Institut **S**upérieur d'**I**nformatique, de **M**odélisation et de leurs **A**pplications

Campus des Cézeaux
1 rue de la Chébarde
TSA 60125
CS 60026
63 178 Aubière CEDEX

RAPPORT D'INGENIEUR

INTEGRATION D'APPLICATION EN ENTREPRISE

FILIERE : GENIE LOGICIEL ET SYSTEMES INFORMATIQUES

GESTION DE RENDEZ-VOUS DE MEDECINS

Présenté par : **Pierre CHEVALIER & Benoît GARÇON**

Responsable ISIMA : **Pierre Colomb**

Année 2016 – 2017

TABLE DES FIGURES ET ILLUSTRATIONS

TABLE DES MATIERES

INTRODUCTION

Dans le cadre du cours d'intégration d'application en entreprise de la filière Génie Logiciel et Systèmes Informatiques de la troisième année d'étude à l'ISIMA, il nous a été demandé de produire une application J2EE fournissant un service web permettant de gérer les rendez-vous d'un cabinet de médecins.

L'objectif est donc de développer une interface de programmation, ou API, sous la forme d'un service web REST consommable par des clients tiers par le protocole HTTP afin de mettre en pratique tout ce qui a été vu en cours.

Ce rapport a pour but de décrire notre projet, notre analyse et les pistes de développement explorées ainsi que le résultat final de notre travail. Nous commencerons donc par une analyse du sujet afin de mettre en évidence les axes principaux du travail. Nous verrons ensuite comment nous avons conçu notre application et quels outils ont eu un rôle majeur dans ce processus. Enfin nous détaillerons le rendu final de cette application, son fonctionnement, son déploiement et ses tests.

Chapitre 1 : ANALYSE

L'objectif de ce projet est donc de fournir un service web REST pour pouvoir gérer un cabinet médical. Pour ce faire on peut sortir du sujet les exigences principales de satisfactions :

1. Persister les données : les informations sur les médecins, patients et rendez-vous doivent être persistées et survivre à l'arrêt de l'application.
2. Fournir les opérations élémentaires : il faut pouvoir manipuler les entités du cabinet pour les créer (C : CREATE), les lire (R : READ), les mettre à jour (U : UPDATE) et les supprimer (D : DELETE). C'est ce qu'on appelle le CRUD et ces opérations sont généralement présentes sur toutes les applications à données persistées, c'est pourquoi il semble inutile de les réimplémenter une énième fois, ce que nous verrons par la suite.
3. Fournir des opérations avancées :
    a. L'énumération des créneaux libres : liste des créneaux disponibles pour un médecin et un jour donné
    b. La prise de rendez-vous
    c. La modification de rendez-vous
    d. L'annulation de rendez-vous
4. Donner accès aux opérations par un service web REST : la présentation des données se fera donc par un client capable de communiquer en http.
5. Produire un script permettant le déploiement automatique en production de l'application.

A ces exigences explicites du sujet on peut rajouter les tests, la documentation et l'intégration continue de notre projet comme exigences naturelles.

Dès lors on peut voir que ce projet se découpe en deux parties majeures : les processus relatifs aux méthodes de génie logiciel (déploiement, ci, tests, …) et le développement à proprement parlé de l'application.

Concernant les méthodes de génie logiciel, il sera donc nécessaire de disposer pour le déploiement de machine vierge afin de proposer des environnements de test sains et reproductibles. Un serveur de CI prend alors tout son sens. Couplé à ceci il faudra aussi disposer d'un outil permettant d'automatiser les différentes étapes de la vie de l'application.

Du côté de l'application les points 1 à 4 montrent bien un découpage naturel en trois parties de celle-ci. En effet, on peut distinguer un modèle classique de 3-tiers comme sur la Figure 1 :

- Une couche d'accès aux données : c'est la partie qui gérera les données persistées et les fournira à l'application, c'est un peu l'interface entre la base de données et l'application.
- Une couche de traitement des données : c'est le cœur de l'application puisque c'est la partie qui va effectuer les traitements demandés par l'utilisateur sur les données

fournie par la couche précédente. Cette couche contiendra donc les opérations des points 2 et 3.

