

# Rapport de Gestion de projet

## Mon espace santé

Projet Java

*Gestion de projet et génie logiciel Mif01 2022 - 2023*

### Encadrants

[MOY Matthieu](#)

MEDINI Lionel

Lucien Ndjie

### Étudiants

LEDRU Romane

PHANG Roméo



# 1. Présentation globale

## A. Présentation du sujet

Mon Espace Santé est un projet sous la forme d'une application lourde en JAVA, s'inspirant de l'existant [Mon Espace Santé](#), espace numérique de santé pour tous les usagers français. Il permet notamment à chacun de consulter, stocker des documents de santé comme des prescriptions, des rapports, des résultats d'analyses, etc. Cet outil permet la centralisation des données afin de faciliter le dialogue entre les différents professionnels de santé sur le cas d'un patient.

L'échange est gratuit, sécurisé et centralisé autour des services publics français. Il est accessible à tous, activable n'importe quand, et partage aussi des informations sur les directives sanitaires du territoire français.



Nous avons donc utilisé cet outil comme inspiration pour créer un espace sous le format d'une application, où il nous est possible de naviguer aussi bien en tant que médecin qu'en tant que patient.

## B. L'existant

En tant que patient, j'ai la possibilité de consulter les différentes prescriptions que mes différents médecins ont prescrites. Je peux aussi les supprimer.

En tant que médecin j'ai la possibilité de chercher un patient, lui prescrire une ou plusieurs ordonnances, qui sont personnalisées ou que j'ai déjà dans mes prescriptions préférées. Je peux aussi supprimer ces mêmes prescriptions.

En tant qu'utilisateur global, je peux naviguer entre les différents onglets patient et médecin afin de me faire passer pour l'un ou l'autre. Je peux copier le ssid d'un patient pour effectuer une recherche via l'onglet médecin. Je peux créer des nouveaux patients ou des nouveaux médecins pour tester.

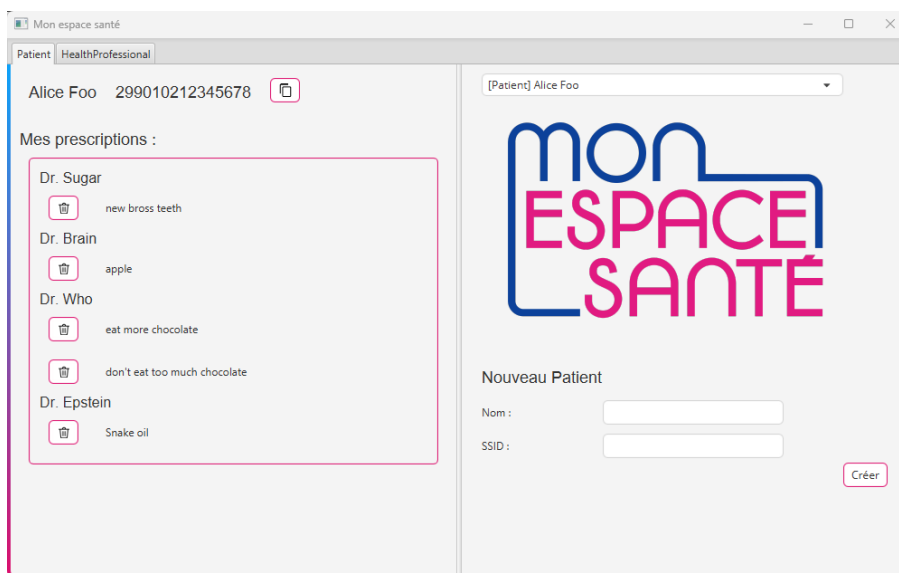


Fig. 1 Vue Patient

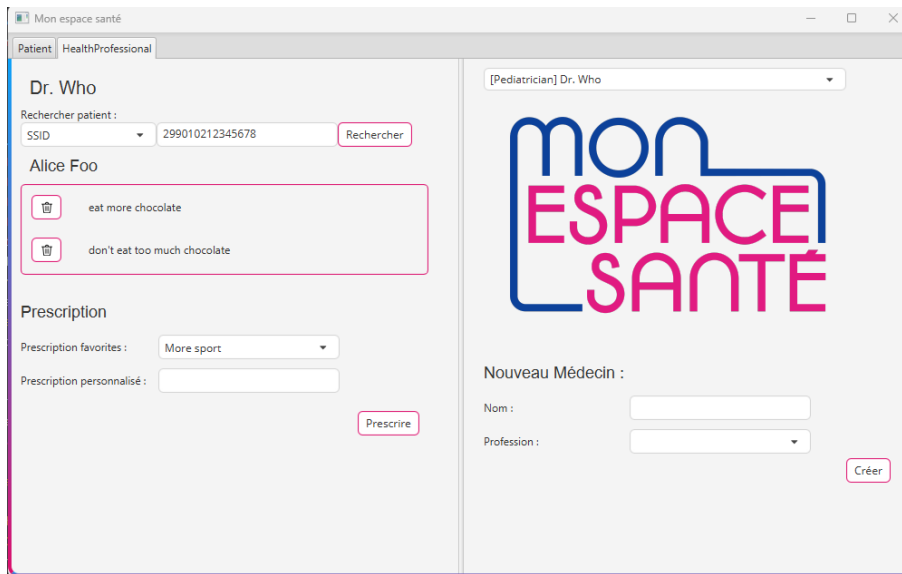


Fig. 2 Vue Professionnel de santé

## 2. Patterns

Le squelette du projet qui nous était donné ne respectait pas de nombreuses bonnes manières de coder. Il nous a donc fallu inclure de **nouveaux patterns** et **refactorer en profondeur** le code.

