IsIMeD

Plan de conception d’entrepôt d’images

Tâche 4.3 - Livrable 4.3

Projet ANR-10-AIRT-07

b<>com
ZAC des Champs Blancs
1219 avenue Champs Blancs
35510 Cesson-Sévigné

Auteurs

Kai Wen	kai.wen@b-com.com
Amandine Le Maitre	amandine.lemaitre@b-com.com
Yannick Morvan	yannick.morvan@b-com.com

Propriétés du document

Nom du projet	IsIMeD
Titre du document	Plan de conception d'entrepôt d'images
Sous-titre du document	Tâche 4.3 - Livrable 4.3
Auteur	Kai Wen, Amandine Le Maitre, Yannick Morvan
Résumé	L'objectif de ce rapport est de définir un plan de conception pour un entrepôt d'images
Dernière édition	2015/06/26 à 15:29:00
Version	1.0
Diffusion	Confidentiel
Mots clés	

Données du document

Éditeur	Amandine Le Maitre
Courriel de l'éditeur	amandine.lemaitre@b-com.com
Adresse de l'éditeur	b<>com, ZAC des Champs Blancs, 1219 avenue Champs Blancs, 35510,Cesson-Sévigné
Date de livraison	2015-06-30

TABLE DES MATIÈRES

1 INTRODUCTION

Les systèmes de communication et d'archivage d'images médicales, dénommé Picture Archiving and Communication Systems, reposent sur le standard DICOM. Ce standard DICOM définit notamment un format de donnée des objets DICOM composés d'une entête, cette entête étant elle-même composée de couples clé-valeur. Ces couples clé-valeurs DICOM sont généralement indexées, intégrées dans une base de données relationnelle de type SQL. Un problème majeur relatif à ce type de base de données est que seul un nombre limité de champs DICOM peuvent être gérés. De plus, une difficulté additionnelle est que les champs DICOM employés varient grandement selon les contextes cliniques. Ils sont finalement difficilement classable et donc semi-structuré. Dans ce contexte, nous explorons de nouvelles approches permettant l'intégration et la gestion d'entrepôt d'images archivant de larges volumes d'objets DICOM. Nous investiguons en particulier deux aspects. Le premier aspect est l'utilisation de base de données orientée document permettant notamment l'intégration de tous les champs DICOM semi-structurés et hétérogènes. Deuxièmement, nous explorons l'utilisation d'une de ces bases de données (MongoDB) pour intégrer et indexer des object DICOM archivés sur un disque. Pour cela trois étapes sont nécessaires :

1. conception d'un système de notification de fichiers basé sur Inotify de Linux,
2. analyse syntaxiques des fichiers DICOM sur le disque, puis
3. stockage et indexation des données dans une base de données.

2 BASES DE DONNÉES D'OBJETS DICOM

Nous investiguons dans ce chapitre l'utilisation de base de données permettant l'accès à de larges volumes d'images DICOM. Dans ce contexte, deux familles de base de données peuvent être utilisées : (1) une base de données hiérarchique de type SQL ou (2) une base de données orientée documents dite 'NOSQL'. Nous présentons donc maintenant les propriétés d'une base de données, que sont l'atomicité, l'isolation, la consistance et la durabilité. Nous discutons par la suite des caractéristiques de ces deux familles et discutons de leur adéquation à la gestion d'images DICOM.

2.1 Propriété d'Atomicité de Consistance d'Isolation et de Durabilité (ACID)

En informatique, les bases de données, SQL ou NOSQL peuvent présenter des propriétés d'Atomicité, d'Isolation, de Consistance et de Durabilité (ACID). Ces propriétés sont à prendre en compte vis-à-vis de l'intégration des données DICOM. En informatique, une transaction est dite :

'atomique' lorsque que la base de données exécute des transactions dites 'atomique', c'est à dire que les requêtes/transactions sont soit effectuées totalement ou pas du tout,

'isolée' lorsque que la base de données exécute des transactions isolées les unes des autres et aucune relation ou dépendance ne devrait exister entre ces transactions,

'durable' lorsque les données de la base sont préservées lorsque la transaction est confirmée, même en cas de défaillance du système,

'cohérente' lorsque la base de données exécute une transaction qui modifie les données de manière autorisé, c'est à dire en accord avec les règles définies.

2.2 Caractéristique des données DICOM

Un objet DICOM est constitué d'un entête binaire suivi des données également binaires. Cet entête se compose de champs d'information associant des valeurs (e.g. le nom du patient) à des clés (e.g. '0010,0010'). En pratique, ces champs DICOM présente les caractéristiques suivantes.