- Une couche de présentation des données : c'est le point d'entrée de l'application pour l'utilisateur, l'interface permettant d'accéder aux traitements de la couche inférieure. C'est la seule couche accessible par les utilisateurs finaux et ce sera donc le service web REST.

Figure 1 - Modèle 3-tiers

Le cœur de l'application va donc tourner autour des quatre entités du sujet : les médecins, les patients, les créneaux et les rendez-vous. Comme on peut le voir sur la Figure 2, il y a plusieurs points à éclaircir :

- Un médecin dispose de zéro ou plusieurs créneaux (non disponibles) et un créneau doit avoir au moins un médecin. En listant les créneaux occupés d'un médecin on économise de l'espace mémoire par rapport au stockage des créneaux libres (et c'est plus utile). Un médecin ne peut pas avoir deux créneaux qui se chevauchent ou en dehors des horaires de travail.
- Un patient peut prendre zéro ou plusieurs rendez-vous et un rendez-vous doit avoir au moins un patient. Ici un patient peut prendre plusieurs rendez-vous sur des créneaux se chevauchant (c'est sa responsabilité et puis en réalité rien n'empêche quelqu'un de prendre deux rendez-vous au même moment).
- Un rendez-vous a lieu sur un seul créneau : le créneau n'a pas de contrainte de durée hormis le fait qu'il doit commencer et se finir durant la même période ouvrable et avoir une durée positive.
- Une classe mère pourrait être créée pour les médecins et les patients (une classe personne par exemple) mais le choix a été fait de ne pas le faire puisque cela servirait seulement à mutualiser deux attributs (nom et prénom) mais ceci est plus un choix de conception.

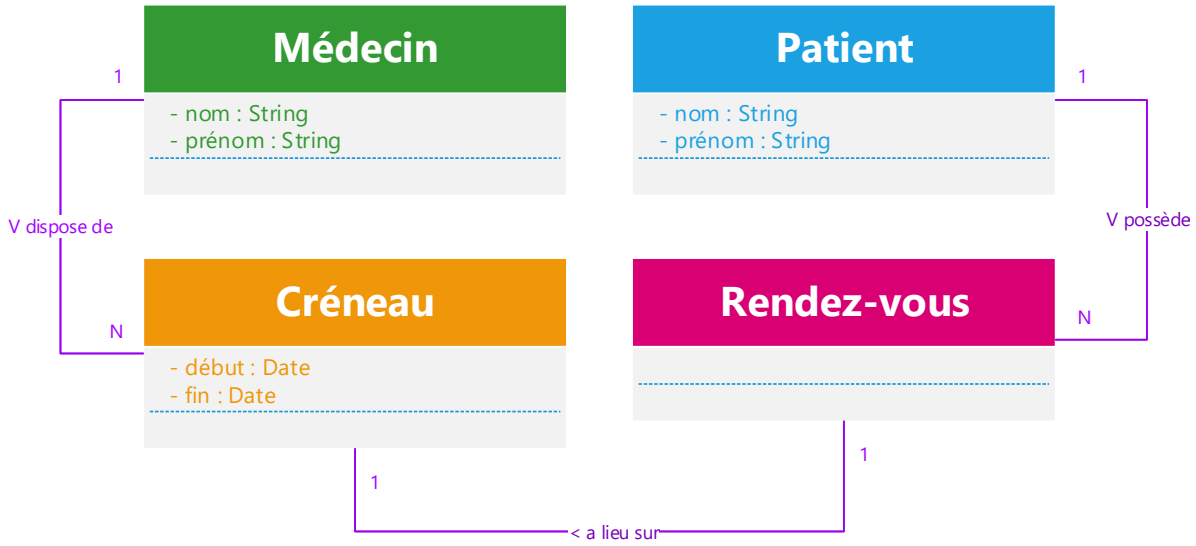

Figure 2 - Diagramme des entités persistées

Pour finir il est important de noter que nous avons fait le choix qu'une journée ouvrable commençait à 8h et finissait à 20h, les week-ends sont des jours comme les autres, libre au médecin de travailler.

