SUPRALOG
CONSEIL ▲ TECHNOLOGIE ▲ PROGICIEL

Grenoble INP
ensimag

Grenoble INP - ENSIMAG
École nationale supérieure d'informatique et de mathématiques appliquées
de Grenoble

Pré-Rapport de Projet de Fin d'Etudes

Effectué chez Supralog

Architecture et Développement Java Backend pour une Solution E-Santé

Alexandre Rupp
$3A$ — Filière Ingénierie des systèmes d'information% Your institution

29 Février 2016 - 26 Août 2016

Supralog
Immeuble Le Chorus
2203, Chemin de Saint-Claude
06600 ANTIBES

Responsable de stage
Nicolas THIBAULT
Tuteur de l'école
Sylvain BOUVERET

Contacts

Nicolas Thibault (Tuteur de la structure d'accueil)

- **Adresse :**

  SUPRALOG
  Immeuble Le Chorus
  2203, Chemin de Saint Claude
  06600 Antibes
  FRANCE

- **Email :** nthibault@supralog.com

Sylvain Bouveret (Tuteur de l'école)

- **Adresse :**

  Bureau D301 bis,
  Bâtiment D 681,
  rue de la Passerelle.
  BP 72. F-38402 SAINT-MARTIN-D'HÈRES.
  FRANCE.

- **Email :** sylvain.bouveret@imag.fr

Alexandre Rupp (Stagiaire)

- **Adresse :**

  Appt 4201
  Les callades de Sophia
  475 rue Evariste Galois
  Sophia Antipolis
  06560 Valbonne
  FRANCE

- **Téléphone :** 06 13 80 34 87
- **Email :** alexandre.rupp@ensimag.grenoble-inp.fr

Contents

List of Figures

Résumé

Dans le cadre de ma dernière année d'école d'ingénieur à l'Ensimag, j'ai réalisé mon projet de fin d'études d'une durée de six mois au sein de l'entreprise Supralog.

Le but du projet était de participer à la conception et au développement d'une application web d'e-santé à destination des cabinets médicaux. Cette application réalisée en Java, Spring et JSF selon une architecture 3-Tiers est la refonte d'un logiciel plus ancien qui est distribué en tant qu'application de bureau pour Windows. Le logiciel permet entre autre de gérer les feuilles de soins électroniques, la facturation, l'activité d'un cabinet et utilise la technologie *Sesam Vitale**.

Dans ce cadre, j'ai eu à réaliser le dossier médical d'un patient et les différentes fonctionnalités permettant de le gérer. Pour ce faire, j'ai d'abord fait un état des lieux des fonctionnalités présentes dans le logiciel d'origine, puis j'ai analysé l'existant sur la nouvelle application et j'ai conçu et implémenté une solution qui puisse s'y greffer.

Finalement, ce stage m'aura permis de développer mes compétences sur les technologies Java 8, Spring, JSF et Hibernate. J'aurais travaillé au sein d'une équipe qui applique les méthodes agiles et sur un projet qui utilise l'intégration continue. Enfin, cela m'a donné l'occasion de découvrir les aspects métiers liés aux prescriptions médicales, aux feuilles de soins et à l'assurance maladie.

Mots-clé: Java, JSF, Spring, Hibernate, e-santé, Architecture 3-Tiers, Sesam Vitale.

Glossary

FSE est l'accronyme de "Feuille de Soins Electronique".. 9

IDEA est la société créée en 1987 qui est responsable de la commercialisation de Topaze.. 7

IDLOG est un groupe familial créé en 2002 afin de regouper les sociétés SUPRALOG et IDEA. 7

OLE est l'accronyme de "Object Linking and Embedding". Il s'agit d'un protocole développé par Microsoft permettant de faire de la liaison dynamique d'objets dans Windows.. 10

ORM est l'accronyme de "Object-relational mapping". Il s'agit d'une technique permettant de manipuler les données de la base de données comme s'il s'agissait d'objets.. 9

RTF est l'accronyme de "Rich Text Format". Il s'agit d'un format de fichier développé par Microsoft et utilisable dans la plupart des éditeurs de textes.. 11

SAAS "Software As A Service" ou logiciel en tant que service. Il s'agit d'un business modèle visant à vendre des accés à une application qui tourne sur un serveur, plutôt que de vendre le logiciel comme un produit.. 6

Sesam Vitale Le programme SESAM-Vitale est un programme de dématérialisation des feuilles de soins pour l'assurance maladie en France, qui repose sur la carte Vitale.. 4

SUPRALOG est la société fondée en 1997 qui a créé le progiciel d'e-santé Topaze.. 7

Topaze Web est la refonte web de Topaze Maestro.. 6

Topaze Maestro est l'une des versions rescente du progiciel d'e-santé actuellement commercialisé par Supralog. Le logiciel fonctionne comme une application de bureau pour Windows.. 6

WYSIWYG est l'accronyme de "What You See Is What You Get". Cela désigne le plus souvent un éditeur graphique qui produit le code de ce qui est affiché à l'écran.. 16

1 Introduction

Ce pré-rapport fait état du travail réalisé durant les six premières semaines de mon stage chez *Supralog*.
Historiquement, *Supralog* est la société éditrice du logiciel *Topaze Maestro**. Il s'agit d'un logiciel d'e-santé destiné aux cabinets médicaux. Le logiciel permet, entre autre, de gérer les feuilles de soins électroniques, la facturation, les patients et leur dossier médical.
Topaze Maestro a été créé comme un logiciel de bureau pour Windows et a progressivement intégré des éléments de réseau. Dans sa version actuelle il consiste en un client léger installable sur la machine client qui communique avec une base de données de *Supralog*.

Afin de suivre les évolutions du marché, *Supralog* souhaite produire une nouvelle version de Topaze qui soit entièrement dématérialisée et commercialisée en tant que service (SAAS*). Cette nouvelle version qui s'appelle *Topaze Web** est une refonte globale du logiciel. Mon travail durant le stage sera de participer à sa conception et à son développement.

Dans ce rapport j'évoquerai d'abord le contexte du stage, puis je détaillerai la problématique du projet avant d'aborder la solution mise en place. Enfin je livrerai mes premières impressions sur cette première partie du stage.

2 Contexte du stage

Le stage se déroule au sein de l'entreprise SUPRALOG, du groupe IDLOG.

2.1 Le groupe IDLOG

IDLOG* est un groupe familial créé en 2002 afin de regrouper les sociétés SUPRALOG* créée en 1997 et IDEA* créée en 1987.
Il réuni les activités d'intégration de technologies, d'édition de progiciels et d'assistance aux utilisateurs.

2.2 Entreprise

Supralog est un éditeur de logiciels avec une activité de conseil en systèmes d'information créée en 1997. L'entreprise compte 45 collaborateurs et est implantée sur la technopole de Sophia Antipolis.

Figure 1: Supralog

Les trois activités de l'entreprise

Supralog est organisé en trois practices : Conseil / Technologie / Progiciel.
L'activité historique est l'édition de progiciels. Supralog développe deux gammes de solutions:

- *Topaze* qui permet la gestion des cabinets médicaux.
- *Intr@ssoc* qui est destiné à la gestion des grandes associations et fédérations.

La deuxième activité de l'entreprise est le conseil, notamment pour *Air France* et *Amadeus*. Enfin, il y a l'activité technologie qui porte sur la conception et le développement de systèmes d'information en architecture client-serveur.

Quelques chiffres

Les progiciels développés par SUPRALOG sont utilisés par plus de 40 000 personnes en France.
Les activités de technologie et de conseil regroupent plus de 15 clients.

L'entreprise a réalisé un chiffre d'affaires de plus de 7 millions d'euros en 2014 et présente une croissance de 114% sur les 3 dernières années.

2.3 Environnement de travail

Mon stage se déroule au siège de l'entreprise au sein de l'équipe de développement du projet *Topaze Web*. L'équipe est constituée de 5 personnes: le chef de projet et directeur technique Nicolas Thibault, les ingénieurs développeurs Abdessalam Eljai et Anthony Biga, l'apprentis Tom Veniat et moi-même.

2.4 Le projet Topaze Web et la gamme Topaze

Topaze est une gamme de progiciels de gestion de cabinets pour professionnels des milieux médicaux et paramédicaux. Cette gamme a été créée par Supralog en 1997 et est historiquement le premier progiciel de santé à obtenir l'agrément SESAM Vitale, agrément permettant l'édition de feuilles de soins électroniques.

Depuis 1997, Topaze a connu de nombreuses évolutions et a été publié et agréé en plusieurs versions ayant des architectures différentes. Lors de sa création Topaze était une application de bureau, puis il a évolué progressivement pour s'ouvrir à internet.
En 2015, le projet Topaze Web a été initié avec comme objectif de donner naissance à une application web multi tiers offrant les mêmes fonctionnalités que la version "Maestro" de Topaze.

2.5 Mission

Mon rôle au sein de *Topaze Web* est tout d'abord de me familiariser avec l'architecture du projet et ses technologies (J2EE, Hibernate, Spring et JSF) et de participer à la conception et au développement de l'application.

Spécifications et contraintes

Les spécifications de l'application à réaliser sont basées sur l'existant : Topaze Web devra fournir des fonctionnalités similaires à celles de Topaze Maestro. Le design de l'application doit être modernisé, mais son ergonomie doit rester proche de l'existant pour ne pas déstabiliser l'utilisateur final. Enfin, un soin particulier doit être accordé à l'architecture, la conception et l'utilisation de patrons de conception lors du développement de l'application, afin de garantir la pérennité et la maintenabilité du logiciel au cours du temps.

Gestion de projet

La gestion du projet se fait en mode agile, selon une méthodologie proche du Kanban. Les fonctionnalités à développer en premier son choisies par le client (la société IDEA d'IDLOG), puis les tâches sont chiffrées et réparties par le chef de projet. Le logiciel *Jira* est utilisé pour l'assignation et le suivi des tâches.
À la fin de chaque sprint, une réunion est organisé avec le client afin de montrer l'évolution du projet.

3 Problématique

3.1 Contexte du projet

Le développement du projet s'effectue en sprints, selon la méthode Kanban. Le contenu de chaque sprint est décidé d'un commun accord entre le chef de projet et le client. Le but est de réaliser en premier lieu les fonctionnalités les plus importantes fonctionnellement.

3.1.1 Mise en place du projet

Durant cette première étape, il a fallu mettre en place l'architecture du projet, avec notamment les différents serveurs *J2EE/Spring*, la base de données *Postgre Sql*, l'ORM* Hibernate, la partie webapp avec *JSF* et *primefaces*. Cette partie est abordée plus en détails, dans la section Architecture.

3.1.2 Sprint 1

Le sprint 1, qui a eu lieu avant mon arrivée, avait pour but la création des fonctionnalités de base permettant la gestion d'un cabinet.

La première fonctionnalité développée durant ce sprint permet la création d'un cabinet médical avec l'ajout de praticiens (kinésithérapeute, infirmiers etc.) et l'ajout de patients au cabinet. La deuxième fonctionnalité porte sur la gestion des ordonnances pour les professions de type kinésithérapeute et infirmiers. Enfin, il y a la gestion des séances, avec la création d'un planning.

