Master 2e année Bioinformatique pour les Biologistes de l'université de Nantes

Projet en Programmation Orientée Objet

Année Universitaire :
2022/2023
Auteur :
Brieuc Quemeneur
Florent Boyer

Table des matières

1 Introduction

Dans l'optique de réduire l'empreinte carbone du système de santé en France, diminuer la consommation de papier est essentiel. Pour atteindre cet objectif, notre projet propose une application à un centre de radiologie leur permettant de gérer les dossiers de leur patients en temps réel, et d'automatiser leur suivi par les médecins.

2 Cahier des charges

L'application doit permettre aux radiologues de gérer l'ensemble des examens radiographiques réalisés au travers d'une seule représentation graphique. Le programme doit prendre en charge l'ajout, la supression et l'affichage des caractéristiques des examens par patients, par date, par numéros d'examens. Ce sont autant de fonctionnalités et de contraintes à prendre en compte pour structurer le projet et pour proposer une application robuste, complète ainsi que simple d'utilisation.

2.1 Besoins et contraintes du projet

Le projet doit implémenter des fonctionnalités précises tout en respectant des contraintes d'accès et des droits d'utilisateurs sur la base de données.

2.1.1 Des rôles différents

Assurer une gestion des rôles et des privilèges des utilisateurs est une partie importante de l'application. On ne souhaite pas que tous les utilisateurs aient les mêmes droits sur la base de données, notamment parce que certaines données sont sensible : comme les rapports ou les photos associés aux radiographies qui sont soumis au secret médical.

Ainsi, l'application comprend deux catégories d'utilisateurs qui se distinguent par leur rôle donnant accès à des fonctionnalités différentes.

- Les Patients : Utilisateurs ayant uniquement le droit de voir leur propres liste de radiographies sans pouvoir les modifier ni avoir accès au rapport médical privé associé.
- Les Docteurs : Ils peuvent voir uniquement les radiographies des patients qu'ils prennent en charge, ils peuvent ajouter des clichés ces radiographies, les supprimer, ainsi que consulter le rapport médical associé ou le modifier.

Cette hiérarchie de rôles entre les patients et les docteurs nécéssite un système d'authentification dans l'application afin de confèrer à l'utilisateur les droits correspondant à son statut de patient ou de médecin ainsi que de charger uniquement les données qui concerne l'utilisateur authentifié depuis la base de données.

2.1.2 Lecture et écriture dynamique dans la base de données

Une base de données médicale devrait dans l'idéal être accessible à plusieurs utilisateurs simultanément. Il s'agit d'une contrainte importante qui demande au programme de lire et d'écrire dynamiquement dans la base de données, ainsi qu'un système de versionnage approprié.

Pour limiter cette containte et rendre la base de donnée accessible à plusieurs utilisateurs en même temps, le programme fait régulièrement des requêtes à la base de données afin de toujours présenter des données mises à jour à l'utilisateurs et insèrer les modifications en temps réel dans la base de données.

Ce système évite de réécrire la base de données à la fin de chaque session utilisateur mais demande beaucoup d'allers retours entre le programme et la base de données. De plus, la sensation d'instantanéité des modifications de la base de données et le support multi utilisateurs n'est pas parfaitement implémenté dans ce programme, puisque si l'un des fichier de la base de données est accèdé exactement au même moment par plusieurs utilisateurs l'accès aux ressources sera refusées.

L'implémentation de ce système supporte quelques utilisateurs en simultanés mais devra être repensé pour supporter la connexion simultanée de davantage d'utilisateurs.

2.1.3 Une structure robuste et extensible

Le code source du programme met en oeuvre les principes d'encapsulation de la Programmation Orientée Objet en C++ ainsi que les bonnes pratiques afin que la structure du code soit à la fois simple à comprendre et que de nouvelles fonctionnalités puisse y être ajoutées dans le futur.

De plus, la structure relationnelle de la base de données offre également une certaines flexibilité dans la création de nouvelles tables pour la base de donnée médicale.

2.2 Fonctionnalités et implémentation

Le programme prend en charge de nombreuses fonctionnalités dont l'accès est dépendant du rôle de l'utilisateur courrant. Les fonctionnalités proposées suivent des opération CRUD (Create, Replace, Update, Delete) sur la base de données médicale contenant les radiographies.

2.2.1 Système d'authentification

Avant de proposer les fonctionnalités à l'utilisateur, une authentification est nécéssaire afin que le programme puisse vérifier si l'utilisateur existe et qu'il possède les droits qu'il prétend avoir.

L'authentification se fait en trois étapes :

— Le choix du rôle : C'est le moment où l'utilisateur indique le rôle qu'il incarne (à savoir docteur ou patient).

— L'identification : Le moment où le programme demande le nom et le mot de passe à l'utilisateur.