### A. Patrons de conception

#### I. Modèle-Vue-Contrôleur

Le **modèle MVC** nous permet de **séparer le métier** (la logique de l'application) et **l'affichage**. On a donc une structure **Modèle**, où sont nos objets métiers, la **vue**, où est géré seulement l'affichage et les **contrôleurs** qui font le lien et les répercussions des changements.

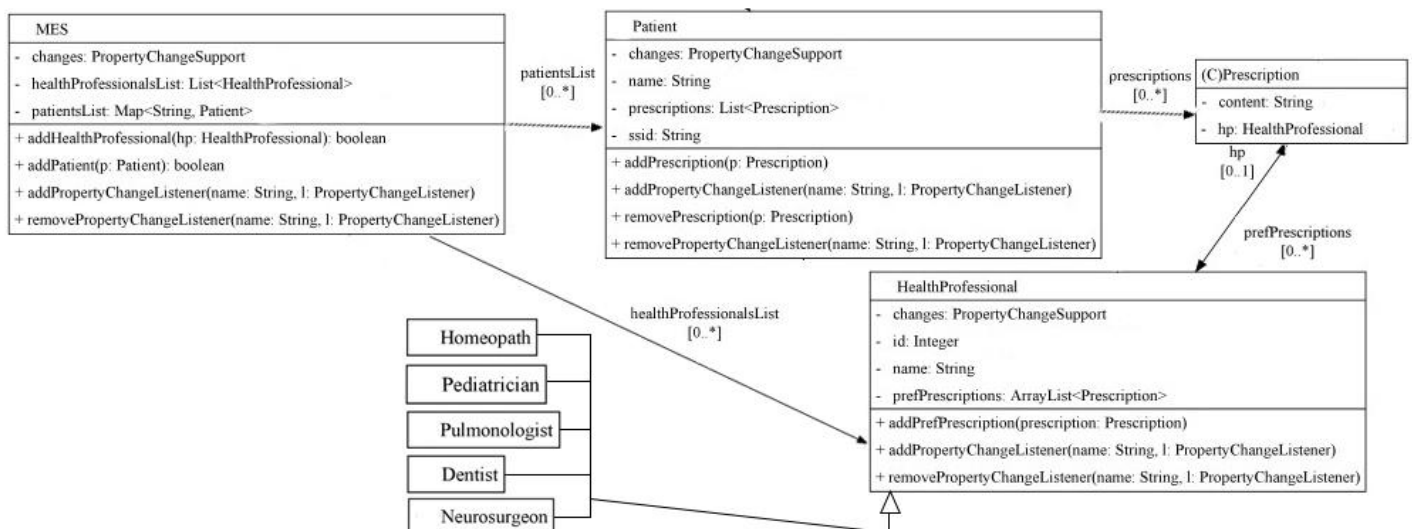


Fig.3 Le diagramme de classe du modèle.

