TP Programmation Orientée Objet – Rapport

Damien Garcia – Florian Echelard

Novembre 2022

Table des matières

1 Introduction **2**
1.1 Objectifs 2
1.2 Approches 2

2 Implémentation **2**

3 Utilisation de l'interface **4**

4 Perspectives **5**

5 Conclusion **5**

Table des figures

1 Diagramme UML présentant la structure de l'implémentation 3
2 Structure en « couches » de l'interface 4

1 Introduction

Le but de ce projet est de fournir un outil de gestion de radiographies pour les médecins au sein d'un centre de santé en s'appuyant sur le paradigme de programmation orienté objet. Pour cela, nous avons commencé par établir les objectifs d'ergonomie et d'accessibilité relatifs à ce logiciel, discuté ensuite de l'implémentation en classes et leur interface relationnelle, puis des éléments utilisés au niveau du code pour élargir et préciser l'application.

1.1 Objectifs

L'interface définie par le projet doit permettre à un médecin d'accéder aux radiographies de ses patients, ainsi qu'aux données associées à ces radiographies. Un médecin donné, n'aura accès qu'aux radiographies de ses propres patients, mais pas à celles des autres médecins.

Du fait de la séparation des accès aux données, des méthodes de recherches, autorisant ou non l'accès à celles-ci doivent être établies et implémentées. De plus certaines de ces données doivent être protégée par mot de passe, notamment le rapport médical associé aux examens médicaux. Dans un objectif de cohérence générale de l'implémentation, nous avons rapidement opté par implémenter un système de *Login*, c'est-à-dire qu'un utilisateur, avant de pouvoir accéder à quelconque donnée, doit s'identifier sur la base de données ce qui permettra par la suite de lui rendre disponible uniquement les dossiers médicaux lui étant associés.

1.2 Approches

Notre première approche pour gérer les données utilisateurs séparément était de regrouper les éléments d'intérêt dans un classe à l'interface des utilisateurs et de la base de données pour distribuer de manière contrôlée les attributs. Nous avons cependant fait évolué cette idée par la suite pour inclure l'interface et la gestion de la base de donnée dans deux objets séparés, respectivement pour gérer la sauvegarde des données et l'autre pour faire appel aux éléments utilisateurs lors d'une session.

Pour la mise en place de la sauvegarde des instances, nous avons envisagé la méthode de *serializing*, qui permet de concevoir l'application en deux états, active et inactive. A l'activation, les données sont téléchargées (*download*), c'est-à-dire que les différents objets sont instanciés, et à la sortie du logiciel sauvegardées (*update*) en format binaire. Le temps qui nous a été alloué pour ce projet, ne nous a pas permis de mettre en place cette méthode, nous avons donc préféré un format texte classique, avec une grammaire bien définie permettant la lecture et l'écriture des instances de manière automatisée.

L'interface visuel (= le « menu ») ayant accès aux méthodes de la base de données (détaillé par la suite), l'existence d'une classe faisant le lien entre utilisateur et radiographie n'est plus nécessaire. Ces instances sont regroupées dans la **session**.

2 Implémentation

La volonté de proposer plusieurs profils pour ouvrir cette application non seulement au corps médical mais aussi aux patients, nous a motivé à créer une classe parent **User**, qui possède des attributs concernant les renseignements généraux, soit le nom, le prénom, et le mot de passe. Contrairement au cahier des charges indiquant le besoin de protéger par mot de passe uniquement le compte-rendu médical et les clichés associés, nous avons fait le choix de vérifier l'identité des utilisateurs à l'ouverture de l'interface. Ainsi, l'utilisateur lambda ne peut pas non plus accéder à la liste des examens. Si un utilisateur souhaite consulter son dossier, il doit être inscrit dans la base de données, où créer un compte qui pourra le cas échéant être modifié pour ajouter de nouveaux examens médicaux.

Un médecin (classe **Doctor**) possède en attribut une liste de patient qui lui sont attribués. Des méthodes pour ajouter, supprimer des patients de liste lui sont accessible. Via la classe **Database-**

Handling, le médecin peut accéder à la liste de radiographies de chacun de ses patients. Cette classe lui permet d'ajouter, supprimer et modifier des radiographies.