Nous allons maintenant voir comment nous avons mis en place l'accès, le traitement et la présentation des données ainsi que le déploiement de l'application.

Chapitre 2 : CONCEPTION

Nous allons maintenant aborder la conception de l'application rdvMed puis le passage en revue des outils et méthodes utilisées.

2.1 Métaprogrammation et adaptation

Comme nous l'évoquions plus tôt le sujet de ce projet est assez classique dans sa forme et ses fonctionnalités. De nombreux développeurs ont depuis des années développées des applications manipulant des entités en base à partir d'opérations CRUD. Mais à l'heure de l'ingénierie dirigée par les modèles, il nous a semblé judicieux de générer par métaprogrammation toutes ces parties génériques qu'un ordinateur peut produire. Nous avons ainsi pu nous concentrer sur les parties plus techniques.

Nous avons donc décider de générer un service web complet en J2EE à partir d'un simple script SQL (Figure 3) décrivant les quatre entités médecin, patient, créneau et rendez-vous. A partir de cette base de données qui est en un sens un modèle de description de données, notre IDE NetBeans est capable de générer les classes entités, les EJB et les API service web correspondant permettant d'effectuer les opérations CRUD.

```
CREATE DATABASE `tp_iae`;

USE `tp_iae`;

CREATE TABLE `medecins` (
  `id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
  `nom` varchar(64) DEFAULT NULL,
  `prenom` varchar(64) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=latin1;

CREATE TABLE `patients` (
  `id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
  `nom` varchar(64) DEFAULT NULL,
  `prenom` varchar(64) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=latin1;

CREATE TABLE `creneaux` (
  `id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
  `debut` datetime NOT NULL,
  `fin` datetime NOT NULL,
  `medecin` smallint(5) unsigned NOT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_medecin_numero` (`medecin`),
  CONSTRAINT `fk_medecin_numero` FOREIGN KEY (`medecin`) REFERENCES
`medecins` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `rdv` (
  `id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
  `date` datetime NOT NULL,
  `patient` smallint(5) unsigned NOT NULL,
  `creneau` smallint(5) unsigned NOT NULL,
  PRIMARY KEY (`id`),
```

```
  KEY `patient` (`patient`),
  KEY `creneau` (`creneau`),
  CONSTRAINT `rdv_ibfk_1` FOREIGN KEY (`patient`) REFERENCES `patients`
(`id`),
  CONSTRAINT `rdv_ibfk_2` FOREIGN KEY (`creneau`) REFERENCES `creneaux`
(`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Figure 3 - Script SQL de création de la base de données du projet (script.sql)

Le script contient donc les informations du diagramme précédent avec les contraintes sur les associations grâce aux clés étrangères.

Toutefois des adaptations ont dû être apportées au code généré pour avoir une application cohérente. En outre, les parties présentation et traitement des données étaient représentées par une seule classe : nous l'avons donc découpé en deux avec des EJB Stateless pour le traitement et des web service REST pour la présentation. Il y avait aussi un problème au niveau de l'unité de persistance générée et nous avons donc créé la nôtre. D'autres modifications éparses ont encore été faite pour répondre plus précisément aux spécificités du sujet mais le gros du travail a été généré automatiquement.

Nous avons donc pu nous concentrer sur l'ajout des traitements avancés en implémentant des méthodes principalement dans l'EJB des rendez-vous (RdvEJB). Ainsi chaque classe de service web appelle un EJB (qui peut appeler d'autres EJB) et on a obtenu un découplage maximal entre service et traitement.

Nous avons développé des tests non pas pour les entités ou les EJB mais bien pour l'API du service web en considérant l'application comme une boite noire. La base des tests a aussi été générée par métaprogrammation et a été largement adaptée pour obtenir 31 tests utiles.

Nous allons maintenant voir ce que nous avons utilisé pour obtenir l'application rdvMed.

2.2 Méthodes et outils de développement

Une des contraintes de ce projet était d'utiliser le J2EE qui est une plate-forme très répandue pour ce genre de développement. Tout comme en travaux pratiques, nous avons utilisé MySQL comme SGBD pour la persistance des données et payara 4.1 comme serveur d'applications (serveur GlassFish administrable par asadmin). En termes d'environnement de développement nous avons donc utilisé NetBeans qui propose de nombreux outils pour le développement J2EE avec en particulier un serveur intégré de développement GlassFish et des outils de métaprogrammation.

Nous avons aussi versionné l'ensemble du projet sur GitHub avec git afin de garder une trace de chaque étape et pour éviter toute régression du code nous avons utilisé Travis CI pour faire de l'intégration continue. Travis CI a aussi été choisi pour tester plus facilement le script de déploiement de notre application puisqu'il propose à chaque build un environnement propre. Comme tout ceci est automatique nous avons dû déléguer la génération de l'application, de la doc et des tests à Ant. La release du projet est faite automatiquement lorsqu'un build se

termine avec succès sur la branche master : on peut ainsi retrouver sur GitHub Releases le code source, le rapport et le script shell de déploiement.

Pour ce qui est de la méthode de développement, le travail a été commun sur tout ce qui est métaprogrammation et déploiement mais un peu plus Xtreme programming pour ce qui est du développement des fonctionnalités et des tests : pendant que l'un développés l'application l'autre écrivait les tests pour les nouvelles features.

Tout ceci a donc abouti à la réalisation d'une application fonctionnelle que nous allons décrire dans la partie suivante.

Chapitre 3 : RESULTAT

Nous avons créé un script bash (Figure 4) qui permet d'installer toutes les dépendances, télécharger le projet configurer le serveur, générer l'application, la déployer, la tester et générer la documentation et les rapports de tests.