3.1.3 Sprint 2

Le second sprint du projet a commencé en début mars, peu de temps après mon arrivée. Il a été dimensionné pour tenir sur 6 semaines. Les objectifs de ce sprint portent sur la création des fonctionnalités suivantes :

- *Les Feuilles de Soins Electroniques* : Il s'agit des feuilles de soins qui sont créées par le praticien et qui décrivent la prise en charge du bénéficiaire des soins ainsi que les différents actes réalisés et leur coût. La FSE* est mise à jour avec la carte vitale du patient à la réalisation du dernier acte et est ensuite transmise à l'assurance maladie.
- *Le contexte de facturation* : Il comprend la gestion des tarifications, des types de couverture et des types de prise en charge. Cette fonctionnalité nécessite la lecture des cartes SESAM Vitale.
- *Le dossier médical du patient* : Il contient une liste des documents associés au patient (ordonnances, factures, prescriptions, scans, images etc.)
- *Les cas métiers de la gestion des séances* : La gestion de base des séances faisait partie du sprint 1. Dans ce sprint ce sont les cas particuliers (i.e. les cas de figure issus du terrain) qui sont traités. Ces cas de figure couvrent par exemple l'annulation ou le décalage d'une séance par le praticien ou le client.

- *L'intégration continue et le TDD* : Ce n'est pas une fonctionnalité à proprement parlé, mais plus une amélioration des méthodes de production. L'intégration continue doit permettre une industrialisation de la production (tests lancés automatiquement lors du build et obtention de rapports portants sur la qualité du code). La méthodologie de TDD[1] porte sur la façon de développer et tester le logiciel. Elle doit permettre d'augmenter la qualité du code développé.

3.1.4 Sprint 3

Ce troisième sprint doit commencer à partir du 11 Avril. Les détails le concernant n'ont pour l'instant pas été abordés avec précision. Cependant, il est prévu que durant cette période, nous travaillions en collaboration avec des ergonomes et des designers afin d'améliorer l'expérience utilisateur.

3.2 Travail à réaliser

La partie qui m'a été attribuée durant le sprint 2 est la mise en place du dossier médical du patient. Il s'agit d'une fonctionnalité du logiciel qui dépend des autres parties déjà développées (cabinets, praticiens, patients), mais qui est relativement indépendante.

Il s'agit d'une fonctionnalité présente dans *Topaze Maestro*, qu'il faut créer dans *Topaze Web*. Une capture d'écran de la fonctionnalité du dossier médical de Topaze Maestro est visible en annexe 11.

3.2.1 Les fonctionnalités du dossier médical dans Topaze Maestro

La liste des documents associés au patient

La fonctionnalité principale est la liste des documents du patient. Cette liste contient différents types de documents : les ordonnances, factures, paiements, suivis, scans, courriers (textes), images, sons, objets OLE*. La liste peut être ordonnée selon le type de documents ou selon la date. Elle peut également être filtrée sur le type de documents.

Le récapitulatif de chaque document

Un panneau latéral permet à l'utilisateur d'avoir accès rapide aux informations du document qui est sélectionné dans la liste.

La lecture ou la suppression de documents

Chaque document de la liste doit pouvoir être ouvert ou supprimé. L'ouverture d'un document est différente selon le type de celui-ci. Si c'est une ordonnance ou un document issu d'un regroupement d'information, l'ouverture provoquera l'affichage d'un nouvel écran. Si c'est une image, un pdf, ou un objet OLE, c'est un logiciel Windows qui s'ouvre. Enfin, si c'est un document *richtext* (cas des courriers et prescriptions), un éditeur propre à Topaze permet l'édition.

[1] TDD: Test-Driven Development (Développement dirigé par les tests).

Les modèles d'images

Un onglet du menu latéral permet l'insertion d'une image à partir d'une librairie de modèles prédéfinis.

Les documents textuels (RTF*)

L'utilisateur peut ajouter et éditer des documents textuels grâce à l'éditeur de texte intégré à Topaze. L'éditeur proposé est agrémenté de différentes fonctionnalités, telles que l'ajout d'images à partir d'une bibliothèque, l'utilisation de modèles de document et l'emploi d'emplacements paramétrés (placeholders). Une capture d'écran de l'éditeur de texte de Topaze Maestro est disponible en annexe 12.

La bibliothèque d'images

L'utilisateur peut insérer dans le document texte une image provenant de son ordinateur ou de la bibliothèque d'images. La bibliothèque d'image présente plusieurs onglets thématiques et chaque onglet contient une liste d'images prédéfinies.

Les "placeholders"

Les emplacements paramétrés (placeholders) sont des balises génériques qui peuvent être introduites dans un document texte et qui seront remplacées à la sauvegarde par leur valeur réelle (valeur en base de donnée).

Le mécanisme fonctionne en plusieurs temps :
D'abord, l'utilisateur sélectionne dans une arborescence l'information dont il a besoin (par exemple: la date d'une ordonnance) et le logiciel insère dans le texte la balise correspondante (dans notre cas : [*ordonnance.numero_securité_sociale*]).
Ensuite, l'utilisateur remplit un contexte. Dans notre cas, le praticien aurait à choisir l'ordonnance ciblée.
Enfin, lorsque l'utilisateur clique sur "prévisualiser" ou "sauvegarder", l'emplacement est remplacé par sa valeur réelle.

Les modèles de texte

Les modèles de texte sont des document RTF contenant des placeholders, qui ont vocation à être réutilisés.
Ces modèles sont accessibles via une bibliothèque similaire à celle des images. Lorsque l'utilisateur ouvre un modèle, il peut le modifier et l'enregistrer comme un document normal.

L'ajout de scans

L'utilisateur a la possibilité de numériser un document dirèctement en cliquant sur un bouton du menu latéral. Il peut également insérer un document déja scanné, via le même bouton.

L'ajout d'objets OLE

Le protocole OLE (Object Linking and Embedding) est un protocole mis au point par microsoft permettant la liaison et l'incorporation d'objets. Cela permet à différents logiciels

de se transmettre des objets.
Dans topaze, l'utilisateur peut donc insérer tout objet/document qui implémente l'interface *IOleObject*. Concrètement, il peut donc insérer un document word, une image paint, une vidéo etc. et lorsqu'il cliquera sur "voir", le logiciel approprié de windows s'ouvrira pour lui permettre de visualiser et/ou éditer le document. Cette fonctionnalité est trés puissante car elle permet le support d'un trés grand nombre de documents.

L'enregistrement de sons

L'utilisateur peut enregistrer un son en cliquant sur un bouton du menu. Cela lui ouvre un logiciel d'enregistrement de sons de Windows.

3.3 Les objectifs précis attendus

Les objectifs attendus lors du sprint 2 :

- L'accès au dossier médical du patient.
- L'édition de documents texte (sans modèle).
- L'ajout de documents numériques au dossier d'un patient.

Pour chaque objectif, les tâches suivantes sont à réaliser :

- Analyse de l'existant (fonctionnalités et comportements de Topaze Maestro)
- Etude de la faisabilité des différentes fonctionnalités.
- Discussion du besoin avec le chef de projet et/ou la maîtrise d'ouvrage.
- Proposition d'une solution qui réponde fonctionnellement au besoin.
- Spécification du modèle métier (entités de la base de données) qui réponde au besoin.
- Spécification de l'API Rest.
- Spécification des interfaces graphiques si besoin.
- Implémentation de la solution (backend + frontend) dans Topaze Web.

4 Solutions techniques

4.1 Synthèse de l'existant

4.1.1 Architecture logicielle

Architecture 3-tiers

Topaze web est bâti selon une architecture 3-tiers :

- la *couche présentation* qui sert à l'affichage des informations et qui tourne sur le navigateur du client.
- la *couche métier* qui contient la logique applicative.
- la *couche d'accés aux données* qui garantie la persistance des données (base(s) de données).

Les utilisateurs

Le projet Topaze Web aura deux types d'utilisateurs :

- *Les infirmiers, kinésithérapeutes et autres professionnels de la santé* qui utiliseront Topaze au quotidien pour la gestion des feuilles de soins, de la facturation, des dossiers patients etc.
- *Le service technique d'IDEA* (la société soeur de Supralog qui commercialise Topaze) : qu'il s'agisse des personnes chargées d'administrer les comptes utilisateurs et cabinets ou les personnes étant au SAV, ils utiliseront l'application pour gérer les abonnements et répondre aux problèmes techniques des clients finaux.

Pour résumer, voici une schéma de l'architecture 3-tiers de Topaze Web :

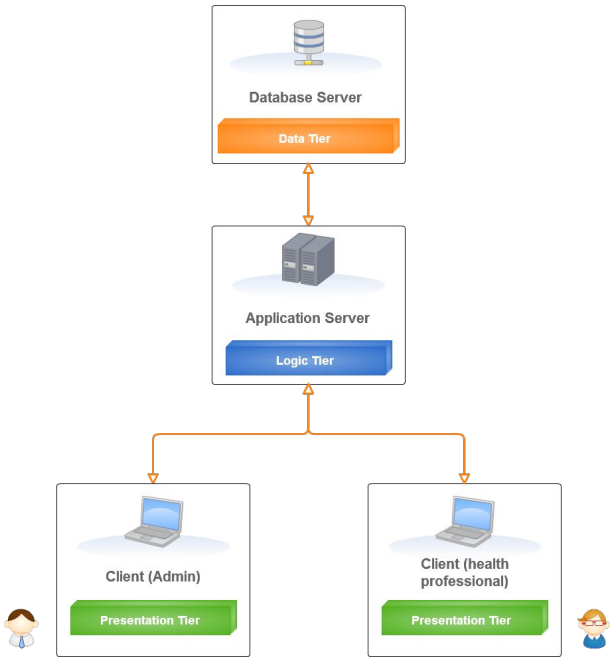

Figure 2: Schéma de l'architecture 3-tiers (créé par Tom Veniat, utilisé avec son accord)

Afin de satisfaire les besoins de ces deux types d'utilisateurs, deux applications qui accèdent aux mêmes données ont été crées:

- *Topaze* : l'application utilisée par les professionnels de la santé.
- *Opale* : l'application utilisée par les administrateurs.

L'application Topaze

L'application Topaze a été scindée en deux parties : *Topaze REST* et *Topaze Webapp*.
Topaze Rest est un serveur REST, à accès sécurisé par token. Il réalise les opérations liées à la logique métier et il fait persister les données en base.
Topaze Webapp gère la mise en forme des données avant affichage. Son accès est sécurisé via une authentification par sessions. Elle contacte topaze REST afin de rapatrier les données, puis elle les met en forme avant de les envoyer au client (le navigateur).

L'application Opale

Opale et Topaze ont une base de données en commun. De cette manière, un administrateur d'Opale peut ajouter un utilisateur qui sera ensuite utilisable dans Topaze.
Les deux applications sont séparées car elles sont utilisées par des utilisateurs différents, mais également parce qu'elles effectuent des tâches de nature différente. En effet, les tâches réalisées par les administrateurs, dans Opale, se résument à des opérations CRUD [2] sur la base de données. Dans Opale, il n'y a donc pas ou peu de logique métier. De ce fait, le code de l'application Opale est presque entièrement générée au moyen du *Skeleton Generator*.