Champs DICOM hiérarchiques Le standard DICOM définit la relation hiérarchique suivante :

1. à chaque patient est associé un ou plusieurs examens ('DICOM studies'),
2. à chaque examen (par exemple un examen TEP/TDM) est associé une ou plusieurs séries d'images ('DICOM series'),
3. à chaque série d'images est associé un ou plusieurs objets DICOM ('DICOM instance'), par exemple les différentes coupes d'un scanner.

En conséquence, une base de données permettant de gérer de manière efficace les données hiérarchiques est désirable.

Champs DICOM hétérogènes/semi-structurés Une seconde caractéristique est qu'une *partie des champs DICOM* n'est pas structurée ni hiérarchisée. En pratique, ces champs sont listés de manière linéaire, si bien qu'une organisation des données en tables est inappropriée. Une approche possible est d'insérer chaque champ DICOM dans une table de grande taille. Nous pouvons discerner deux difficultés relatives à cette approche. Premièrement, la liste de champs employée dépend du type d'examen, du contexte de l'acquisition, certains champs étant optionnels. Une table de champs DICOM, même de grande taille ne peut être déterminée par avance. Deuxièmement, des champs dit 'privés' sont régulièrement employés. La signification de ces champs est privée, c'est à dire 'propriétaire' et donc non inter-opérable. Encore une fois, ce type de champ est donc non classable dans une table de données prédéfinie.

Les champs DICOM présentent donc diverses caractéristiques gérées de façon différente par les deux familles de base de données : SQL et NOSQL.

2.3 Bases de données SQL

De manière traditionnelle, les Picture Archiving Communication Systems sont basés sur des bases de données relationnelles de type SQL. Ces bases de données requièrent une structuration ainsi qu'un modèle fixe de schéma de données composé de tables. En pratique, les données sont insérées dans les différentes tables (ex : table de données patient ou image). L'avantage majeur de structurer les données est la possibilité d'optimiser les accès, les sélections et les jointure de données. Cependant, ce type de modèles de données impose une sélection des informations. Dans le cas d'un objet DICOM, les informations sélectionnées sont par exemple le nom du patient, les UIDs de 'study, 'series' ou encore 'instance'. Dans certains cas, le concepteur de la base de donnée insère les informations restantes dans un 'Binary Large Object'. La conséquence principale de cette sélection est que les requêtes SQL se concentrent sur la sélection de données et des recherches basées sur d'autres critères deviennent impossibles. Il est évidemment possible de stocker toutes ces informations restantes dans une table dédiée, composée de plusieurs centaines de lignes. Cette stratégie n'est cependant pas réaliste pour deux raisons. Premièrement, la création d'une table de très grande taille réduirait de manière très significative les performances du système. Deuxièmement, la structure d'une table SQL reste fixe et ne peut donc être adaptée dynamiquement à de nouveaux types d'information (nouveau DICOM tag).

2.4 Bases de données NOSQL

Une approche alternative est d'utiliser une base de données orientée document de type NOSQL [1]. Cette famille de base de données NOSQL a graduellement émergée ces dernières années et inclue notamment BigTable (Google), Cassandra (Twitter), CouchDB (Apache Foundation) ou MongoDB (MongoDB Inc.). Nous discutons maintenant des propriétés importantes de ce type de base de données.

Schéma de données non fixe Une propriété importante des bases de données orientées document est que le schéma de données peut-être adapté de manière dynamique. Considérant l'intégration des champs DICOM, un tel schéma dynamique permet en particulier l'intégration de champs DICOM variés, ou même privés.

Hierarchical versus non hierarchical (nested data) Les bases de données NOSQL orientées document permettent généralement de hiérarchiser et créer des collections de données. Il est donc par exemple envisageable de créer un document 'DICOM series' contenant une collection de données relatives aux images de cette série.

Transactions atomiques Un nombre important de bases de données NOSQL balance la consistance des données avec la disponibilité. Par exemple, il est nécessaire de pouvoir empêcher l'accès à des données en cours de modification (le cas d'école étant la modification du solde sur un compte bancaire). Dans le cas de données DICOM, une base de données NOSQL ne permet pas de transaction atomique. Il n'est cependant pas envisagé de modifier les champs DICOM.

Unicité des données et index Dans certains cas, les données DICOM seront insérées de manière multiple dans la base de données. En pratique, les UIDs de ces objets devraient être utilisés pour empêcher de multiples insertions du même objet.