Les radiographies (classe **Radiography**), possèdent un identifiant, un type (*xRay, MRI, ultrasound*) un état permettant de savoir si la radiographie à été effectuée où si elle est programmée à une date ultérieure (*done, pending*) ainsi qu'une date. Les méthodes de cette classe permettent d'ajouter de nouveaux clichés (classe **Snapshot**) ou d'en supprimer, ainsi que les méthodes pour ajouter ou supprimer le compte-rendu médical (classe **Report**).

Un patient utilisant pourra voir sa liste de radiographies, accéder aux clichés et compte-rendu associés via des méthodes d'accès par identifiant, sans pouvoir en modifier le contenu. Dans un cas où la liste de radiographies est importante, où simplement pour faciliter la recherche, une méthode de recherche par date est accessible au patient (comme au docteur).

Comme expliqué précédemment, la classe **DatabaseHandling** contient les méthodes à sauvegarder les informations des instances vers des fichiers texte, pour former une base de données une fois les modifications effectuées dans le programme : ce sont les méthodes d' « *upload* ». Inversement, les fonctions de « *download* » permettent au moment de l'ouverture du programme, d'instancier tous les patients, docteurs et radiographies déjà présent dans la base de données.

Enfin la classe **CommandLineInterface** offre des menus interactifs, des visualisations de toutes les informations de la base de données, les options disponibles aux utilisateurs selon leur type (**Doctor** ou **Patient**) et envoie par « message » les modifications effectuées dans la « session » vers **DatabaseHandling** pour les stocker.