Le *Skeleton Generator* est un projet open-source, développé par le chef de projet, qui permet de faire du M2C [3]. Dans opale, on décrit donc les modèles de données via un fichier XML (cf annexe 13), puis le générateur est utilisé pour créer les entités en base de données, générer le code des DAO [4] qui serviront pour Opale et Topaze et générer tout le code d'Opale (les composants-métier, les objets métiers, les services, l'api rest, ainsi que l'application web).

L'application Sesame

La manipulation des Feuilles de Soins Electroniques (FSE) et la facturation nécessitent la lecture d'informations présentes sur la carte vitale du patient. D'autre part, les logiciels utilisant les FSE sont soumis à l'obtention d'un agrément délivré par les représentants des organismes d'Assurance Maladie. Cet agrément est obtenu après de nombreux tests vérifiant que le logiciel en question respecte un cahier des charges très précis.
La gestion de la facturation et des FSE étant nécessaire à l'application, mais pouvant être externalisée, c'est une troisième application qui en a la responsabilité.
Cette troisième application nommée *Sesame* répond à trois problématiques :

- La communication avec le lecteur de cartes vitales.
- La facturation des actes réalisés par les praticiens, en fonction des informations du patient présentes sur la carte vitale.

[2] CRUD: Create, Read, Update, Delete
[3] M2C: Modèle To Code
[4] DAO: Data Access Object

- La communication avec Topaze et la restriction des accés à la carte vitale.

Afin de pouvoir communiquer avec la carte vitale un driver doit être installé sur la machine de l'utilisateur. Par la suite, le serveur Sesame communique avec le driver via des websockets afin d'avoir accès aux informations de la carte vitale.

Schéma d'architecture global

Voici le schéma d'architecture globale représentant l'architecture 3-Tiers et la communication entre les différentes applications :

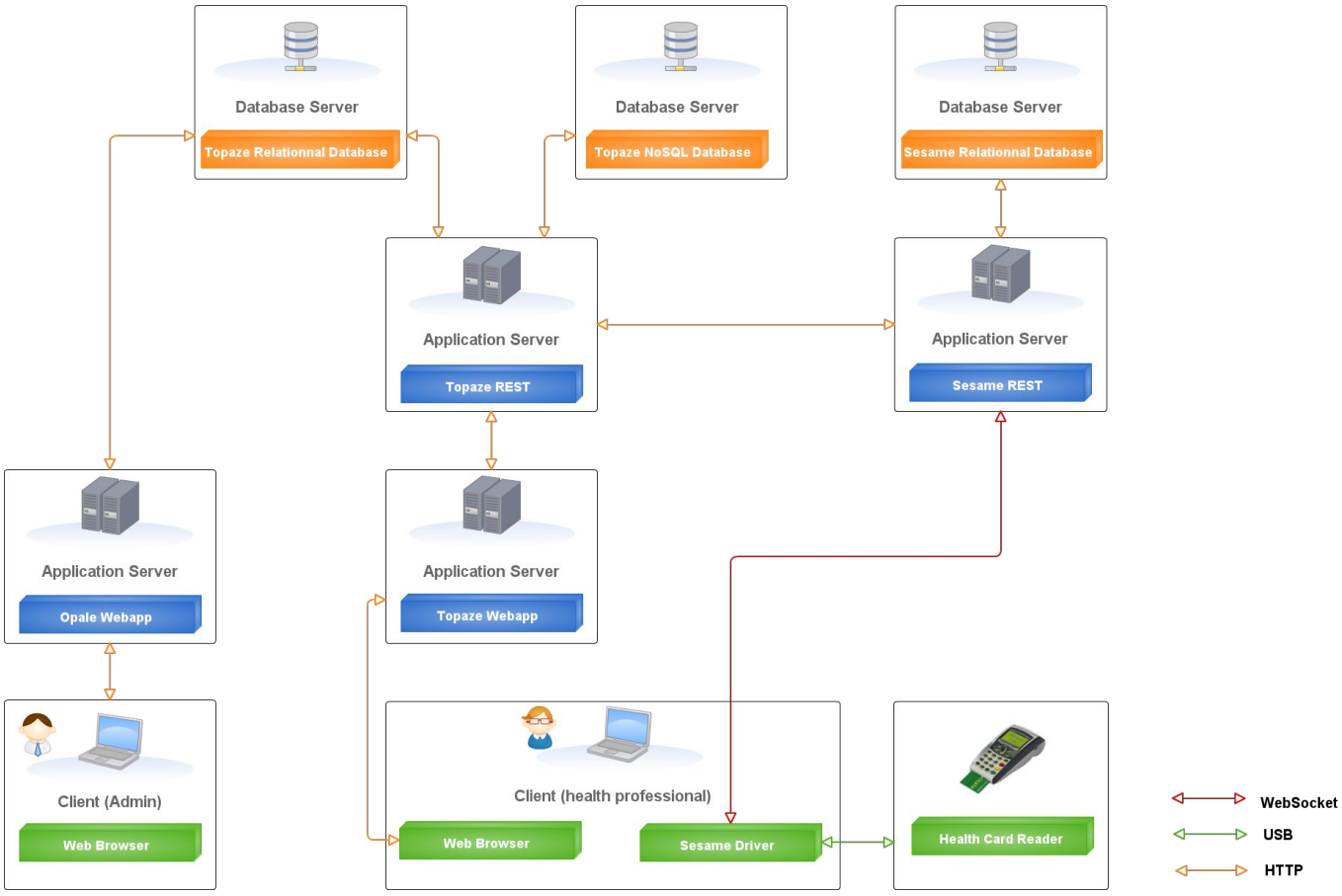

Figure 3: Schéma d'architecture global (créé par Tom Veniat, utilisé avec son accord)

4.1.2 Technologies utilisées

Les technologies utilisées par le projet sont :

- Bases de données : PostgreSQL pour les modèles de données, MongoDB et GridFS pour les fichiers.
- ORM : Hibernate
- Serveur : Tomcat
- Backend : Java et Spring

- Front-End : JSF et Primefaces

4.2 Solution envisagée

4.2.1 Conception et implémentation du dossier médical

Analyse de l'existant

Le dossier médical de Topaze Web doit, si possible, apporter les mêmes fonctionnalités que celui présent dans Topaze Maestro. La première étape de conception a donc consisté à faire une analyse de l'existant et à lister les fonctionnalités présentes (il s'agit de la liste décrite dans la section Problématique).

Etude de la faisabilité

Ensuite, j'ai étudié la faisabilité des différentes fonctionnalités et les ai triées en fonction de leur difficulté à être implémentées en web. La plupart des fonctionnalités peuvent être créées sans difficulté, cependant certaines ont soulevé des questions :

Le cas des objets OLE

est particulier, car il s'agit là d'un protocole de Microsoft à destination de Windows. Il ne peut donc pas être utilisé dans une solution web. À ce niveau, il était donc nécessaire de savoir quels types d'objets devaient être supportés et si la fonctionnalité était vraiment importante. En effet, chaque objet à supporter nécessite de batir une solution sur mesure.
Après discussion avec le chef de projet, il a été décidé que l'utilisation des objets OLE ne serait pas géré à proprement parlé dans l'application. Cependant, l'utilisateur peut, s'il le désire, télécharger différents types de fichiers sur l'application et les récupérer ensuite pour les éditer avec ses propres logiciels.

Le cas des documents RichText

a lui aussi demandé un traitement spécifique. Dans Topaze Maestro, les documents au format richtext peuvent être téléchargés puis édités directement depuis l'application. En web, l'édition de documents au format richtext n'est pas habituelle. En revanche, de nombreux éditeurs WYSIWYG* existent et permettent d'éditer du Markdown, du BBCode ou du HTML. J'ai donc choisi d'intégrer un composant extérieur afin de remplir cette tâche. Afin que les utilisateurs de Topaze Maestro puissent réutiliser leurs données dans Topaze Web, il a été décidé que les fichiers au format RTF seraient transformés en HTML au moyen d'un traitement batch en cas de migration de donnée.
Les choix techniques et l'intégration du composant seront décrits dans la section 4.2.2.

Le cas des sons

est particulier. Pendant longtemps, cette fonctionnalité a été implémentée en web en utilisant du flash. Aujourd'hui, la balise audio d'HTML5 permet de lire des sons de façon native dans les navigateurs. Cette fonctionnalité d'HTML5 est supporté par quasiment tous les navigateurs, sauf IE8 [5].
En ce qui concerne l'enregistrement des sons (possible dans Topaze Maestro via un objet

[5]Support de la balise audio : http://caniuse.com/#feat=audio

OLE), HTML5 offre l'API getUserMedia qui permet d'accéder au flux audio ou vidéo d'un périphérique externe. Cependant, le support de cette fonctionnalité n'est pas encore assuré par tous les navigateurs [6]. Afin de résoudre la question du support des navigateurs, des bibliothèques open-source telles que *getUserMedia.js*[7] ont été développées. Ces bibliothèques utilisent la getUserMedia API si elle est disponible et utilisent Flash si ce n'est pas le cas.

Elaboration d'une solution

Afin d'élaborer une solution, j'ai commencé par réaliser un diagramme de classe d'analyse modélisant les différents objets qui interviennent dans le dossier médical:

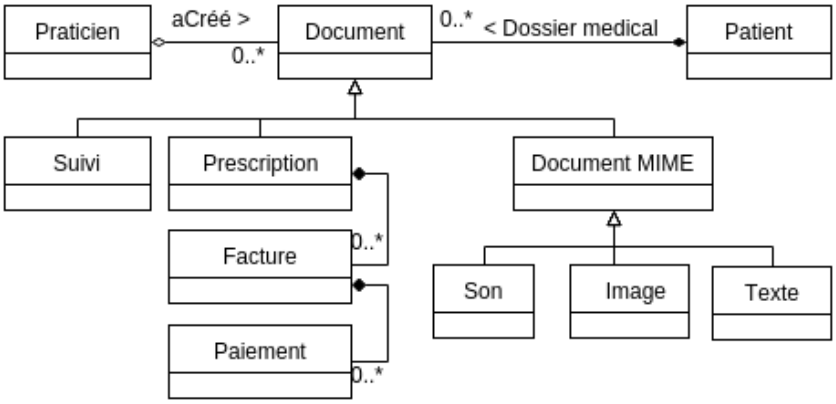

Figure 4: Diagramme de classe d'analyse du dossier médical.

Dans ce diagramme d'analyse, on peut voir que le dossier médical est constitué de deux types de documents : ceux que j'ai appelé les "Documents MIME" qui correspondent à des fichiers binaires (sons, images, textes) et les autres, qui sont des objets métiers manipulés dans l'application.