3 SYSTÈME D'INDEXATION D'IMAGES DICOM

3.1 Architecture système

Notre système est composé de deux parties (voir figure 3.1). La première partie est un système de notification qui surveille un répertoire prédéfini. Quand un fichier DICOM est ajouté ou supprimé dans ce répertoire, le système avertit l'utilisateur de cette modification et démarre les autres opérations comme par exemple l'analyse des fichiers DICOM. La seconde partie concerne l'analyse, le stockage et l'indexation. Lorsque le système démarre, il parcourt le répertoire prédéfini, cherche et analyse les objets DICOM, stocke les données analysées et crée un index sur ces données. Grâce à la coopération de ces deux parties, un grand nombre de fichiers DICOM peut être traité.

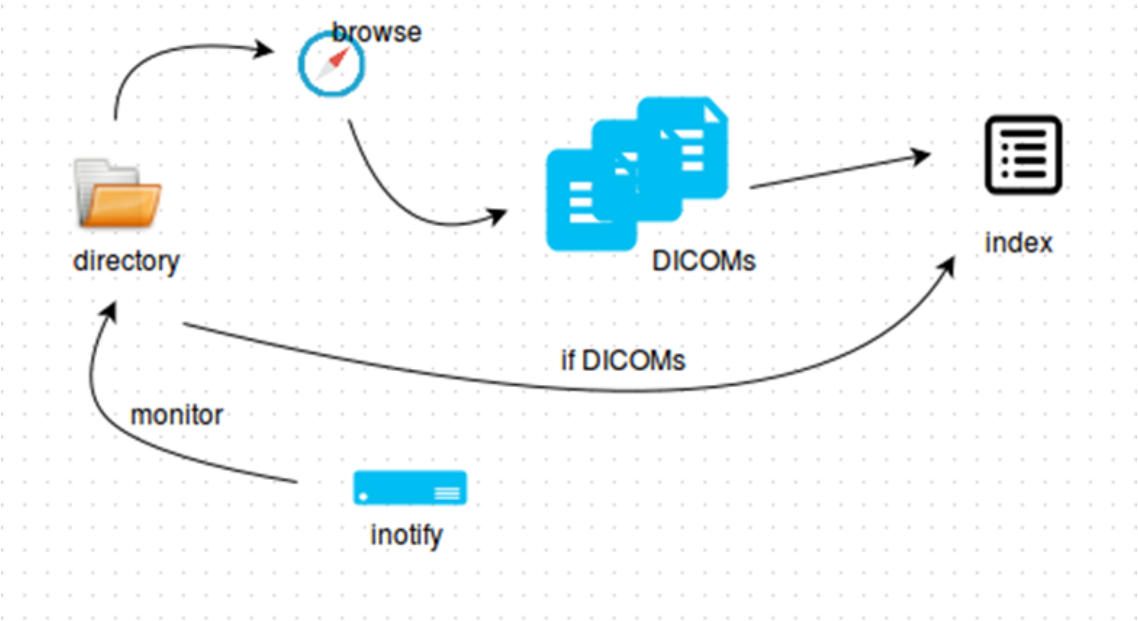


Figure 3.1 – Présentation globale du système

3.2 Implémentation

L'idée principale est de récupérer les métadonnées des objets DICOM puis d'enregistrer et d'indexer ces données dans une base de données de MongoDB [2] (voir figure 3.2). Pour analyser les fichiers DICOM, la librairie Rest DICOM library faisant l'objet des livrables "Tâche 2.2, livrables 2.3 et 2.4" est utilisée. Nous utilisons le drive C++ de MongoDB afin d'implémenter le stockage et l'indexation.

