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.

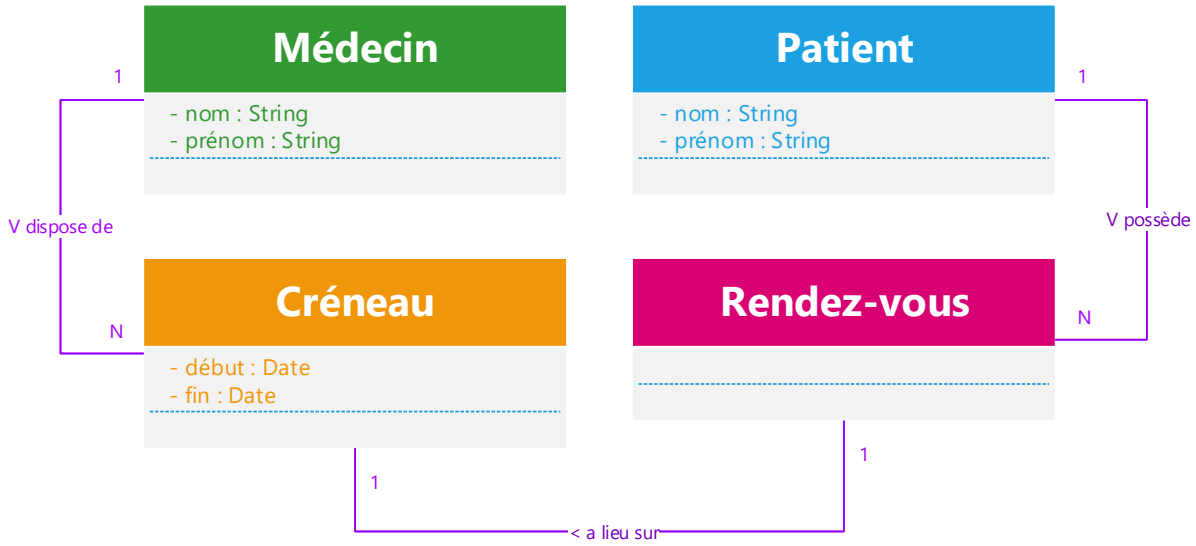

Figure 2 - Diagramme des entités persistées

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

2.1 Méthodes et outils

2.2 Métaprogrammation

Utilisation de netbeans à partir du sql

2.3 Adaptation

Adaptation du code

Chapitre 3 : RESULTAT

Présentation du script de déploiement + Tests et utilisation du service

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.).