Une fois ce diagramme réalisé s'est posée la question de la persistance des données. Avant toute chose, j'ai regardé le contenu de la base de données afin de savoir quelles entités étaient déjà présentes.
Voici le modèle conceptuel de la base de données au départ (seules les entités concernant le cas étudié ont été représentées) :

[6] Support de l'API getUserMedia : http://caniuse.com/#feat=stream
[7] getUserMedia.js : https://github.com/addyosmani/getUserMedia.js

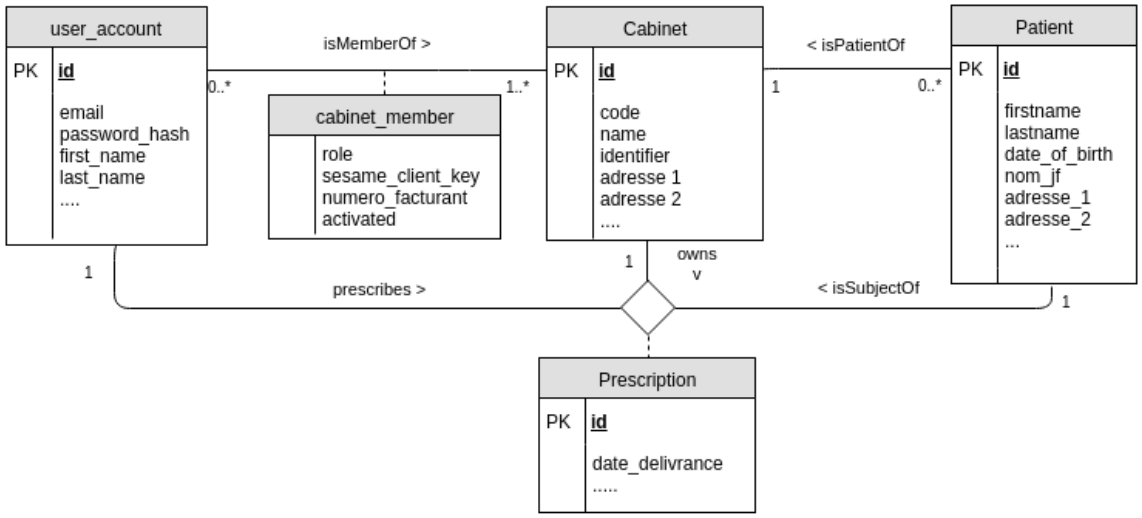

Figure 5: Modèle UML conceptuel de la base de donnée, au départ.

Dans la base de donnée, les prescriptions étaient déja présentes. Il restait donc à ajouter les documents issus de fichiers.
Voici donc l'ajout que j'ai réalisé au modèle de donnée :

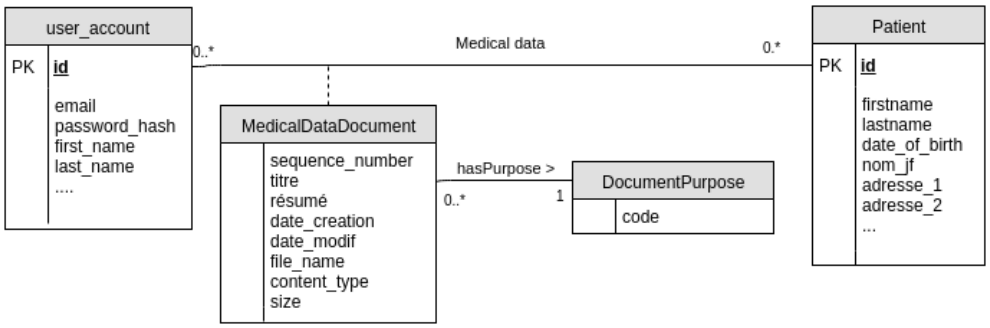

Figure 6: Modèle UML illustrant les ajouts à la base de données.

Ensuite, s'est posée la question du stockage du contenu des fichiers. Deux choix étaient possibles : les stocker en base, ou directement dans un système de fichier. Cette question s'était déjà présentée plus tôt dans le projet, et l'équipe avait décidé de stocker les données binaires dans une base de données Mongo DB munie de GridFS. Ce choix a été fait pour les caractéristiques de disponibilité et de scalabilité propre à MongoDB.
La base de données PostgreSQL est donc utilisée pour stocker les méta données liées au fichier et leur contenu est stocké dans MongoDB. Lorsqu'on a besoin du contenu d'un fichier, on récupère son id dans Postgre et l'on fait une requête dans MongoDB pour obtenir son contenu. La convention choisie est que le filename dans Mongo correspond à l'id du document dans Postgre.

Dans MongoDB, deux collections sont utilisées, par défaut, par GridFS pour stocker les fichiers :

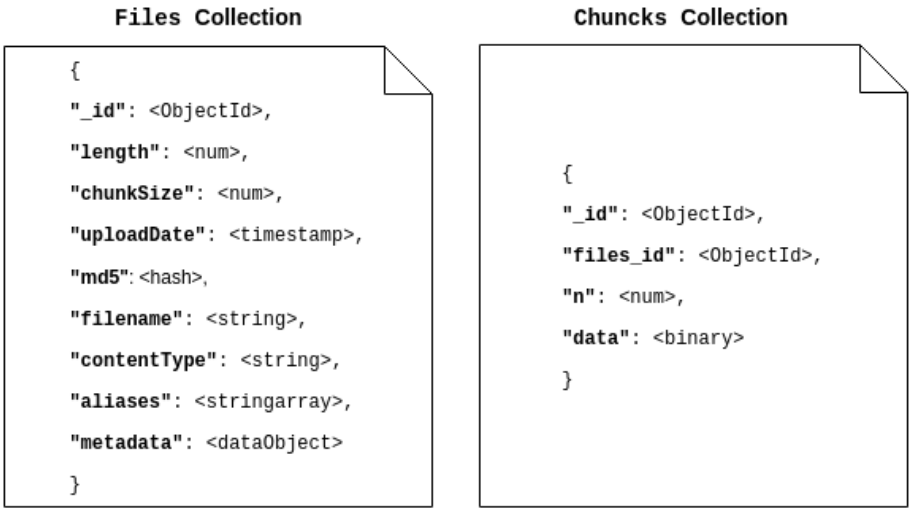

Figure 7: Collections utilisées par GridFS pour le stockage des fichiers.

La collection *Chuncks* sert à stocker les données du fichier et la collection *Files* sert à stocker les méta-données du fichier.

Phase d'implémentation

Une fois le modèle de données mis en place, il ne reste plus qu'à réaliser l'implémentation dans Topaze Rest et *Topaze Webapp*. À ce niveau, différentes classes sont à implémenter dans la *Topaze Webapp* et dans *Topaze Rest*. Afin de comprendre quelles classes sont à implémenter, voici un schéma illustrant la requête permettant d'obtenir le dossier médical (le diagramme est également fourni en plus grand, en annexe 14, pour les lecteurs sur papier):

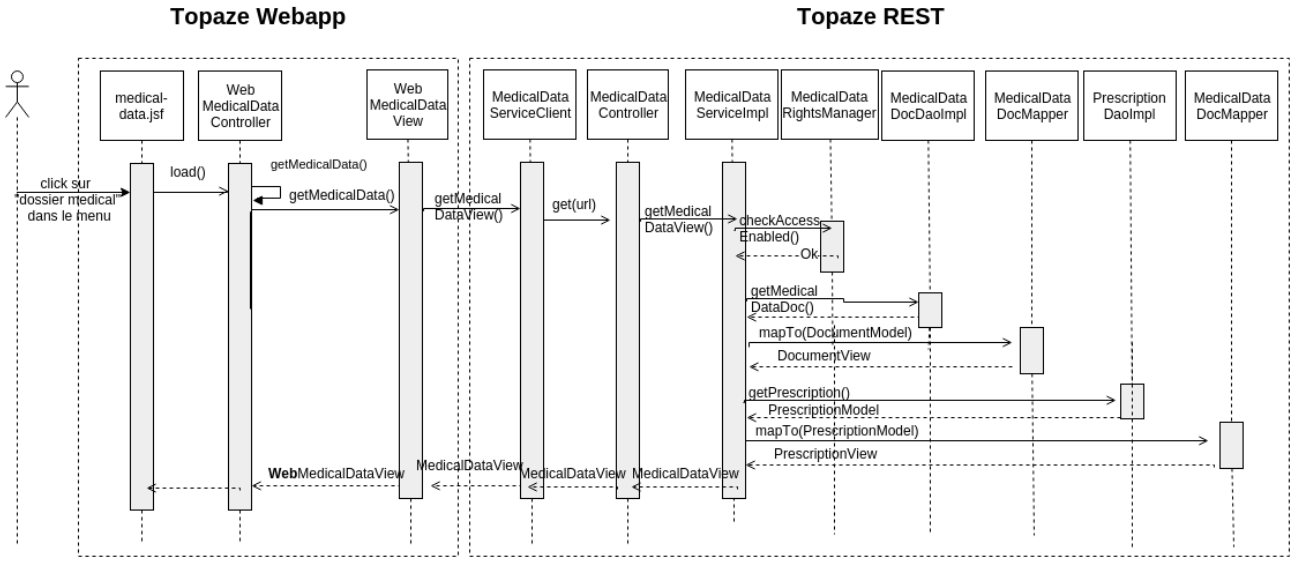

Figure 8: Diagramme de séquence modélisant la requête du dossier médical.

Comme on peut le voir, l'architecture de la *webapp* suit un modèle MVC. Du côté de *Topaze Rest*, il y a plusieurs packages :

- *API* qui définit les services et les modèles (qui correspondent à des vues simplifiés des objets tirés de la base)
- *REST* qui définit les les contrôleurs (qui font le mapping entre une url et un service)
- *REST Client* qui définit des services clients utilisés par la web-app pour contacter les contrôleurs.
- *Business components* qui définit les objets tels que les mappers qui permettent de créer des vues ad-hocs à partir des modèles issus de la base de données.

Enfin, on utilise également les DAO [8] définis dans Opale qui permettent d'obtenir les informations de la base de données.

Une capture d'écran du dossier médical de Topaze Web est disponible en annexe 15.

4.2.2 Choix et intégration de l'éditeur de texte

L'édition de texte est une fonctionnalité importante de Topaze Maestro. Il fallait donc qu'un éditeur soit implémenté ou intégré. De nombreux éditeurs de texte de qualité ont déjà étés développés et il en existe sous toutes les licences. Il a donc été choisi d'intégrer un composant existant. Cela permet d'avoir un composant robuste, bien documenté, éprouvé par toute une communauté et en continuelle évolution.

Les critères que j'ai pris en compte lors du choix étaient la notoriété du projet, le nombre de contributeurs et d'utilisateurs, son ancienneté, le support proposé, le nombre de fonctionnalités offertes, les possibilités de configuration, la possibilité de greffer de nouveaux plugins et enfin, la licence.

La licence était une question importante car Topaze Web est destiné à être commercialisé. Il n'était donc pas possible de choisir un éditeur distribué sous une licence à fort copyleft (par exemple la GPL). Un tel choix aurait contraint Topaze Web à être lui-même distribué sous licence GPL.

Pour finir, l'éditeur CKEditor a été choisi car il propose une licence LGPL, dispose d'une bonne communauté, de nombreux plugins et de la possibilité d'ajouter facilement ses propres plugins.

Intégration de l'éditeur

L'intégration se fait très facilement en ajoutant la librairie de l'éditeur. Il ne reste plus ensuite qu'à gérer le flot de données qui circulent entre l'éditeur et l'application. Du javascript est utilisé pour gérer la sauvegarde lors du click sur le bouton ou périodiquement si l'utilisateur

[8] DAO: Data Access Object

ne sauvegarde pas.