Figure 3.2 – Utilisation de MongoDB pour indexer des fichiers DICOM

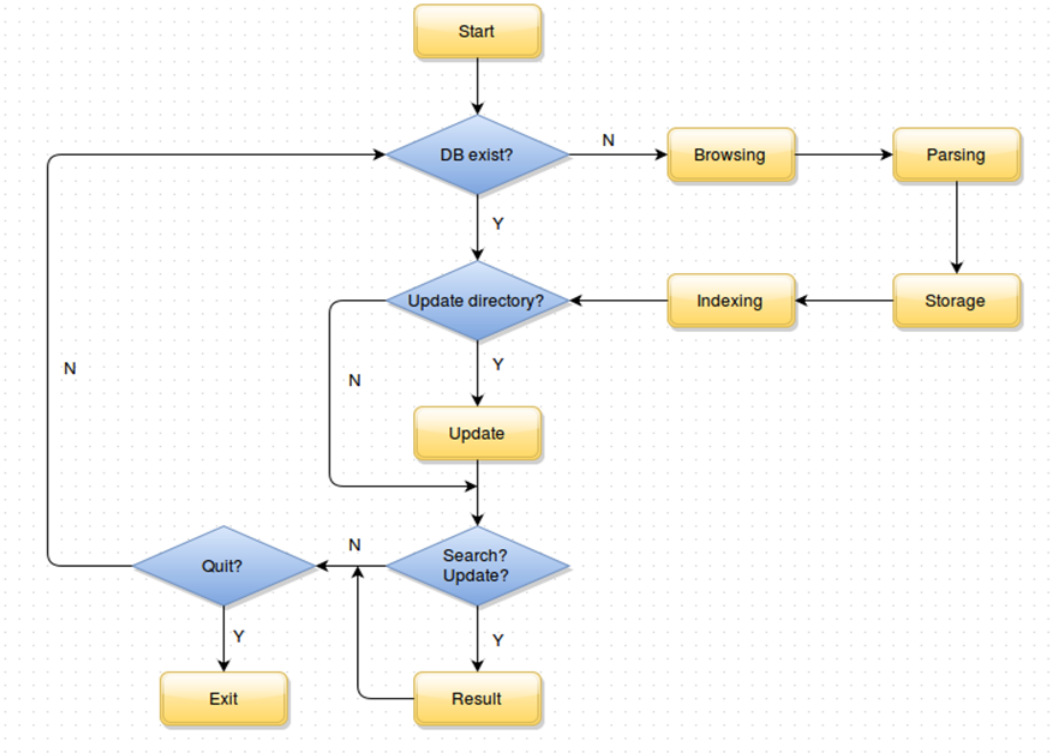


Figure 3.3 – Framework

3.2.1 Stockage, indexation et recherche des fichiers DICOM

La figure 3.3 présente le framework global du système. Lorsque le système démarre, l'utilisateur précise si le répertoire fourni est déjà indexé ou non. S'il n'est pas indexé le système scanne le répertoire prédéfini, recherche les objets DICOM qu'il contient (y compris ceux des sous répertoires) et stocke les chemins vers ces fichiers. Ces objets sont ensuite parsés à l'aide du parseur DICOM de la Rest DICOM library. Un objet DICOM est constitué d'une liste de *Data Element*. Chaque *Data Element* est constitué d'un *Tag*, d'une *Value Representation (VR)*, d'une *Value Length (VL)* et d'une *Value*. Le tag est un code représentant l'élément, la *VR* indique la façon dont il est encodé, la *VL* représente la longueur de la valeur de l'élément. Dans notre cas nous nous intéressons au *Tag*, et à la *Value* de ces *Data Elements*. Lorsque la *VL* est trop importante, nous ne stockons que les premiers caractères afin d'optimiser l'espace de stockage. La structure de stockage proposée par MongoDB est un couple "Key-Value". Pour les objets DICOM nous avons choisi d'utiliser le Keyword, qui est une description unique du Tag, comme Key et la Value du Data Element comme Value. Pour les Tags privés, le Keyword est inconnu. Nous utilisons donc une chaine de caractère représentant le tag (ex : "2005,0010"). En plus des métadonnées DICOM, le chemin vers le fichier est également stocké. La figure 3.4 présente un exemple de stockage dans MongoDB d'un fichier DICOM. Certains *Data Element* sont des séquences ($VR = SQ$). Il s'agit en fait d'éléments dont la *Value* est, elle-même, une liste de *Data Element*. Afin de stocker ces éléments nous utilisons le stockage de document embarqué de MongoDB. La figure 3.5 présente un exemple de stockage d'une séquence.

```
> db.test.findOne({FilePath: "/home/kaiwen/Dicom_Image/888.888.1.19950627.121647.1.102"})
{
        "_id" : ObjectId("554777c78000fc0bf155c8a6"),
        "FileMetaInformationVersion" : "0\\1",
        "MediaStorageSOPClassUID" : "1.2.840.10008.5.1.4.1.1.12.1",
        "MediaStorageSOPInstanceUID" : "888.888.1.19950627.121647.1.102",
        "TransferSyntaxUID" : "1.2.840.10008.1.2.4.70",
        "ImplementationClassUID" : "1.2.40.0.13.1.1",
        "ImplementationVersionName" : "dcm4che-2.0",
        "ImageType" : "ORIGINAL\\PRIMARY\\SINGLE PLANE",
        "SOPClassUID" : "1.2.840.10008.5.1.4.1.1.12.1",
        "SOPInstanceUID" : "888.888.1.19950627.121647.1.102",
        "StudyDate" : "19950627",
        "ContentDate" : "19950627",
        "StudyTime" : "121647",
        "ContentTime" : "",
        "AccessionNumber" : "",
        "Modality" : "XA",
        "Manufacturer" : "Acme Products",
        "InstitutionName" : "AKH Wien",
        "InstitutionAddress" : "Department of Cardiology, 1090 Vienna, Austria",
        "ReferringPhysicianName" : "Porenta",
        "StudyDescription" : "Cardiac Catheterization",
        "SeriesDescription" : "Diagnostic Catheterization",
        "PerformingPhysicianName" : "Schmidinger",
        "PatientName" : "R^H",
        "PatientID" : "000-63-7652",
        "PatientBirthDate" : "19390101",
        "PatientSex" : "M",
        "ContrastBolusAgent" : "",
        "KVP" : "",
        "FrameTime" : "40",
        "Exposure" : "",
        "RadiationSetting" : "GR",
        "PositionerMotion" : "STATIC",
        "PositionerPrimaryAngle" : "-88",
        "PositionerSecondaryAngle" : "-5",
        "StudyInstanceUID" : "888.888.1.19950627.121647",
        "SeriesInstanceUID" : "888.888.1.19950627.121647.1",
        "StudyID" : "1995-35",
        "SeriesNumber" : "1",
        "InstanceNumber" : "102",
        "PatientOrientation" : "",
        "ImageComments" : "Right coronary",
        "SamplesPerPixel" : "1",
        "PhotometricInterpretation" : "MONOCHROME2",
        "NumberOfFrames" : "209",
        "FrameIncrementPointer" : "24\\4195",
        "Rows" : "512",
        "Columns" : "512",
        "BitsAllocated" : "8",
        "BitsStored" : "8",
        "HighBit" : "7",
        "PixelRepresentation" : "0",
        "PixelIntensityRelationship" : "LIN",
        "CalibrationImage" : "",
        "FilePath" : "/home/kaiwen/Dicom_Image/888.888.1.19950627.121647.1.102"
}
```