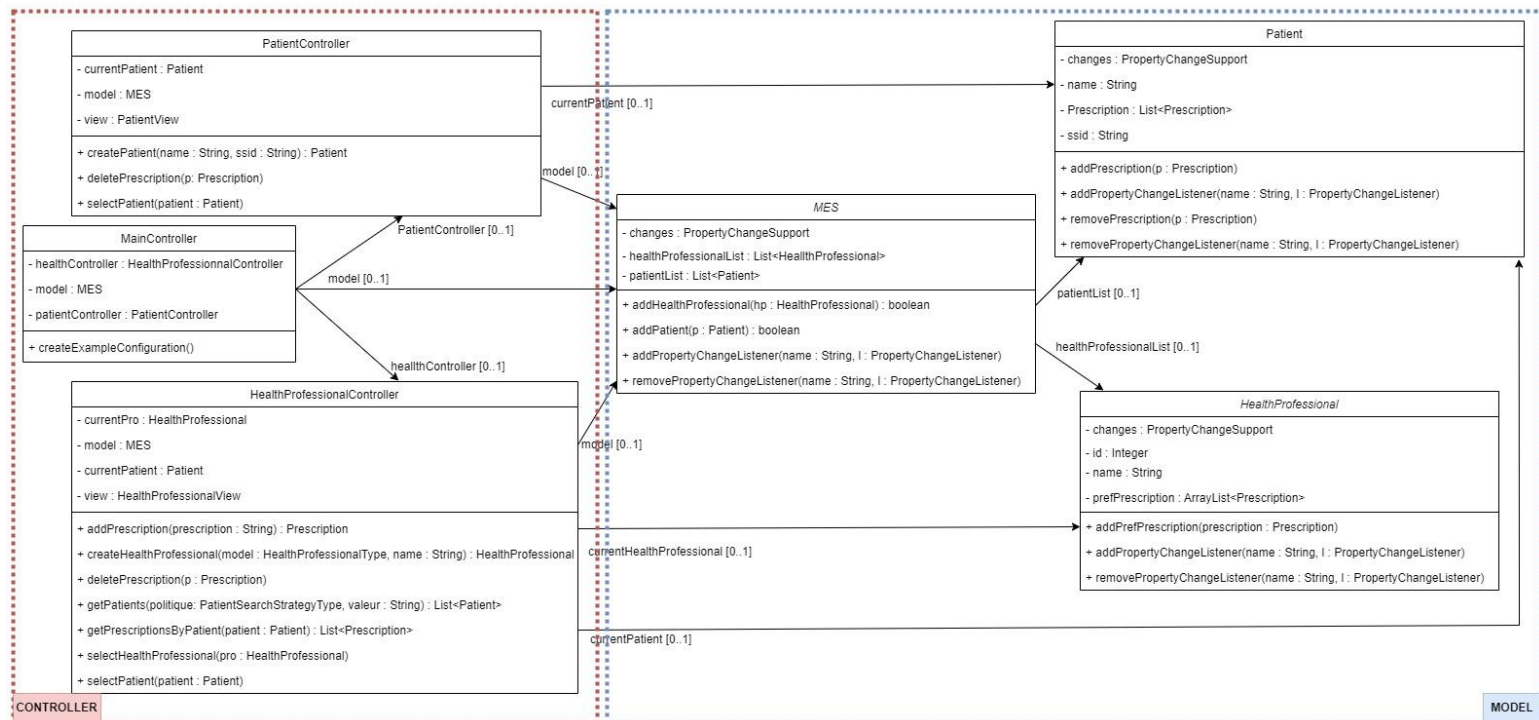


Fig. 4 Le dialogue entre les **controllers** et les **models**

Nous verrons plus loin (cf. C.Pattern de comportement fig. 8) le dialogue entre la vue et les controllers.

## II. Data Access Object

Le **DAO** a pour but d'introduire une **couche d'accès aux données**. Notre objectif étant d'avoir un jeu de données prédéfini afin de tester correctement l'application, il nous fallait donc avoir la possibilité de les charger via un outil qui s'appelle le DAO. Nos données sous forme yaml sont chargées via l'outil snakeyaml, et nous utiliserons seulement l'opération CRUD find. Les opérations insert, update et delete n'ont pas été implémentées ici. *Nota bene*: sur ce schéma ci-dessous, on met l'accent sur le dialogue pour la création des données, mais la Couche DAO a bien accès au modèle afin de créer les objets correspondants.

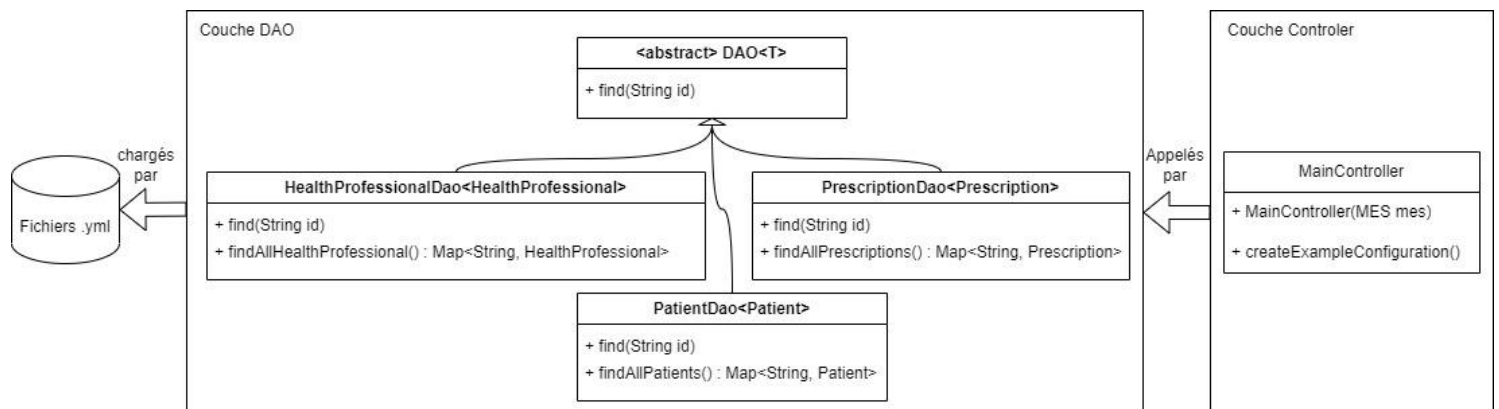


Fig 5. Schéma Data ← couche DAO ← Controller



```
# HealthProfessional
id: "1"
type: PEDIATRICIAN
name: Dr. Who

# Patient
name: Alice Foo
ssid: "299010212345678"

# Prescription
id: "1"
hp: "1"
patient: "872618762826890"
content: One apple a day
```

Fig 6. La structure des données.

## B. Pattern de création

### I. Singleton

La classe MES est une **instance unique**. Elle n'est créée qu'une seule fois dans l'application et nous souhaitons nous assurer que ce soit bien le cas. Son constructeur est donc privé, on n'a qu'une méthode publique statique de cette classe qui va créer cette instance au premier appel et qui ne retournera que celle-ci.

MES
<b>- singleton: MES</b>
- changes: PropertyChangeSupport
- MES()
<b>+ getInstance(): MES</b>
+ getHealthProfessionals(): List<HealthProfessional>
+ getPatients(): List<Patient>
+ addPatient(Patient): bool
+ addHealthProfessional(HealthProfessional): bool
+ addPropertyChangeListener(String, PropertyChangeListener): bool
+ removePropertyChangeListener(String, PropertyChangeListener): bool

Fig 7. Class MES singleton

### II. Fabrique

Une Factory est une classe responsable de la **création d'objets**. C'est une méthode appelée à la place de la création des objets via leurs constructeurs. On ne verra donc pas de "new HealthProfessional()" en dehors de la factory qui s'occupera de créer les objets de ce type.