Par la suite d'autres modifications seront apportées (vue réaliste faisant apparaître les pages telles qu'elles seront imprimées, galerie d'image, import d'images).

Une capture d'écran de l'éditeur de texte de Topaze Web est disponible en annexe 16.

4.2.3 La bibliothèque d'images

Dans Topaze Maestro, la bibliothèque d'images contient uniquement des images prédéfinies. Dans la version web, il a été décidé qu'elle puisse également contenir des images importées par l'utilisateur.
De manière à ce que la bibliothèque soit simple à gérer, nous avons décidé (le chef de projet et moi) de créer un gestionnaire de fichiers avec une arborescence de dossiers.

Elaboration du modèle de données Lors de la conception du modèle, la contrainte suivante devait être prise en compte :
Lorsqu'un utilisateur supprime une image de sa bibliothèque, si celle-ci a été insérée dans un document, elle doit continuer d'y appraître.
Ainsi, il était nécessaire de séparer en base, l'apparition d'une image dans la bibliothèque de son stockage pour utilisation dans les documents texte.
J'ai donc créé le modèle suivant :

Spécification de l'API REST En ce qui concerne l'API REST, elle contient les opérations de CRUD[9] sur les dossiers et leur contenu. L'API expose donc les urls suivantes concernant les dossiers :

- GET /account/models/directory/directoryId
- DELETE /account/models/directory/directoryId
- POST /account/models/directory

Concernant les images contenues, ont trouve :

- GET /account/models/directory/directoryId/picutre/pictureId/content
- DELETE /account/models/directory/directoryId/picutre/pictureId
- POST /account/models/directory/picutre

[9]CRUD: Create Read Update Delete

Modèle de l'API L'API REST récupère les donneés de la base au moyen de DAOs[10] et peuple des objets métiers qu'elle va ensuite retourner.

Pour la base de donnée, il était plus simple de stocker l'arborescence de façon ascendante (chaque fichier contient l'id du dossier parent). Mais, concernant les objets métiers (qui eux seront exposés), on souhaite que l'arborescence soit stockée de façon descendante, pour que leur usage soit naturel. En effet, les objets métiers sont ensuite utilisés pour générer les vues en HTML. Dans ce cas, on a donc en général accés à un dossier et on souhaite obtenir la liste de ses fils.

Pour représenter ce type de modèle, on pense naturellement au design pattern composite :

Cependant, les objets retournés par l'API REST seront transmis par le réseau et devront donc être sérialisés. Hors, l'outil utilisé pour la sérialisation (Jackson), ne gère pas nativement le polymorphisme de java.
Le moyen utilisé pour pouvoir le faire quand même consiste à utiliser des annotations Jacksons, ou à inclure un champ "type" dans le json retourné par l'API. Cependant, ce choix est couteux car il impose des contraintes fortes sur le client utilisant l'API: l'usage de Jackson ou l'ajout d'un mécanisme particulier pour parser les objets reçus.
N'ayant que deux types d'objets, j'ai donc préféré briser l'héritage :

Implémentation de la partie WebApp La webapp effectue des traitements assez standards. Elle effectue des appels aux services REST pour obtenir les dossiers et leur contenu et expose un certain nombre de méthodes utilisables par les vues JSF grâce un controlleur. Le modèle utilisé par la webapp est rudimentaire car il ne fait qu'encapsuler les objets métiers obtenus de l'API REST et stocke en plus quelques variables d'états pour la vue.

Spécification de l'interface graphique Avant d'implémenter la vue en JSF dans l'application, j'ai réalisé un prototype en HTML et javascript dans plunker. Plunker est un éditeur en ligne pour le front-end. Il permet de faciliter la programmation fron-end en proposant un aperçu en temps réel, une inclusion de librairie simple, une gestion de version, un stockage privé du code et un partage simple et rapide du résultat avec un tiers.
Une fois que le résultat était satisfaisant, il a suffit de convertir une partie du HTML en JSF pour l'intégrer au code.

Création d'un composant réutilisable Le gestionnaire de fichier utilisé pour la bibliothèque d'image a vocation à pouvoir être réutilisé à d'autres fin. J'ai donc créé un composant réutilisable.
En JSF, trois choix sont possibles pour créer des composants réutilisables :

- Les *Custom Tags* : il s'agit de la solution la plus légère proposée par JSF. Elle permet d'externaliser dans un fichier une portion de code JSF afin de pouvoir la réutiliser ensuite. Un tag peut accepter des paramètres afin de personnaliser le comportement du composant. Les tags peuvent enuite être utilisés en JSF au moyen d'une balise.

[10] DAO: Data Access Object

- Les *Composite Components* : cette solution est trés semblable aux tags, mais elle est plus coûteuse en temps de traitement. Elle a l'intérêt de présenter un passage de paramêtres plus propre que les tags.
- Les *Custom Components* : ce sont des composants définis en java (et non plus en JSF). Pour en créer, il faut définir une classe qui assurera le rendering du composant et une autre ....

Pour le gestionnaire de fichiers, mon choix s'est porté sur les tags, car la solution est performante et s'avère suffisante pour traiter le besoin.
Ici, l'usage d'un custom components aurait été disproportionné car coûteux et non nécessaire (un rendering en Java n'est pas nécessaire si il peut être fait en JSF).
Les composite components n'étaient pas non plus nécessaires et auraient apporté plus de lourdeur.

4.2.4 Les "placeholders"

Dans l'éditeur de texte de Topaze Maestro, une fonctionnalité permet d'utiliser des emplacements paramétrés (placeholders) qui sont remplacés à la sauvegarde par leur valeur réelle (en base de donnée).
Cette fonctionnalité est trés puissante car elle permet à l'utilisateur d'utiliser directement une partie des données présentes en base, dans ses documents textes.

Faisabilité La faisabilité de la foncionnalité reposait sur trois éléments : La possiblité d'insérer des balises html complexes dans ckeditor. La possibilité de créer un lien entre une balise du texte et une donnée en base. La possibilité de faire le remplacement de la balise par sa valeur.

La solution mise en place

L'insertion de placeholders dans l'éditeur

Le premier pas pour réaliser la fonctionnalité était d'ajouter un plugin à l'éditeur pour insérer les placeholders.
Dans l'éditeur, les placeholders doivent avoir un formattage spécifique pour être reconnaissables, ne pas pouvoir être éditables et le html doit éventuellement pouvoir encapsuler des données (attributs *data-** de HTML).

Le plugin a été réalisé de la même manière que pour la bibliothèque d'image : en ajoutant un fichier *plugin.js* et en utilisant une modale bootstrap.
En ce qui concerne les balises : le CSS permet de leur donner l'allure souhaitée, l'API de CKEditor permet d'insérer du HTML dans le texte à l'endroit du curseur et la liste blanche des éléments acceptés par l'éditeur est modifiable via un fichier de configuration.

Le lien entre les données vues par l'utilisateur et les données en base

Lorsque l'utilisateur souhaite insérer un placeholder, il doit d'abord le sélectionner dans une arborescence. Lors du click sur une feuille de l'arborescence, le placeholder est inséré.

Lors de la manipulation, l'utilisateur voit donc deux types d'informations :

- les noeuds de l'arborescence.
- les placeholders.

Pour pouvoir construire l'arborescence et effectuer le remplacement des placeholders par leur valeur, les informations visibles par l'utilisateur doivent être reliées à des objets métiers de l'API. Dans ce but, deux solutions étaient disponibles :

- utiliser des tables en base de données.
- socker l'information dans les classes java.

En JSF, on est déjà capable d'insérer dans les vues des informations provenant du modèle grâce aux EL [11]. Par exemple, l'expression $\#\{prescriptionView.date\}$ sera remplacée par sa valeur par JSF, lors de la génération du HTML. Lors du remplacement d'un placeholder par sa valeur, on doit donc faire le lien entre cette notation pointée et le nom du placeholder.

La première approche conciste à ré-utiliser la table directory [12] pour les dossiers de l'arborescence et à ajouter une nouvelle table pour les feuilles. Cette deuxième table sert à faire le lien entre les noms des placeholders, la notation pointée de java et le libellé à utiliser dans l'arborescence:

La deuxième approche vise à utiliser les capacités de Java pour stocker l'information nécessaire directement dans le code.
Pour créer l'arborescence, il est possible d'utiliser la fonctionnalité d'introspection de java. L'introspection permet de prendre une classe et de parcourir ses propriétés et méthodes. Avec cette technique, on peut parcourir les objets métiers à exposer par les placeholders pour remplir un arbre (qui sera utilisé pour construire l'arborescence dans la vue).
Pour faire le lien entre les propriétés des objets métiers, les libellés de l'arbre et le nom des placeholders, on ajoute des annotations sur les propriétés des classes.

Pour comparer les deux solutions : en terme de cout, dans la première approche on a un accés direct à l'information, mais celle-ci requiert un accés à la base de données et la reconstruction de l'arbre. Dans la deuxième approche, l'accés à l'information n'est pas direct, il faut parcourir les classes (couteux) et également reconstruire l'arborescence, mais on évite un accés à la base de données.
En ce qui concerne la maintenabilité, la deuxième solution est meilleure, surtout pour le lien entre le placeholder et la notation pointée.
Finalement, c'est donc la deuxième solution qui a été retenue et que j'ai implémentée.

[11] EL: Expression Language
[12] la table directory est celle introduite pour gérer l'arborescence du gestionnaire de fichier

Le remplacement des placeholders par leur valeur

Le remplacement des placeholders par leur valeur a lieu lors de la sauvegarde du document, ou lorsque l'utilisateur demande une prévisualisation.

Trouver des expressions tokenisées dans un texte et les remplacer par leur valeur est une opération courante et de nombreuses bibliothèques permettent de le faire (velocity, ... Ayant déjà la bibliothèque Velocity sur le projet, c'est la librairie qui a été utilisée.

Le remplacement des placeholders se fait en 3 phases :

La phase 1 : elle sert à remplacer les noms des placeholders par la notation pointée.
Cette phase se déroule en deux étapes :
D'abord, les classes java correspondant aux placeholders utilisés dans le texte sont parcourues pour remplir un contexte. Le contexte consiste en un dictionnaire de clés-valeurs, avec en clé le nom du placeholder et en valeur, la notation pointée.
Ensuite, Velocity se sert du contexte pour remplacer les noms des placeholders par leur valeur.

La phase 2 : elle sert à remplacer les notations pointées par leur valeur réelle.
Durant cette phase, on fait appel à l'API REST pour récupérer les objets métiers nécessaires, remplis avec les informations de la base de données.
Avec les objets métiers on remplit un nouveau contexte clé-valeur, avec en clé le nom de l'objet (première partie de la notation pointée) et en valeur l'objet lui même.
Ensuite, on refait appel à Velocity pour remplacer les notations objet par leur valeur.

La phase 3 : elle sert à remplacer les valeurs non trouvées par des points d'interrogation.
En effet, lorsque Velocity ne trouve pas une valeur, ou que la valeur est "null", il ne remplasse pas la notation pointée.
Dans un soucis de propreté et également pour signifier à l'utilisateur que le remplacement n'a pas fonctionné, on insert une série de points d'interrogation.