Figure 3.4 – Stockage d'un objet DICOM dans MongoDB

```
"GraphicAnnotationSequence" : {
        "GraphicAnnotationSequence0" : {
                "ReferencedImageSequence" : {
                        "ReferencedImageSequence0" : {
                                "ReferencedSOPClassUID" : "1.2.840.10008.5.1.4.1.1.4.1",
                                "ReferencedSOPInstanceUID" : "1.3.46.670589.11.1.5.20.1.1.5176.2008021217315304736",
                                "ReferencedFrameNumber" : "1",
                                "(2001,0010)" : "Philips Imaging DD 001",
                                "(2001,10c1)" : "ImageReference"
                        }
                },
                "GraphicLayer" : "1",
                "TextObjectSequence" : {
                        "TextObjectSequence0" : {
                                "BoundingBoxAnnotationUnits" : "PIXEL",
                                "AnchorPointAnnotationUnits" : "PIXEL",
                                "UnformattedTextValue" : "263",
                                "BoundingBoxTopLeftHandCorner" : "103.659\\83.5203",
                                "BoundingBoxBottomRightHandCorner" : "142.659\\95.5203",
                                "BoundingBoxTextHorizontalJustification" : "LEFT",
                                "AnchorPoint" : "103.659\\89.5203",
                                "AnchorPointVisibility" : "N",
                                "(2001,0010)" : "Philips Imaging DD 001",
                                "(2001,106d)" : "--13-*-0-0-medium-0-0-0-*-*-*-*-*-*-Philips Screen Font 2-",
                                "(2001,10a3)" : "4294902531"
                        }
                },
                "GraphicObjectSequence" : {
                        "GraphicObjectSequence0" : {
                                "GraphicAnnotationUnits" : "PIXEL",
                                "GraphicDimensions" : "2",
                                "NumberOfGraphicPoints" : "5",
                                "GraphicData" : "202.263\\102.27\\103.659\\102.27\\103.659\\192.079\\202.263\\192.079\\202.263\\102.27",
                                "GraphicType" : "POLYLINE",
                                "GraphicFilled" : "N",
                                "(2001,0010)" : "Philips Imaging DD 001",
                                "(2001,1046)" : "SOLID",
                                "(2001,1047)" : "0",
                                "(2001,104b)" : "LINEAR",
                                "(2001,1055)" : "4294902531",
                                "(2001,1056)" : "POLYGON",
                                "(2001,1071)" : "RECTANGLE",
                                "(2001,109b)" : "0"
                        }
                },
                "(2001,0010)" : "Philips Imaging DD 001",
                "(2001,1048)" : "2",
                "(2001,109c)" : "263"
        }
},
```