— La vérification : Après que l'utilisateur aie saisi sont nom et son mot de passe, le programme procède à une vérification de ces données. Si l'utilisateur est bel et bien existant et que les informations indiquées sont juste alors il accède aux fonctionnalités qu'il a le droit d'utiliser.

2.2.2 Lecture des données

L'une des opérations que l'utilisateur peut réaliser après s'être connecté est l'affichage des radiographies qui le conserne. Par défaut, il s'agit de l'unique fonctionnalité accessible par le patient et nécéssite la lecture du fichier des radiographies dans la base de données.

Pour le docteur cependant, la lecture dans la base de données est plus complexe puisqu'elle concerne aussi la lecture des radiographies depuis la base de données par le programme, et nécéssite la lecture des clichés qui y sont associés, ainsi que la lecture du rapport médical.

Toutes ces données sont chargées en mémoire dans les objets correspondant afin de les rendre accessible aux autres opérations CRUD et bien sûr de permettre au docteur de retrouver les dossiers des patients facilement.

2.2.3 Ajout des données

L'ajout d'une radiographie ou d'un cliché se fait en deux étapes :

Tout d'abord, le programme écrit les données correspondantes à la nouvelle radiographie ou au nouveau cliché dans le fichier correspondant de la base de données, puis le programme relit toutes les données de toutes les radiographies afin de recomposer entièrement le vecteur de radiographie du docteur.

Lors du rechagement du vecteur de radiographie, toutes les modifications réalisées entre temps par les autres utilisteurs sur la base de données sont alors prise en compte, et sont donc visible par l'utilisateur courant.

2.2.4 Modification et suppression des données

La bibliothèque jsoncpp ne permettant pas de modifier des données directement dans le fichier json, la modification des données dans la base donnée demande de copier localement les données dans des variables du programme, de supprimer la donnée dans la base, puis d'ajouter la version modifier dans le fichier json.

Pour la suppression, la bibliothèque jsoncpp offre une routine pratique, permetant de supprimer des données depuis la base immédiatement dans le fichier comportant la donnée cible.

3 Diagramme UML

Le programme implémente une structure organisée en classe pour répondre aux contraintes de manipulation d'une base de données médicale pour le cabinet de radiologie. Ainsi, on y retrouve deux classes correspondant aux rôles utilisateurs différents : Patient et Docteur, ayant un ensemble de fonctionnalités eux aussi différentes.

Les docteurs et les patients ont donc tout deux accès à une collection de radiographies mais leurs actions possibles sont différentes.

Chaque radiographie est associée à une date, une collection de clichés et un rapport médical, mais ces deux derniers éléments sont uniquement accessible par le docteur grâce à des méthodes que le patient ne possède pas.