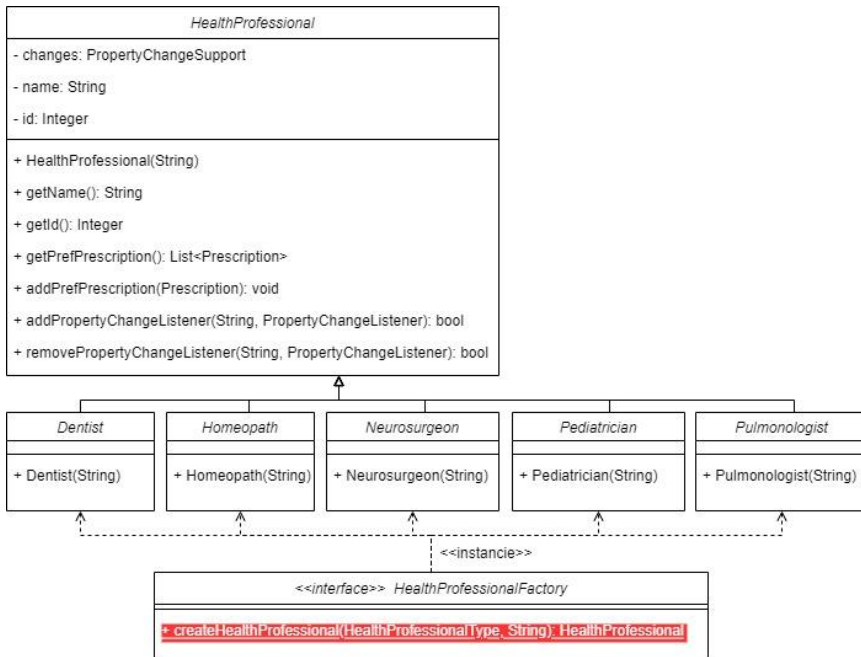


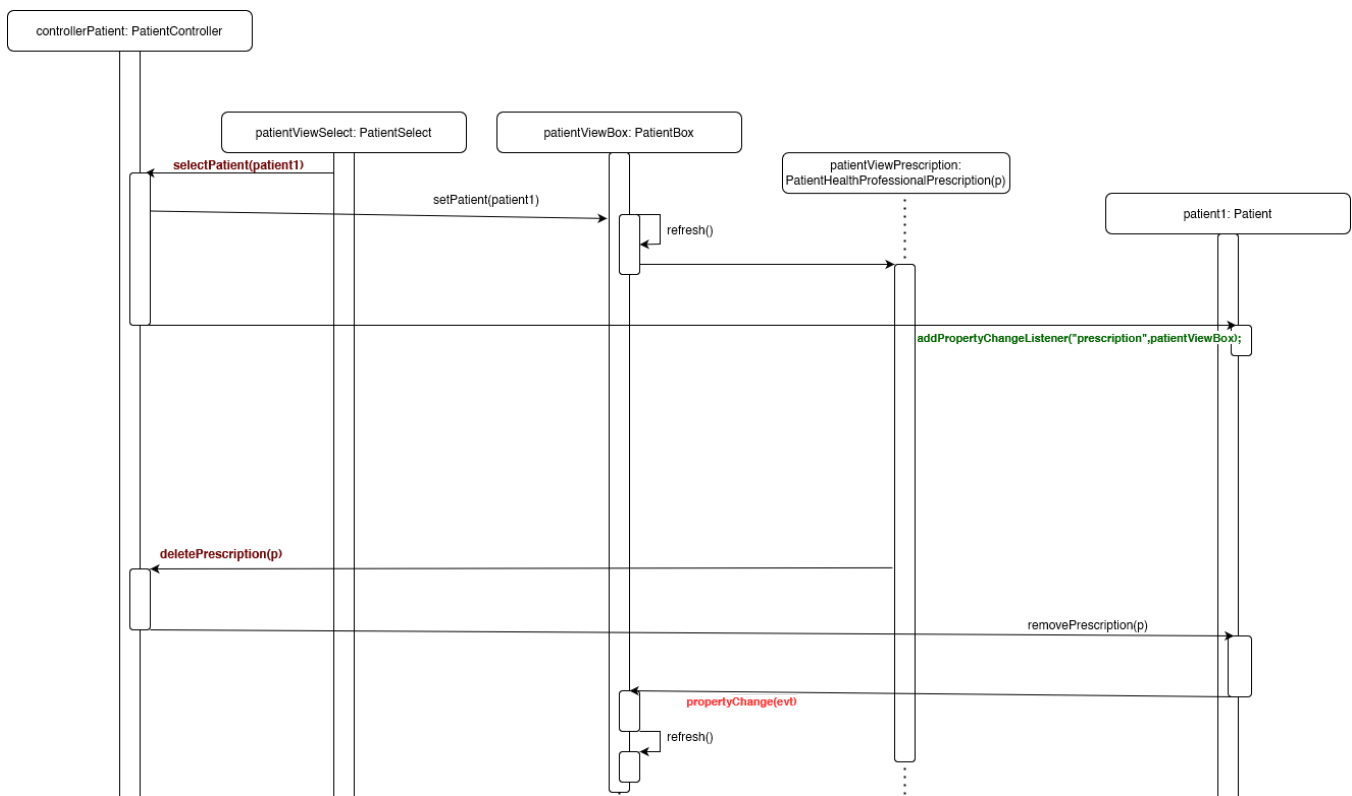
Fig. 8 UML Factory HealthProfessional

## C. Pattern de comportement

### I. Observateur (Observer)

L'**Observateur** est un pattern de conception qui permet à un objet **d'observer** un autre objet afin de recevoir des **notifications** lorsque celui-ci **change d'état**. Ce pattern est implémenté dans le projet afin de notifier les Vues de changements de propriété aux seins des patients, professionnels de santé et MES.