Figure 3.5 – Stockage d'une séquence dans MongoDB

Une fois que les données sont stockées nous créons un index afin de faciliter la recherche. A la création de la base de données, un index est créé automatiquement sur le champ *_id*. Cependant celui-ci n'est pas suffisant pour nos besoins. MongoDB fournit trois méthodes d'indexation : *Single Field Indexes*, *Compound Indexes* et *Text Indexes*. Afin de pouvoir faire des recherches sur les valeurs sans le nom de champ un index de type Full-Text est créé. Toutes nos valeurs sont stockées sous forme de chaîne de caractères ce qui est un prérequis pour les *Text Indexes* de MongoDB. Nous indexons ensuite les champs PatientName, PatientID, StudyID, SeriesNumber, InstanceNumber et FilePath par un *Single Field Index* et un *Compound Index*. La performance de l'indexation sera étudiée dans la section 3.3.

Une fois que les données sont indexées on peut faire des recherches sur celles-ci. Trois fonctionnalités de recherche, liées à notre indexation, sont possibles : (1) Single Field Search, (2) Multiple Field Search et (3) Full-Text Search. La Single Field Search fonctionne sur un tag indiqué par l'utilisateur. Nous recherchons par exemple une donnée dont le PatientName est DUPONT. La Multiple Field Search permet de rechercher plusieurs Tag. On recherche par exemple des informations sur un examen d'un patient. La Full-Text Search permet de faire des recherches sur les valeurs directement sans indiquer le nom du champ.

Une fonctionnalité a été ajoutée permettant de mettre à jour ou de corriger des informations administratives relatives au patient. Par example, si le nom d'un même patient a été orthographié de deux façons différentes dans deux institutions il est possible de fusionner les deux PatientName afin de garantir la cohérence des données. Pour cela on remplace le champ PatientName par le nom correctement orthographié pour les deux objets et un nouveau champ OldPatientName est ajouté afin de garder la référence vers le nom modifié.

La figure 3.6 présente le menu lors du démarrage de notre application. Les informations sur les méthodes de recherche possibles ainsi que la fonctionnalité de mise à jour des données sont affichées.

```
****************** Operations ******************
* Full Search: Word, Word1&Word2 or Word1|Word2  *
* Field Search: Field:Word                       *
* Fields Search: Field1:Word1;Field2:Word2...    *
* Document Update: Field//Old//New              *
* Show Operations' Menu: m/M                    *
* Help: help/Help                               *
* Exit: exit/Exit                               *
************************************************
>>>>>>>>>>>>>>>
```

Figure 3.6 – Menu principal lors du démarrage de l'application

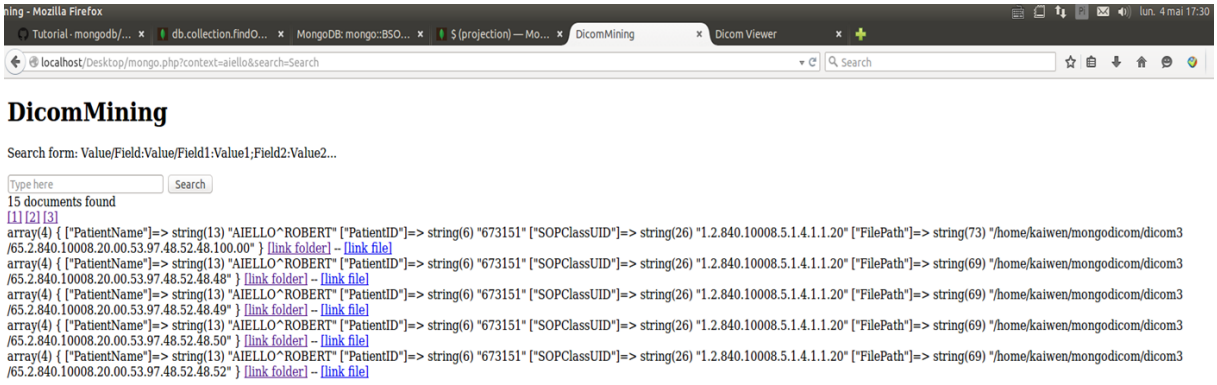


Figure 3.7 – Interface pour interroger la base de données

3.2.2 Interface de recherche d'objets DICOM

MongoDB offre une interface REST pour interroger la base de données. Ceci permet de créer facilement une page web de recherche dans notre base de données. Sur la figure 3.7 est illustrée une page simple permettant de faire les trois types de recherches dans la base de données.