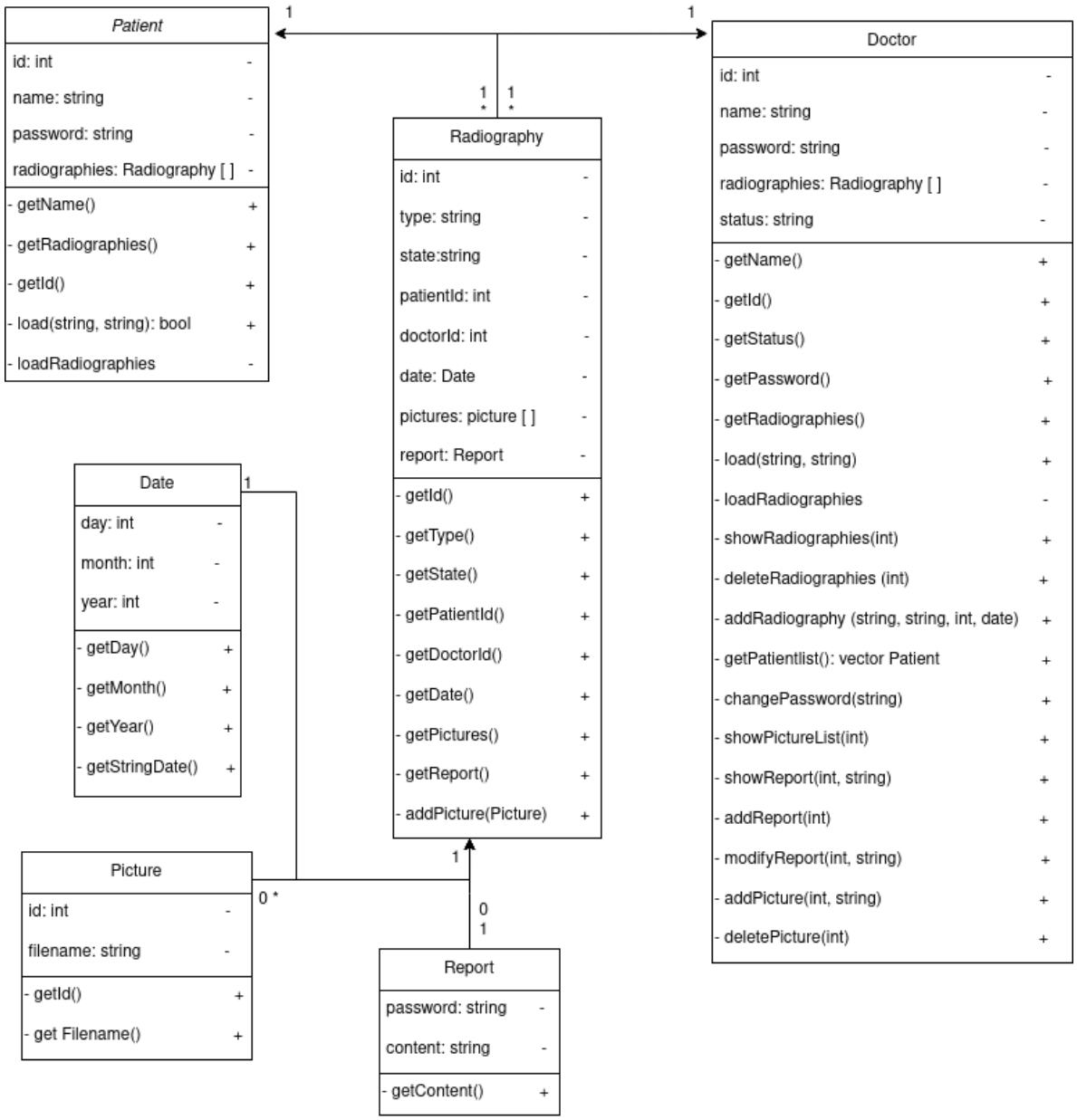

FIGURE 1 – Diagramme UML de notre application (les - correspondant au private d'une classe et les + au public).

4 Notice utilisateur

4.1 Installation et compilation

Le code source du programme est associé à ce rapport mais peut être téléchargé à nouveau via github : https ://github.com/Phloemus/M2-Projet-OOP/tree/main/final ou en cliquant içi.

Pour compiler le programme, il faudra installer la bibliothèque jsoncpp pour linux, puis compiler le code source fourni sous linux (ditribution ubuntu conseillée) avec g++.

Commande de compilation à exécuter :

g++ -c classes.cpp -ljsoncpp

g++ -o main main.cpp classes.o -ljsoncpp

N'oubliez pas de télécharger la base de données contenant le jeu de données présent dans le répertoire (cf /data sur github).

4.2 Execution du programme

Lors de l'exécution du programme vous aurez la possibilité de choisir votre type d'authentification. Deux utilisateurs sont disponible dans le jeu de donnée test : un patient et un docteur.

Le patient :

— nom : patient

— mot de passe : 1234

Le docteur :

— nom : docteur

— mot de passe : 1234

Ensuite l'interface graphique indique quelle fonctionnalité vous est disponible selon votre rôle.

5 Conclusion et limites

L'application propose une interface dynamique permettant de réaliser des opérations très classiques sur la base de données médicale, tout en adoptant un système de rôles et d'authentification, ainsi qu'une séparation des fonctionnalités de l'application en une structure rigide composée d'objets de différent types.

La structure de classe suit les principes de la programmation orientée objet et offre une solution qui pourrait être facilement améliorée avec de nouvelles fonctionnalités notamment pour rectifier les limites de ce projet.

En effet, malgré les nombreuses fonctionnalités que l'application propose, il reste certaines limites contraignantes pour une utilisation optimum. Il faudrait notamment ajouter un moyen de créer de nouveaux patients et de nouveaux docteurs dans la base de données par l'intermédiaire d'un nouveau rôle utilisateur : un administrateur.

De plus, d'un point de vue structurel, l'organisation en classes réalisée dans ce projet est perfectible. Par manque de connaissance et de temps, nous ne sommes pas parvenu à centraliser l'accès à la base de donnée via une classe dont la fonction serait exclusivement d'écrire et de lire dans la base de données. Les objets qui ont besoin de lire et d'écrire dans la base de données aurait pu hériter de cette classe permettant ainsi de réduire énormément la répétition de code et de respecter le paradigme de simple responsabilité (une classe n'a qu'une seul but) demandé par la programmation orienté objet.

La façon dont l'accès à la base de donnée a été réalisé doit aussi être repenser si l'application à pour vocation d'être utilisée par beaucoup de personnes simultanément. Il faudrait implémenter un système de versions des modifications et un processus de fusion des données avant leur écriture dans les fichiers consernés.

Bien que l'application soit perfectible, elle reste fonctionnelle et nous a permis de franchir un premier pas dans le paradigme de la programmation orientée objet utilisé transversalement dans un grand nombre de langages de programmation.