Dans le diagramme de séquences suivant, les interactions utilisateurs sont de couleur **marron**. L'appel **vert** est l'ajout de la vue en tant qu'observateur de la classe *Patient*. Les éléments observateur reçoivent les notifications de modification de l'objet grâce à l'appel en **rouge** (L'objet envoyé est de type *PropertyChangeEvent* contenant la nouvelle valeur modifiée).





## II. Fonction de rappel (Callback)

Le pattern **Callback** permet de passer une fonction d'un objet (client) à un autre objet (appelé serveur) qui exécutera à un moment dans le temps ultérieur. L'usage de callback est notamment présent lors d'affichage d'alertes avec la classe d'utilité "**EasyAlert**" lorsque nous voulons associer une action à un bouton sur le dialogue.

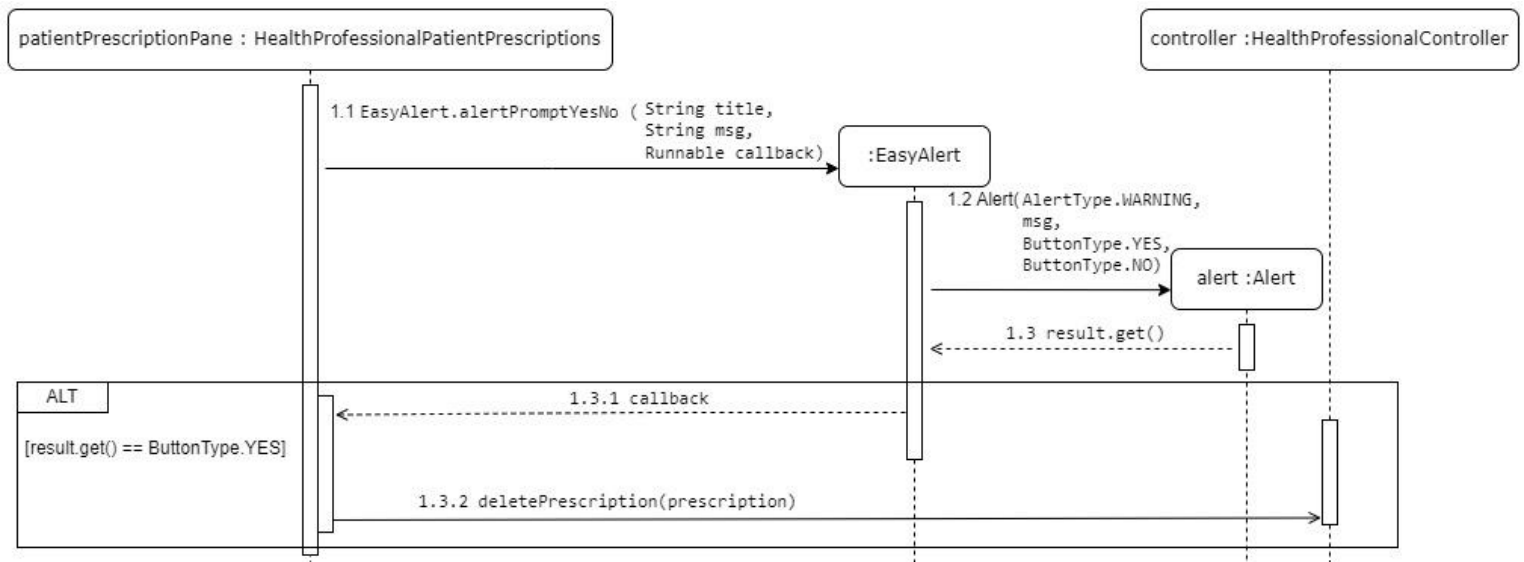


Fig 10. Diagramme de séquence callback suppression prescription

## III. Stratégie

Le pattern **Strategy** permet d'effectuer des opérations différentes selon le contexte et la stratégie indiquée. Celle-ci permet également d'avoir un code extensible avec notre stratégie de recherche de patient dans le projet. On a donc implémenté une classe **BaseStrategy** qui sert de modèle pour n'importe quelles stratégies. On peut donc ajouter/retirer des stratégies facilement.

