils utilisés :**

<http://www.objis.com/formation-java/tutoriel-formation-maven-2.html>

<http://mvnrepository.com/>

<http://www.dropwizard.io/>

<https://github.com/bucharest-jug/dropwizard-todo>

<http://blog2dev.blogspot.fr/2015/03/dropwizard.html>

<http://docs.jboss.org/hibernate/orm/4.3/manual/en-US/html/>

<http://www.tutorialspoint.com/hibernate/>

<http://blog.xebia.fr/2011/11/14/rest-java-serveur/>

<http://blog.xebia.fr/2010/08/18/comparatif-des-librairies-json/>

Et pour finir, un grand merci à <http://www.mkyong.com/>, tout simplement.