4.2.5 Les modèles de texte

La fonctionnalité des modèles de texte consiste à pouvoir sauvegarder dans une hiérarchie de dossiers, des documents RTF pouvant contenir des emplacements paramétrés.
Cette fonctionnalité se base sur la réutilisation de plusieurs autres :

- Le gestionnaire de fichiers et les hiérarchies de dossiers/fichiers.
- Les documents texte.
- Les placeholders.

4.2.6 Documents audio

Web RTC

Choix librairie front-end

Intégration

4.2.7 Les scans d'ordonnances

Apport au driver (devices/postes utilisateurs)

Fonctionnalité de numérisation

Traitement d'image (cropping/rotation/zoom)

4.2.8 Installeur Windows pour le driver

Ayant travaillé sur le driver python et dans l'objectif d'une commercialisation qui approche, il m'a été demandé de réaliser un installateur windows pour le driver.

Fonctionnalités de l'installateur Tout d'abord, l'installateur doit déplacer les fichiers du driver dans le bon dossier sur windows. Puis, il doit installer les dépendances et les .msi nécessaires.
Il doit également compléter le fichier de configuration du driver.
Ensuite, il doit contacter le serveur *Sésame* afin de vérifier la clé de licence et d'enregistrer le nouveau poste utilisateur.
Enfin, il doit installer le logiciel Pyxvital qui est utilisé pour les opérations de facturation.

Choix de la solution technique Pour le choix de l'installateur, la solution d'*Inno Setup* m'a été fortement recommandé par le directeur de l'entreprise. En effet, il s'agit de la solution qui est utilisée dans les dernières versions de Topaze Maestro.
Avant d'opter définitivement pour Inno Setup, j'ai vérifié que l'installateur recommandé satisfasse les exigences du projet.
Lors de l'installation, il fallait par exemple que je puisse installer des .msi[13] en mode silencieux. Il fallait aussi que je puisse ajouter des écrans supplémentaires lors de l'installation. Enfin, je devais pouvoir effectuer des requêtes HTTP vers notre serveur Sésame.

Implémentation Inno Setup est un outil qui permet de réaliser simplement des installateurs. L'outil prend en charge la création des interfaces graphiques et d'une partie des opérations standards (ex: déplacements de fichiers). Pour l'utiliser, on remplit un fichier *.iss*[14] qui permet de configurer l'installateur et d'étendre ses fonctionnalités. Le fichier .iss est structuré en sections et chaque section permet de décrire de façon déclarative la façon dont l'installateur doit se comporter. Les sections principales sont :

[13] MSI: Microsoft Installer
[14] .ISS: Inno Setup Script

- Setup : section qui décrit le nom du programme, sa version, son dossier d'installation etc.
- Dirs : les dossiers à créer.
- Files : les fichiers à déplacer (avec leur source et leur destination).
- Icons : les raccourcis à créer dans le menu démarrer ou sur le bureau.
- Ini : les variables à initialiser dans le fichier *.ini* de l'application.
- Run : le programme à lancer lorsque l'installation est finie.
- Code : la partie qui permet d'étendre inno setup au moyen de code pascal.
- d'autres sections moins importantes : types (les types d'installations), components (des groupes de tâches), tasks, languages, installDelete, uninstallDelete, uninstallRun, registry etc.

Dans l'installateur, la partie la plus importante est la section *"Code"* car elle permet d'ajouter des écrans à l'installateur et de créer de nouvelles fonctionnalités. Sur le projet, je l'ai utilisé pour créer une page de validation de clé de licence, et une page servant à définir le nom du nouveau poste utilisateur.
Dans ces deux cas, l'installateur réalise une requête HTTP vers le serveur Sésame, puis traite la réponse obtenue.

L'intégration de pyxvital Pyxvital est un logiciel qui sert à créer des FSE[15] et à assurer la transmission avec le lecteur Sésam-Vitale. Il est utilisé temporairement afin de faciliter la certification de Topaze Web et accélerer ainsi sa mise sur le marché.
Le driver de Topaze Web utilise Pyxvital et il est donc nécessaire que celui-ci soit installé en même temps que les autres dépendances. Le problème de Pyxvital est qu'il ne peut être installé qu'au moyen d'un *.exe*. Hors, nous souhaitons que son installation soit transparante pour l'utilisateur. Il a donc été nécessaire de comprendre les différentes opérations réalisées lors de l'installation, afin de les inclure dans notre installateur.
Pour ce faire, j'ai donc du lancer l'installateur de Pyxvital et regarder si des registres étaient modifiés, quels fichiers étaient installés et quels fichiers de configuration étaient complétés. Ensuite, j'ai dû ajouter ces opérations à la phase finale d'installation dans Inno Setup.

4.3 Protocole d'évaluation

4.3.1 Evaluation de la qualité du code

Tests unitaires

Pour s'assurer de la qualité du logiciel, nous effectuons des tests unitaires JUnit sur les services réalisés, les composants métiers et tout ce qui contient de la logique métier. Ces tests permettent également de garantir la non-régression du code.

[15] FSE: Feuille de Soins Electroniques

Le développement dirigé par les tests

Les tests unitaires sont la plupart du temps réalisés en respectant la méthodologie du Test Driven Development. La méthode de développement dirigé par les tests a été progressivement mise en place lors du sprint 2 et permet de garantir une meilleure qualité du code. En effet, par l'utilisation de cette méthode, on est certain que chaque fonctionnalité a été testée. D'autre part, chaque test sert en quelques sortes de spécification pour le morceau de code testé.

L'intégration continue et l'inspecteur de qualité de code

Durant le sprint 2, un serveur d'intégration continue a été mis en place. Pour ce faire, un membre de l'équipe a déployé un serveur Jenkins. De ce fait, les tests unitaires sont désormais lancés à chaque build.
Par la suite, SonarQube a également été installé. Il s'agit d'un outil réalisant l'inspection de la qualité de code. Il fournit une estimation de la dette technique et pointe les problèmes rencontrés. Il fournit également une explication détaillée pour chaque problème rencontré.

Revues de code

Fréquemment, le chef de projet effectue des revues de code et peut demander à ce que certaines parties soient refactorées ou que l'architecture soit revue.

4.3.2 Evaluation fonctionnelle du logiciel

Les réunions avec le client en fin de sprint

À la fin de chaque sprint, une réunion de démonstration est organisée avec le client. C'est alors le moment d'avoir son retour sur les fonctionnalités développées. Cette réunion peut donner lieu à des rectifications à réaliser si le besoin a été mal compris ou si des évolutions sont souhaitées.

Avancement

Voici le diagramme de Gantt prévisionnel :

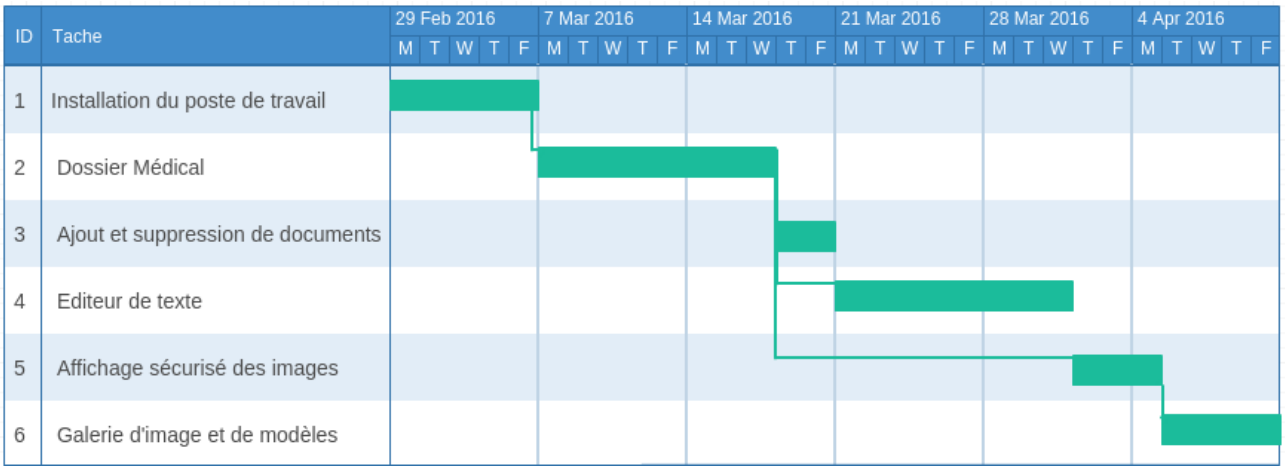

Figure 9: Gantt prévisionnel.

Et voici le diagramme de gantt réel :

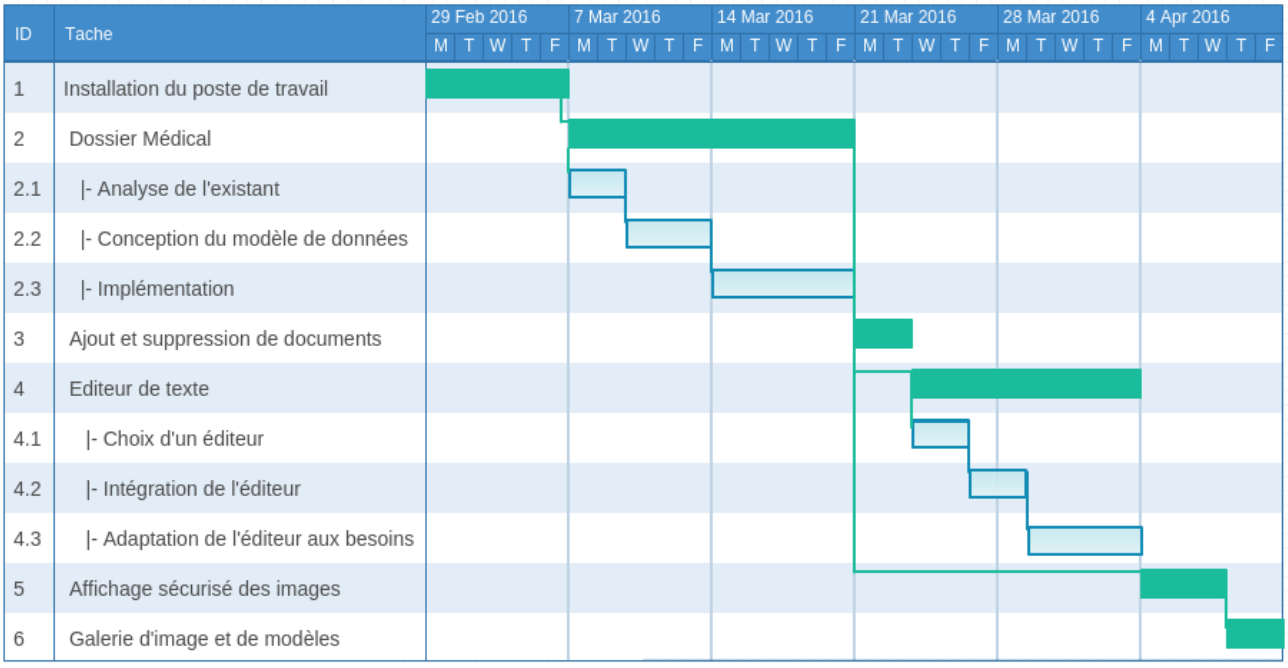

Figure 10: Gantt réel.