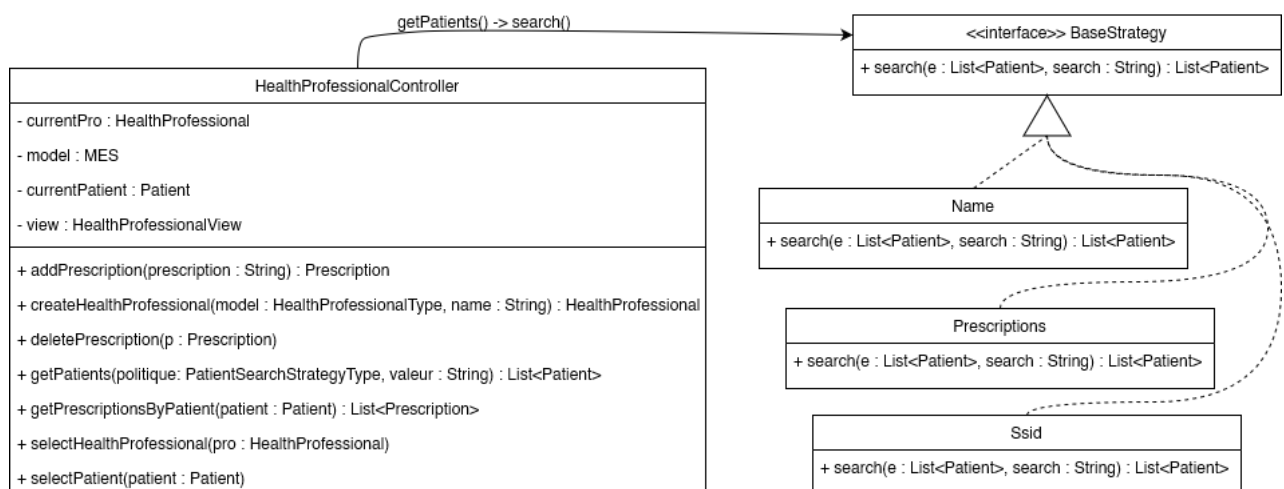


Fig.11 Diagramme de classe implémentant le pattern strategy.





Nous en avons implémentés 3 :

- **Name** : recherche sur le **début du nom** strict sur la casse.  
*exemple: "al" ne va pas retourner les patients comme "Alice" mais "Al" oui.*
- **Prescription** : recherche **inclusive** en strict sur la casse.  
*exemple : "New" ne va pas retourner "new boss teeth" alors que "bros" ou "tee" oui.*
- **SSID** : recherche **totalement stricte**.  
*exemple : "299010212345678" va retourner le Patient Alice, mais pas "2990".*

*Nota bene* : Si une recherche renvoie plusieurs Patient, il affiche un select avec la liste des patients qui correspondent. Sinon il affiche directement le Patient trouvé.

## D. Ajouts Interface

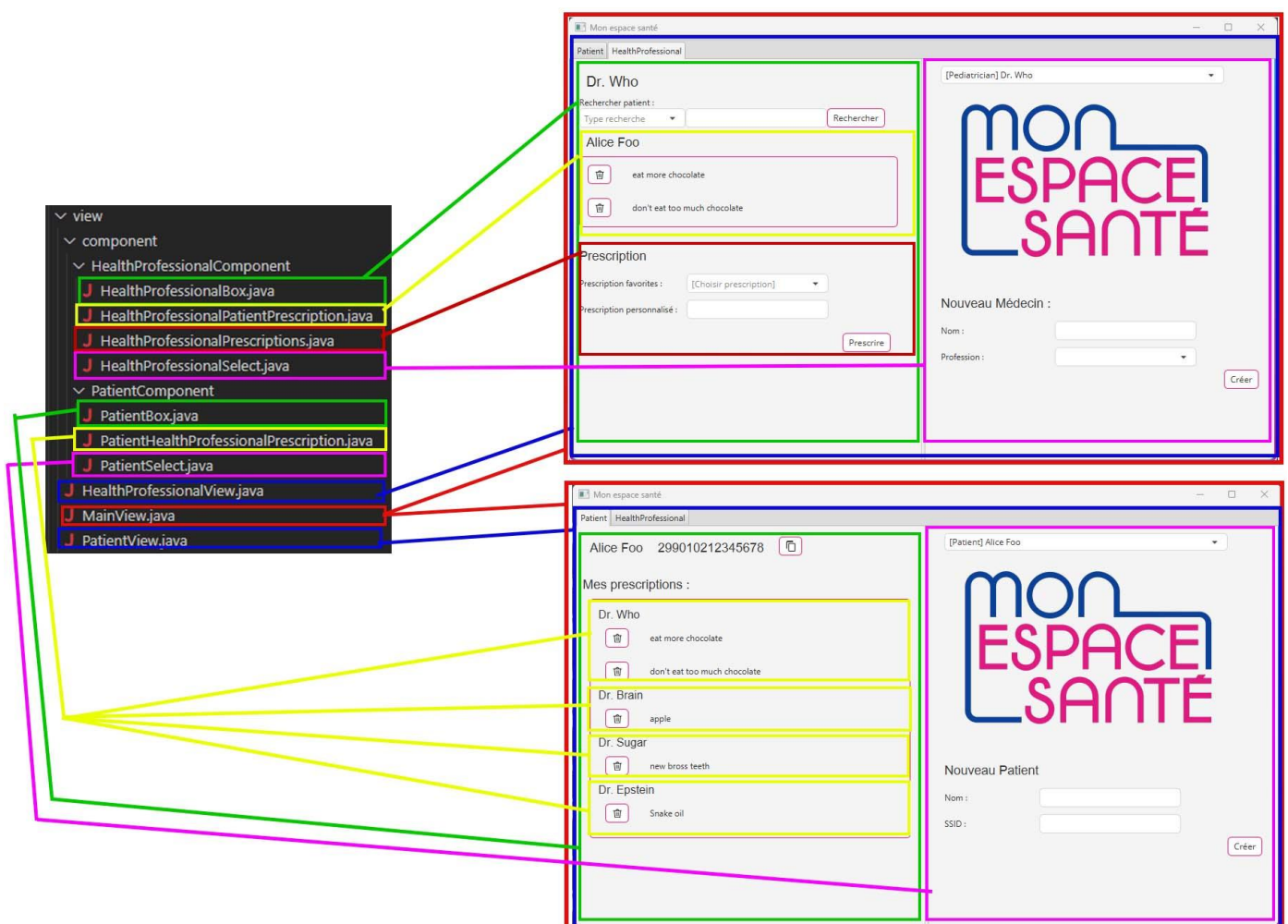
En ce qui concerne l'interface, nous avons opté pour une librairie Java en open source afin d'améliorer le design de nos composants (Boutons, ComboBox, etc.) : [MaterialFX](#).