```
#!/bin/bash

# updating system only if needed
apt-get update -y -qq
# apt-get upgrade -y -qq
# installing java jdk 8
apt-get install -y oraclejdk8 -qq
# installing mysql
apt-get -y install mysql-server -qq
mysqladmin -u root password iae2016
# installing git
apt-get install -y git -qq
# downloading project
git clone https://github.com/ZZ3-IAE/IAE.git
# installing payara
cp -r IAE/matos/payara41 .
# driver jdbc
cp IAE/matos/mysql-jdbc.jar payara41/glassfish/lib
# creation of database
mysql -u root --password=iae2016 < IAE/matos/script.sql
# starting payara
payara41/bin/asadmin start-domain
# creating a connexion pool
payara41/bin/asadmin create-jdbc-connection-pool --datasourceclassname
com.mysql.jdbc.jdbc2.optional.MysqlDataSource --restype
javax.sql.DataSource --property
user=root:password=iae2016:DatabaseName=tp_iae:ServerName=localhost:port=
3306 tp_iae_pool
payara41/bin/asadmin ping-connection-pool tp_iae_pool
# creating a data source
payara41/bin/asadmin create-jdbc-resource --connectionpoolid tp_iae_pool
tp iae
# build
chmod +rwx . -R
cd IAE/rdvMed
chmod +rwx ../matos/apache-ant-1.10.1/bin/ant
../matos/apache-ant-1.10.1/bin/ant dist
cd ../..
# deploying app
payara41/bin/asadmin deploy IAE/rdvMed/dist/rdvMed.war
# launching demo
cd IAE/rdvMed
../matos/apache-ant-1.10.1/bin/ant mytest
../matos/apache-ant-1.10.1/bin/ant javadoc
```

Figure 4 - Script de déploiement (deploy.sh)

Nous avons donc développé 31 tests qui viennent vérifier le bon fonctionnement et le bon disfonctionnement de différentes requêtes sur les quatre entités. Ces tests sont tous positifs en environnement de développement et sur les environnement Travis CI. Pour plus de tests, il est possible d'interagir manuellement avec le serveur grâce à une commande comme curl mais pour ce faire il est nécessaire d'apporter quelques explications sur les différents services.

Tout d'abord l'application est par défaut atteignable sur le port 8080 avec son nom c'est-à-dire pour un usage local : http://localhost:8080/rdvMed. Toutes les réponses se font en JSON. Avec ceci il est possible d'atteindre la page d'accueil du service. Pour obtenir un service il faut simplement ajouter « ws/nomService », par exemple :

- Pour les patients on a l'url : http://localhost:8080/rdvMed/ws/patients.
- Pour les médecins on a l'url : http://localhost:8080/rdvMed/ws/medecins.
- Pour les créneaux on a l'url : http://localhost:8080/rdvMed/ws/creneaux.
- Pour les rendez-vous on a l'url : http://localhost:8080/rdvMed/ws/rdv.

Ensuite tout dépend de la méthode http utilisé :

- Avec un GET sur les url précédentes on obtient la liste de toutes les entités relatives au service.
- Avec un POST on peut ajouter une entité à condition d'envoyer l'objet JSON correspondant.