Impressions

4.4 Journée type et évènements.

La journée type commence à 9h avec un café, elle est ponctuée d'une pause déjeuner de 12h à 14H et se finit à 18h.
Fréquemment, des évènements sont organisés par l'entreprise, ou de manière plus informelle entre collègues. Le vendredi matin, il y a la tradition du petit déjeuner : chaque semaine un salarié ramène les croissants pour toute l'entreprise. Le vendredi midi, il arrive souvent que certains aillent au restaurant. Enfin, une fois par mois, un afterwork est organisé.

4.5 Difficultés rencontrées

L'architecture de l'application

Il m'a fallu un certain temps pour appréhender l'architecture de l'application et comprendre le flot d'exécution d'une requête. J'ai parfois eu du mal à savoir quelles classes, composants et services je devais implémenter et à quel moment chacun intervenait. Durant l'implémentation de ma première fonctionnalité, mon collègue Abdessalam a souvent été là pour me guider.

Maîtrise de nouvelles technologies

Lorsque j'ai commencé mon stage je ne connaissais pas la plupart des technologies utilisées sur le projet. L'utilisation d'hibernate, Spring, Jsf et primefaces était nouvelle pour moi. Cependant, je connaissais des technologies proches de celles utilisées : java, j2ee, jsp et j'ai donc pu faire des parallèles avec mes connaissances (issues du monde java ou autres).

Impression générale

Après ces six semaines dans l'entreprise, mes premières impressions sont trés bonnes.
Je trouve que les aspects métiers liés au projet sont intéressants, et il est motivant de savoir que le logiciel sera amené dans le futur à être utilisé par de nombreux cabinets médicaux.
Ensuite, il est appréciable de travailler sur un projet relativement récent (environ un an). Des efforts importants sont portés sur la conception et l'architecture de la solution. Le code est donc clair, maintenable et il est facile de rajouter des fonctionnalités.
Ensuite, j'apprécie les récentes démarches portant sur l'amélioration en continue des outils de production (mise en place du serveur d'intégration continue Jenkins, de l'inspecteur de qualité de code SonarQube et du gestionnaire de dossiers Nexus).
Pour finir, j'apprécie le fait de pouvoir développer mes connaissances et mes compétences en java 8, Hibernate, Spring et JSF.

En ce qui concerne l'ambiance de travail, elle est agréable. Les déjeuners entre collègues, les afterworks et autres évènements sont très appréciables.

Annexes

Dossier médical dans Topaze Maestro

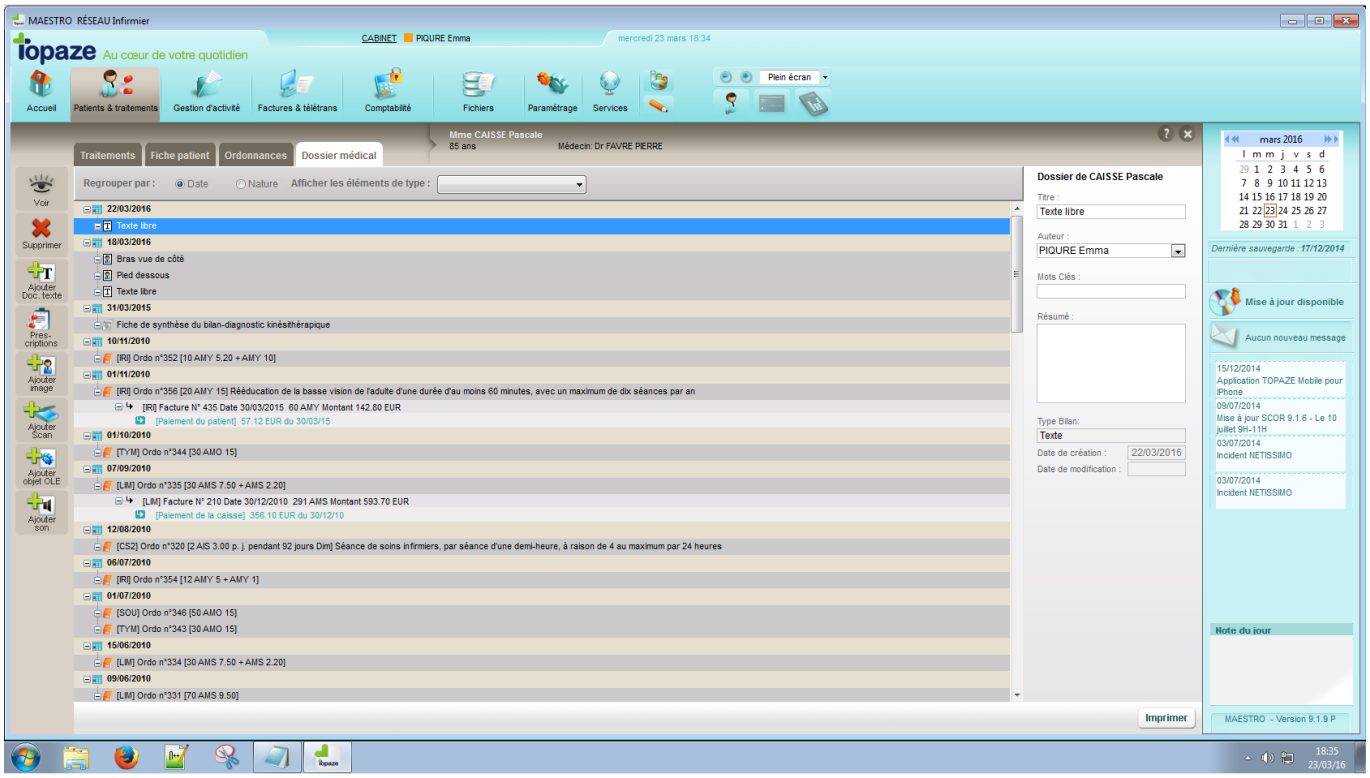

Figure 11: Dossier médical tel qu'il est présent dans Topaze Maestro.

L'editeur de texte de Topaze Maestro

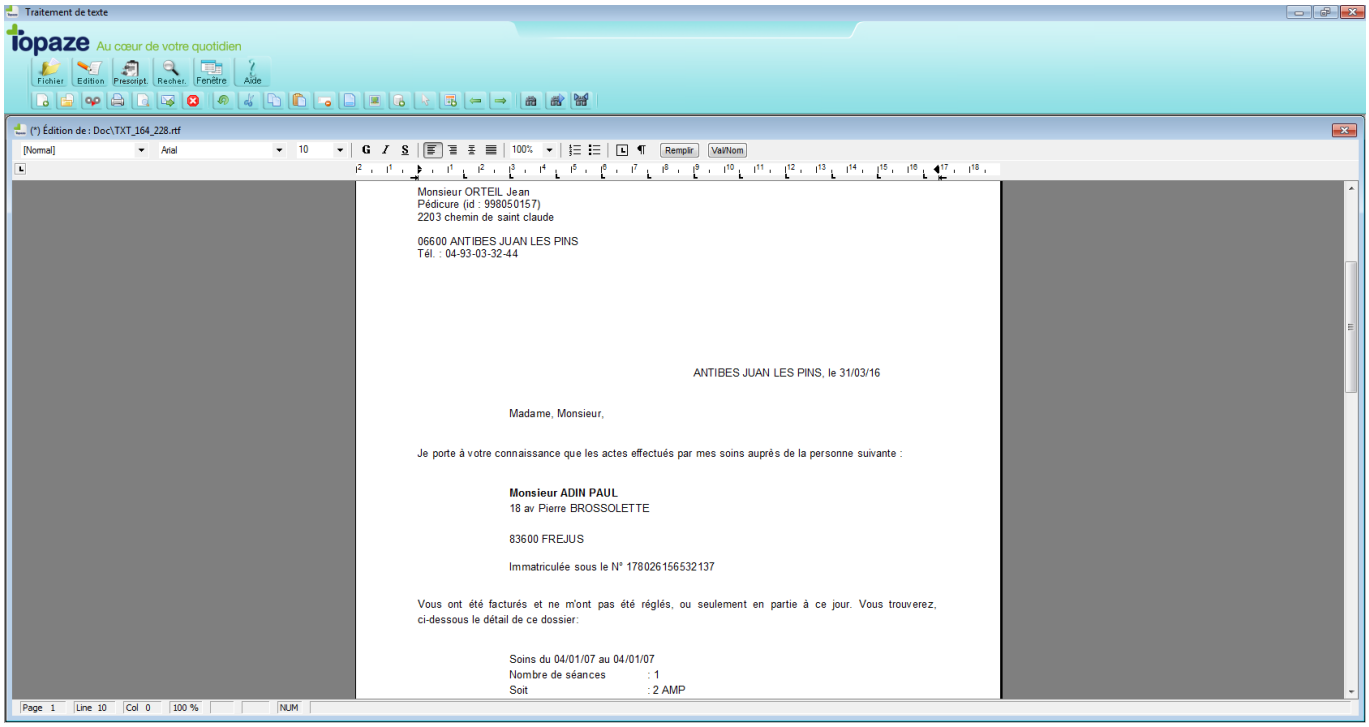

Figure 12: L'éditeur de texte présent dans Topaze Maestro.

Exemple de modèle XML utilisé par le générateur de code

```xml
<table name="DOCUMENT_PURPOSE" cardinality="1" listRendering="Types bilans" detailRendering="Type de documents" comboxable="true" createEnabled="true
    <columns>
        <column name="CODE" dataType="STRING" nullable="false" editable="true" rendering="Code">
        </column>
    </columns>
</table>
<table name="MEDICAL_DATA_DOCUMENT" cardinality="2" listRendering="Documents" detailRendering="Document" comboxable="false" createEnabled="true" upda
    <interfaces>
        <interface>AttachmentEntity</interface>
    </interfaces>
    <columns>
        <column name="PATIENT_ID" dataType="LONG" referenceTableName="PATIENT" referenceTableRelation="MANY_TO_ONE" nullable="false" editable="false"
        </column>
        <column name="SEQUENCE_NUMBER" dataType="LONG" nullable="false" editable="false" rendering="Sequence Number">
        </column>
        <column name="CREATOR_ID" dataType="LONG" referenceTableName="USER_ACCOUNT" nullable="false" editable="false" rendering="Praticien">
        </column>
        <column name="DOCUMENT_PURPOSE_ID" dataType="LONG" referenceTableName="DOCUMENT_PURPOSE" nullable="false" editable="false" rendering="Type Bi
        </column>
        <column name="TITRE" dataType="STRING" nullable="true" editable="false" rendering="Titre">
        </column>
        <column name="RESUME" dataType="TEXT" nullable="true" editable="false" rendering="Résumé">
        </column>
        <column name="DATE_CREATION" dataType="DATETIME" format="DATE" nullable="false" editable="false" rendering="Date de création">
        </column>
        <column name="DATE_MODIF" dataType="DATETIME" format="DATE" nullable="false" editable="false" rendering="Date de modification">
        </column>
        <column name="FILE_NAME" dataType="STRING" nullable="false" editable="false" rendering="Nom du fichier">
        </column>
        <column name="CONTENT_TYPE" dataType="STRING" nullable="false" editable="false" rendering="Content-type">
        </column>
        <column name="SIZE" dataType="LONG" nullable="true" editable="false" rendering="Taille du fichier">
        </column>
    </columns>
</table>
```

Figure 13: Exemple de modèle décrit en XML à destination du générateur.