On aura aussi inclus du css pour des besoins de design, et des images (icons pour les boutons, logo du projet).

Les vues sont structurées sous forme de composants dits composants, afin d'intelligemment séparer certaines portions et d'alléger les fichiers vues.

De ce fait, on a des fichiers avec moins de méthodes, moins lourds à lire et qui peuvent être facilement modulables (si on veut retirer une ou deux parties de la vue, on retire simplement une seule ligne de code d'appel au composant de la vue racine par exemple).

Fig. 12 Schéma découpage de la vue en composants.







C'est un fonctionnement largement utilisé en javascript, comme le Framework VueJS par exemple.

### 3. Ethique

Le sujet des données de santé est large, très large. C'est un sujet récurrent depuis que l'informatique et la numérisation des données se sont généralisées dans le milieu de la santé et aujourd'hui encore, elles sont sujettes à de nouvelles discussions sur son aspect éthique et impactant.

Aujourd'hui encadré par la **CNIL** ([Commission nationale de l'informatique et des libertés](#)) et par le fameux **RGPD** ([Règlement Européen sur la Protection des Données Personnelles](#)), les données de santé se démarquent de par leur caractère sensible et personnel. Fun fact: notons néanmoins que les données à caractères physiques comme le nombre de pas relevés par une montre ou un appareil connecté n'entrent pas dans la notion de données de santé.

Ces données sont encadrées par des [règlements et des guides](#) adressés autant aux professionnels de santé qu'aux créateurs et développeurs de contenus et supports informatiques liés au domaine de la santé. Les hébergeurs, les développeurs d'applications et sites comme Doctolib, les libéraux, les hôpitaux et même les utilisateurs et les journalistes sont soumis à la loi (notamment la loi de modernisation du système de santé qui interdit depuis 2016, sauf dérogation de la part de la CNIL si l'enquête à une finalité d'intérêt public, l'accès aux données des hôpitaux [aux journalistes](#)).

Outre la police des données que peut représenter la CNIL, Mon Espace Santé est encadré par le **SNDS**, [Système National des Données de Santé](#), qui permet de chaîner toutes les données entre les hôpitaux, l'assurance maladie, causes médicales de décès, handicaps, organismes privés (mutuelles).

On pourrait se demander, aujourd'hui dans un monde ultra numérisé, s'il n'est pas dangereux et immoral d'avoir des données aussi sensibles rassemblées et accessibles sur des interfaces de plus en plus nombreuses et qui peuvent tendre à être de moins en moins encadrées. On peut facilement trouver des sanctions de la CNIL pour des [fuites de données biomédicales](#) massives, où le prestataire de service et l'éditeur de logiciel ont accumulé les fautes et les litiges.

Au même titre, nous avons encore la chance d'avoir une interdiction stricte sur l'accès à ces données, mais les données fuitées sont rarement récupérables, et l'unification derrière MES et Doctolib renforce ce risque: ces données valent un prix d'or et elles permettraient au ministère de la santé ou à des organismes privés d'observer à la loupe les besoins et les tendances médicales des Français.

Aujourd'hui même, on observe des organismes anonymiser des données afin de les vendre. Celles-ci sont certes impossibles à relier à un groupe de population ou une personne en particulier mais peuvent servir à des fins aussi bien morales qu'immorales (aka les différents sites liés à la santé qui se permettent l'utilisation de cookies afin de récupérer des données et affiner des suggestions aux utilisateurs).

Le sujet est vaste, mais pas si profondément alarmant. De très nombreux acteurs s'activent aujourd'hui afin d'éviter toutes utilisations malveillantes d'outils liés à la santé de leurs utilisateurs. Nous avons encore de nombreuses garanties et garde-fous et en tant que



futurs acteurs dans le numérique, nous avons aussi notre rôle à jouer sur la protection de nos données personnelles, aussi et surtout de nos données de santé.

Au sein de notre projet, les données sont totalement conservées en local. S'il devait être réellement utilisé, il ne serait absolument pas réglementaire sur la protection de ces données, qui ne seraient pas automatiquement supprimées après le temps réglementaire par exemple, ou elles seraient extrêmement faciles d'accès si une personne extérieure a accès à l'ordinateur de notre utilisateur.

Dans le cadre d'une vraie application, les données doivent être **cryptées** et être obligatoirement hébergées chez un **fournisseur spécialisé et certifié HDS** (Hébergeur des Données de Santé). Aussi, l'archivage de ces données est soumis à de nombreuses contraintes et sécurités (**Intégrité, confidentialité, sécurité**). Des certifications (comme ISO 27001 HDS) permettent de garantir une mise à l'abri d'un sinistre ou d'un vol les données de santé.

## 4. Tests

Dans notre application, les classes métiers tels que les *Models*, *Dao* et *Controllers* sont testées automatiquement par JUnit. Cependant l'implémentation de ces tests automatiques sur la vue étaient trop sophistiquées par rapport à la complexité de l'application. Nous avons donc manuellement testé cette partie tout au long du projet.