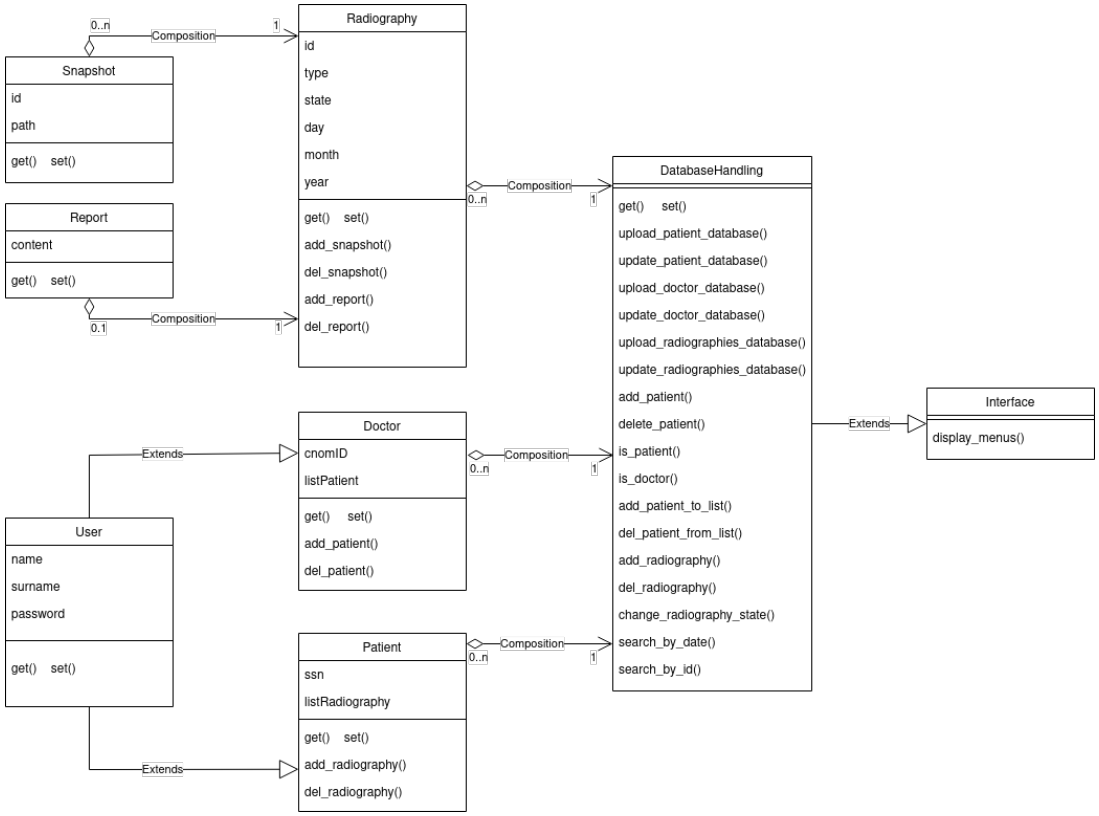

FIGURE 1 – Diagramme UML présentant la structure de l'implémentation

3 Utilisation de l'interface

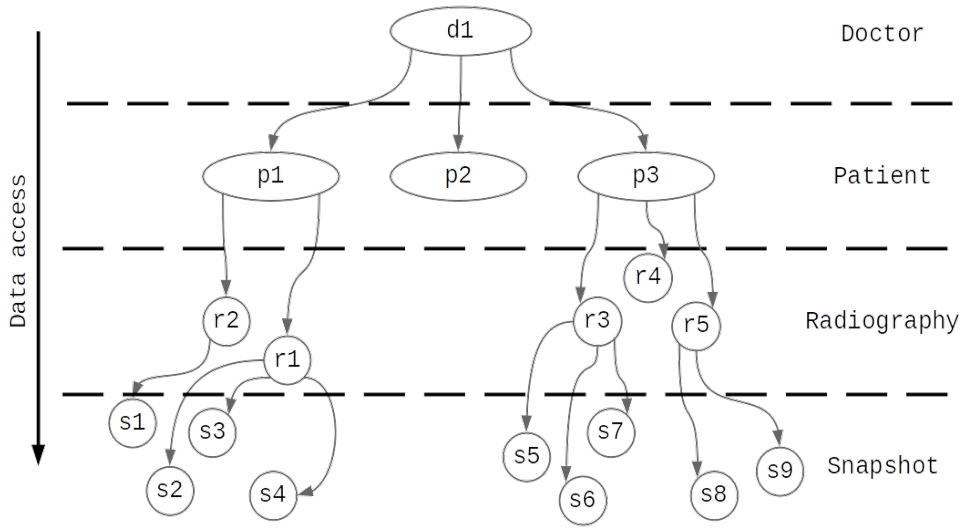

FIGURE 2 – Structure en « couches » de l'interface

4 Perspectives

Pouvant affirmer avec confiance avoir respecté le cahier des charges, nous avons cependant rencontré beaucoup de possibilités d'implémentation au cours du développement. De part les contraintes de temps, nous sommes allé à l'essentiel, favorisant le fonctionnement de l'infrastructure interface - base de données plutôt que l'implémentation d'outils poussés pour gérer certaines des tâches de notre application.

Parmi celles dont nous perçevons une possibilité d'amélioration est le serialising, la base de donnée étant du texte pour l'instant, écrit en dur et actualisé à la sortie de l'application. Du "vrai" serialising, compilant les objets en binaire sur le disque dur, permettrait d'implémenter des méthodes évitant le conflit que deux utilisateurs possèdent le même identifiant, ce qui est le cas pour l'instant.

Le reste des modifications auxquelles nous avons pensé prennent avantage du language C++ et de son environnement. Par exemple, la gestion des dates et de types de données connus par des librairies spécifiques apporteraient une modularité supplémentaire quant aux vérifications et fonctions que nous souhaitons faire sur celles-ci. Ensuite, en raison de comment nous avons construit notre interface, nous avons fait de la surdéfinition de plusieurs constructeurs, et pourrions utiliser un constructeur prenant avantage du polymorphisme, qui regroupe l'initialisation d'élément en commun pour les constructeurs surdéfinis. Certaines des classes liées que nous avons écrite pourraient bénéficier de la construction d'un patron dédié, comme le couple Docteur - Patient, ou Photos - Compte rendu.

Finalement, nous avons construit la classe abstraite Utilisateur pour l'héritage, mais aurions pu étendre son utilité en lui fournissant des fonctions virtuelles et bénéficier de méthodes optimisées par la suite.

5 Conclusion

Notre application permet un accès à une base de données dynamique et la séparation de méthodes de consultation et d'édition selon le statut présent dans celle-ci.

L'architecture des classes nous permet d'encapsuler les utilisateurs et les données de radiographie dans une interface grâce à une classe de gestion dédiée.

Nous avons réussi à remplir le cahier des charges, ainsi que d'aller plus loin pour se rapprocher d'une application réaliste, même si quelques étapes restent à considérer avant cela, que ce soit au niveau de l'architecture ou des fonctionnalités.

N.B. : *Tout numéro de sécurité sociale où identifiant de l'ordre des médecins présent dans notre base de données est totalement fortuit et ne résulte absolument pas d'une attaque informatique sur les serveurs gouvernementaux.*