Diagramme de séquence de la requête d'obtention du dossier médical

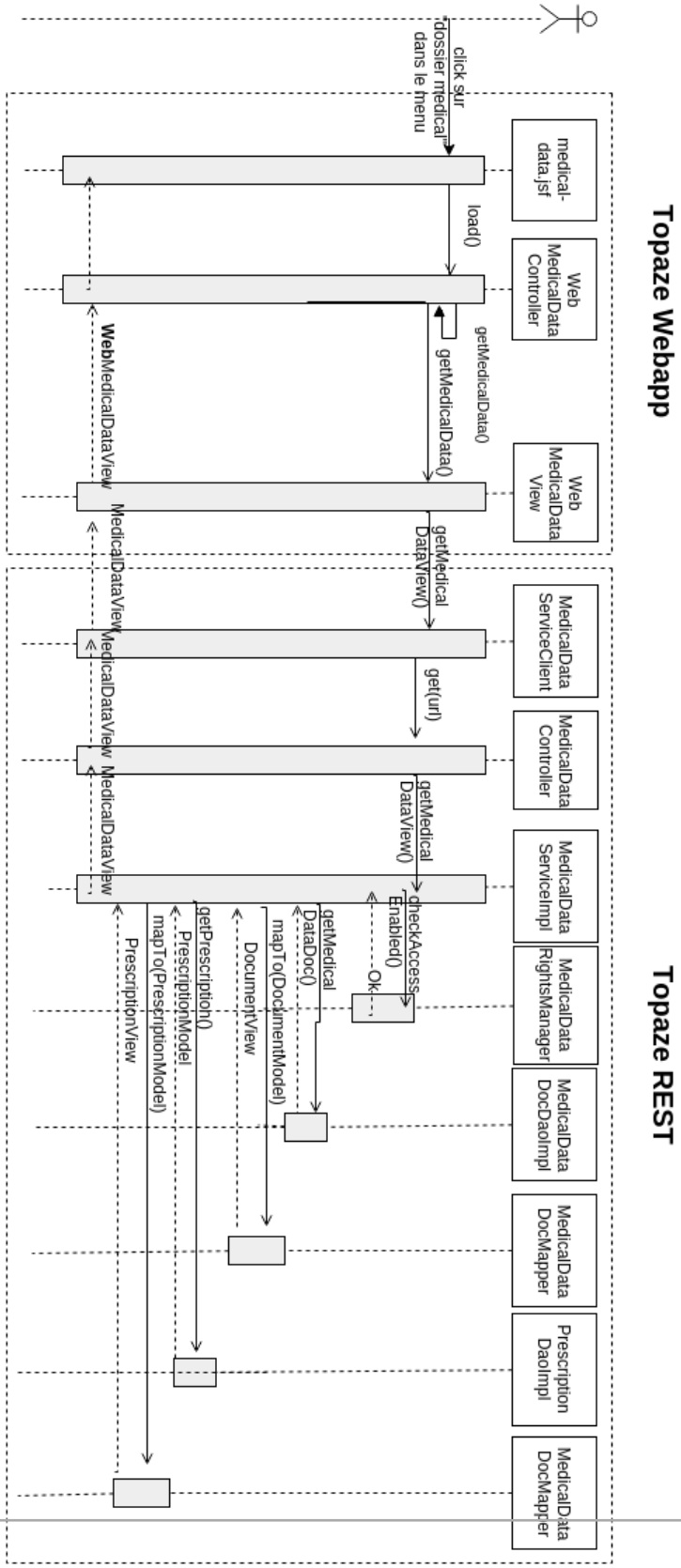

Figure 14: Diagramme de séquence modélisant la requête du dossier médical.

Le dossier médical de Topaze Web

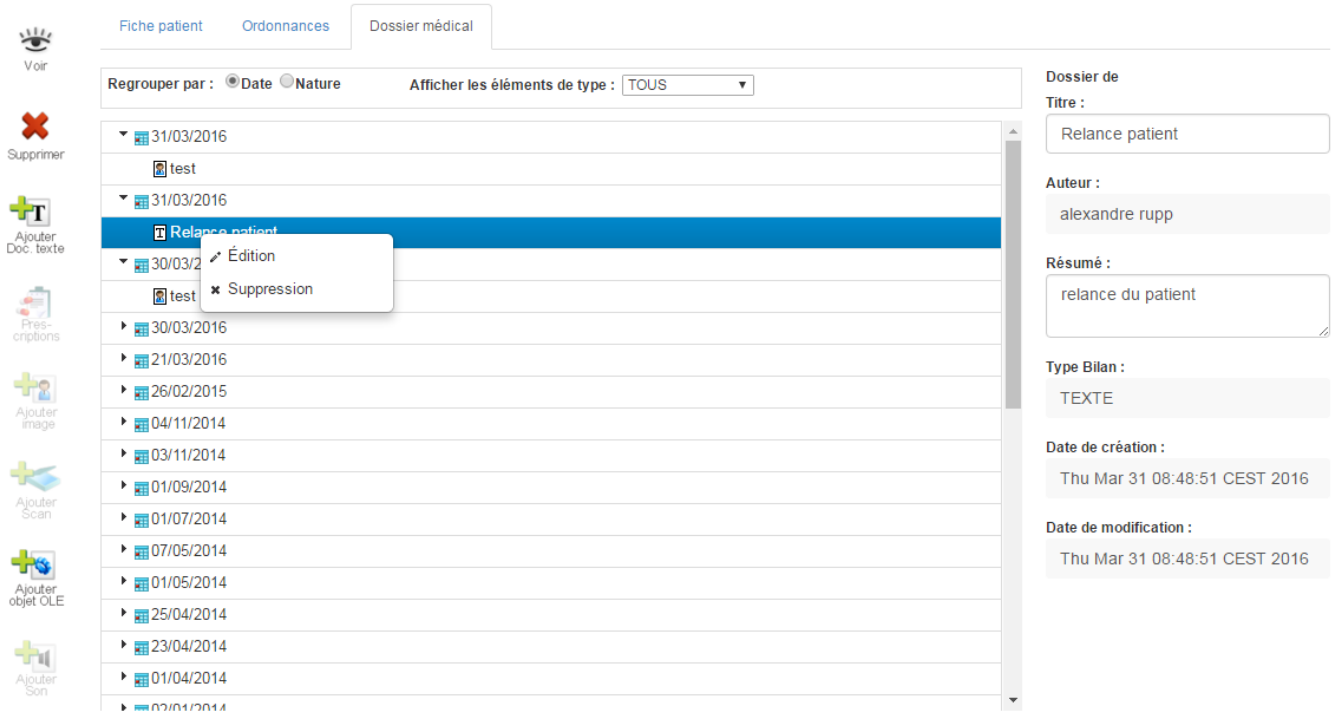

Figure 15: Le dossier médical implémenté dans Topaze Web.

L'editeur de texte de Topaze Web

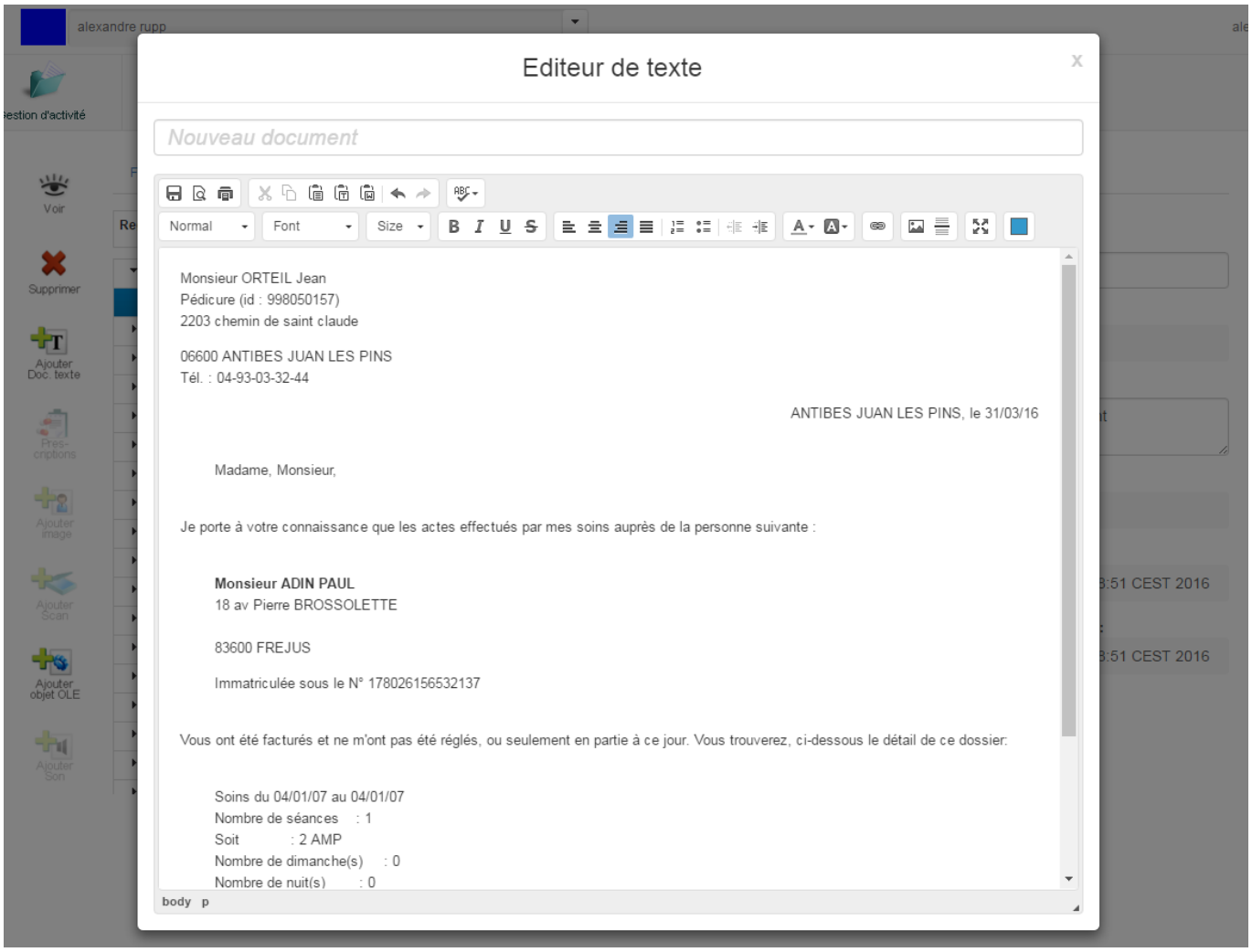

Figure 16: L'éditeur de texte (CKEditor) intégré dans Topaze Web .