### A. Tests des interactions

Les tests de nos éléments avec interactions étaient la partie la plus simple des tests de la vue. En effet, lorsque nous ajoutons des *Buttons*, *Combobox*, ou autre élément avec une quelconque interaction, il nous suffisait de lancer l'application et vérifier que le comportement correspondait aux attentes. Les cas d'erreurs furent aussi testés, par exemple lorsqu'une entrée utilisateur nécessaire à un traitement est vide ou dans un format incorrect.

Exemples de tests d'interactions mis en œuvres durant le développement :

- ☐ Tester la création d'un nouveau médecin :

1. Si un ou plusieurs champs ne sont pas remplis : erreur.
2. Si la création est réussie : affichage et sélection automatique du nouveau HP créé et il est aussi accessible sur la seconde vue.

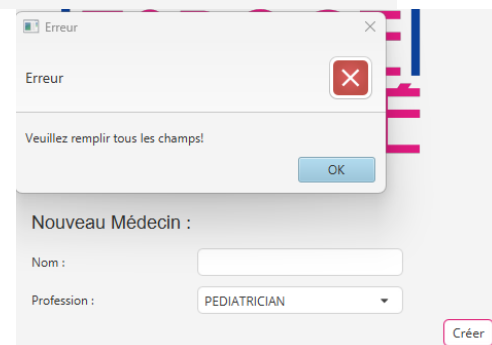


Nouveau Médecin :

Nom :

Profession :

Créer



Erreur

Erreur

Veillez remplir tous les champs!

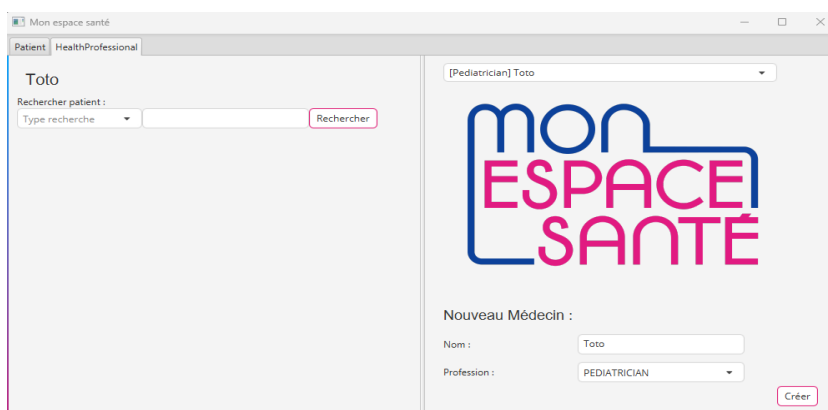
OK

Nouveau Médecin :

Nom :

Profession :

Créer



Mon espace santé

Patient HealthProfessional

Toto

Rechercher patient :

Type recherche  Rechercher

[PEDIATRICIAN] Toto

mon ESPACE SANTÉ

Nouveau Médecin :

Nom :

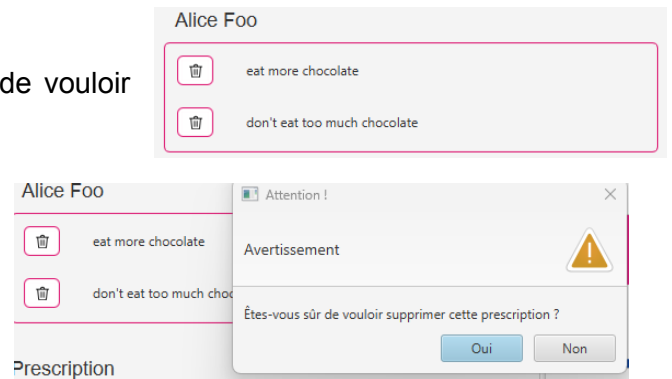
Profession :

Créer



☐ Tester la suppression d'une prescription :

1. Au clic: vérification de la bonne volonté de vouloir supprimer la prescription.
2. Si non : la prescription reste affichée.
3. Si oui : la prescription disparaît des DEUX vues.



Le reste des tests s'est principalement basé sur le même principe avec des tests manuels pas à pas vérifiés sur les deux vues en simultanée.

## B. Tests des évènements

Les tests d'évènements sont similaires aux tests d'interactions. Au lieu que l'évènement écouté provient d'un élément visuel; celui-ci était envoyé depuis les modèles écoutés (Modèle Push-Based du MVC). Nos principaux évènements sont:

- L'ajout d'un Patient dans la base de MES
- L'ajout d'un Professionnel dans la base de MES
- L'ajout d'une prescription dans un Patient
- La suppression d'une prescription dans un Patient

Afin de déclencher l'un des évènements, il suffisait d'exécuter la tâche qui lui est associée et par la suite vérifier le comportement de l'application.

## 5. Conclusion

Ce projet nous a permis d'appliquer et d'apprendre de nombreuses bonnes pratiques dans l'élaboration d'un projet de développement en groupe. Il sera sans nul doute très utile dans l'application de ces connaissances en entreprise (en alternance ou en contrat, dans un futur proche). Que ce soit du côté technique, design pattern, code propre, ou du côté organisationnel, avec git, les pull request, tag et issues, ce sont des notions largement utilisées et encouragées par le monde du travail.

Nous tenons donc à remercier nos enseignants et les différents intervenants du cours mif01 pour cette mise en pratique tout du long du semestre.