3.2.3 Système de notification

Pour la notification nous utilisons Inotify du système Linux. Lorsque le répertoire est scanné, les chemins vers les fichiers DICOM sont stockés ainsi que les chemins vers le répertoire prédéfini et les sous répertoires contenus dedans. Au démarrage du système, la surveillance est également lancée à l'aide des APIs Inotify. Un *File Descriptor* est présent par répertoire. Les évènements qui nous intéressent dans notre cas sont la suppression et la création de fichiers DICOM. Lorsqu'un nouveau fichier DICOM est ajouté, le système enregistre le chemin vers ce fichier puis parse le fichier et stocke et indexe les données par la procédure décrite précédemment. Si un nouveau répertoire est créé, le chemin vers ce répertoire est ajouté à la liste des répertoires et la liste de surveillance est réinitialisée. Lorsqu'un objet DICOM est supprimé, le système cherche les données correspondantes à cet objet et les supprime. Si un répertoire est supprimé, le système annule la surveillance sur ce répertoire et sur tous les sous répertoires qu'il contient. La suppression d'un répertoire déclenche la fonction de suppression décrite ci-dessus pour tous les fichiers contenus dans le répertoire. Les performances du système de notification seront étudiées dans la section 3.4.

3.3 Performance du système d'indexation

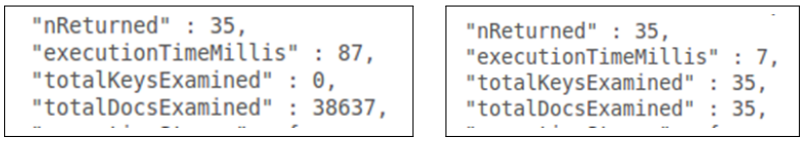
3.3.1 Temps d'indexation et de parsing de fichiers DICOM

Afin de tester notre système nous avons indexé 38637 fichiers DICOM. Ces fichiers sont placés dans un répertoire contenant 587 sous répertoires. Lors du démarrage de notre système le répertoire est scanné et la totalité de ces fichiers est parsée, stockée et indexée. Le temps total de lancement de notre système est de 2704.73 secondes. Ce temps global comprend : (1) le temps de scannage qui est de 445.23 secondes, (2) le temps de parsing des objets DICOM qui est de 2243.97 secondes et (3) le temps de stockage et d'indexation qui est de 15.53 secondes. C'est le parsing des objets DICOM qui est le plus lent. Nous avons essayé, afin d'améliorer ce temps de parsing, de ne pas considérer la totalité des Data Element contenus dans les objets DICOM mais seulement ceux qui sont les plus utilisés. La liste de Tag considérée est celle correspondant aux Tags renvoyés lors d'une requête QIDO-RS (voir Tâche 2.1 – livrable 2.1). Il s'agit de : SpecificCharacterSet, StudyDate, StudyTime, AccessionNumber, ModalitiesInStudy, ReferringPhysicianName, PatientName, PatientID, PatientBirthDate, PatientSex, StudyInstanceUID, StudyID, Modality, SeriesDescription, SeriesInstanceUID, SeriesNumber, PerformedProcedureStepStartDate, PerformedProcedureStepStartTime, SOPClassUID, SOPInstanceUID, InstanceNumber, Rows, Columns, NumberOfFrames et TransferSyntaxUID. En ne considérant que ces tags, le temps de parsing est réduit à 1336.65 secondes.

3.3.2 Temps de recherche de fichiers DICOM avec et sans index

Afin d'évaluer les temps de recherche dans la base de données, les trois paramètres importants sont : (1) la quantité de résultats, (2) la quantité de champs d'index scannés et (3) la quantité de documents scannés. En temps normal la tendance est la suivante : la quantité de champs d'index scannés est supérieure à celle de documents scannés qui est supérieure à celle de résultats. Si les trois quantités sont égales, l'index est parfait. Nous avons fait nos tests de recherche sur une base de données contenant 38637 documents.

Pour la Single Field Search nous faisons une recherche de "CHILI" sur le champ PatientName. Si le champ n'est pas indexé, la quantité de documents scannés est de 38637, soit tous les documents. Le temps de recherche est de 87ms. Si l'on indexe ce champ, la quantité de documents scannés est égale à celle de champs scannés qui est égale au nombre de résultats, soit 35. Le temps de recherche est réduit à 7ms (voir figure 3.8).