Si on ajoute l'id d'une entité à l'URL il est possible de manipuler une entité particulière (http://localhost:8080/rdvMed/ws/patients/12) :

- Avec un GET on obtient l'entité demandée.
- Avec un PUT on peut mettre à jour cette entité à condition d'envoyer l'objet JSON correspondant.
- Avec un DELETE on peut la supprimer.

Ainsi la création, la modification et l'annulation d'un rendez-vous s'effectue avec les POST, PUT et DELETE correspondant sur rdv.

Une petite particularité est l'inventaire des créneaux libres : pour obtenir les disponibilité d'un médecin pour un jour donné, il faut utiliser, associé à l'envoi par POST du médecin en question, une url de ce type : http://localhost:8080/rdvMed/ws/rdv/libres/2017/04/12. Ainsi on obtient pour le médecin demandé ses disponibilités pour le 12 avril 2017. Attention tout de même à la façon d'envoyer les dates au service : en Java un 4 signifie le mois de mai mais dans cette commande le 4 est bien le mois d'avril.

Il existe aussi d'autres commandes dont le détail ici ne revêt pas un grand intérêt, il est donc temps de conclure.

CONCLUSION

Nous avons donc réussi à produire un service web répondant aux critères du sujet. Notre application rdvMed conçu en J2EE permet donc de manipuler des entités persistées comme des médecins et des patients, avec toutes les fonctionnalités CRUD essentielles. Il est aussi possible d'organiser des rendez-vous, de les modifier et les annuler, et aussi de lister les créneaux libres pour les médecins.

L'ensemble de ce travail est disponible librement sur GitHub et géré par intégration continue sur Travis CI permettant un développement ouvert et collaboratif. Un simple script disponible dans la release permet de télécharger, builder, déployer et tester simplement l'application avec une seule commande (sur Travis ou sur une machine vierge).

Tout ceci a pu être développé facilement grâce aux outils de génie logiciel à notre disposition : les outils de métaprogrammation de NetBeans pour générer du code basique, l'outil Ant pour gérer la compilation, le packaging, la documentation et les tests automatiquement et la CI pour pouvoir tester notre déploiement automatiquement sans recréer manuellement une nouvelle machine virtuelle pour chaque essai.

Notre application répond au cahier des charges initial mais pourrait fournir encore plus. Par exemple nous avions pensé fournir un client web Angular consommant notre service. Nous l'avons fait grâce à NetBeans qui a généré automatiquement une interface graphique basique à partir de notre service mais nous l'avons supprimé puisqu'il sortait du cadre du projet. Il pourrait être intéressant donc de poursuivre les travaux sur une partie client et ajouter au serveur des éléments lui permettant d'être mis à l'échelle dans le « cloud » (architecture en micro-services, circuit breaker, feature flipping, etc.).