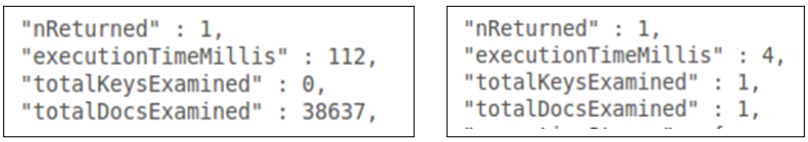


(a) (b)

Figure 3.8 – Temps de recherche d'un champ (a) avec et (b) sans index

Pour la Multiple Field Search nous faisons une recherche de "CHILI" et "6" sur les champs PatientName et SeriesNumber. Sans le Compound Index le nombre de documents scannés est de 38637, soit le nombre total de documents. Le temps de recherche est de 112ms. Si les champs PatientName et SeriesNumbre sont indexés, le nombre de documents scannés est égal au nombre de champs scannés qui est égal au nombre de résultats soit 1. Le temps de recherche est réduit à 4ms (voir figure 3.9).

Pour la Full Text Search il n'est pas possible de faire de recherche sans indexation. Par contre, si l'on veut faire une recherche sur un champ spécifique qui n'est pas indexé, la recherche Full Text est plus efficace que la recherche Single Field. Le nombre de champs d'index et de documents scannés est moins important.



(a) (b)

Figure 3.9 – Temps de recherche de deux champs (a) avec et (b) sans index

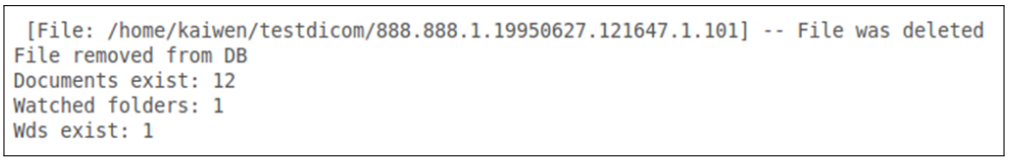


Figure 3.10 – Suppression d'un fichier DICOM avec Inotify

3.4 Performance du système de notification

Nous étudions dans cette section la performance du système de notification. Nous testons les scénarios de création et de suppression de fichiers DICOM ou de répertoire. Le répertoire principal testé se nomme "testdicom", il contient au début 13 fichiers DICOM indexés. La figure 3.10 illustre la suppression de l'image "888.888.1.19950627.121647.1.101". Lorsque ce fichier est ensuite recherché dans la base de données, celle-ci ne retourne aucun résultat. Le même fichier est ensuite remis dans le répertoire et réindexé avec succès (voir figure 3.11). Le système fonctionne toujours lors de l'ajout de plusieurs fichiers. La figure 3.12 illustre la création d'un nouveau répertoire "test", contenant deux objets DICOM, dans le répertoire "testdicom". Les fichiers contenus dans ce nouveau répertoire sont ajoutés et indexés avec succès. Le sous répertoire "test" a bien été ajouté la surveillance. La figure 3.13 illustre la suppression de ce même répertoire. Les deux fichiers DICOM contenus dedans sont bien supprimés de la base de données.

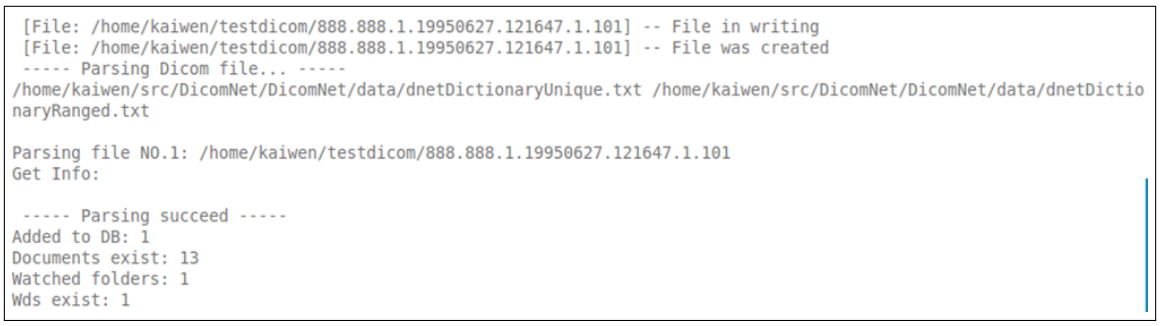


Figure 3.11 – Création d'un fichier DICOM avec Inotify

```
 [Directory: /home/kaiwen/testdicom/test/] -- Directory was created
wd build --- [OK]

 [File: /home/kaiwen/testdicom/test/888.888.1.19950627.121647.1.101] -- File was created
 [File: /home/kaiwen/testdicom/test/888.888.1.19950627.121647.1.102] -- File in writing
 [File: /home/kaiwen/testdicom/test/888.888.1.19950627.121647.1.102] -- File was created
 ----- Parsing Dicom file... -----
/home/kaiwen/src/DicomNet/DicomNet/data/dnetDictionaryUnique.txt /home/kaiwen/src/DicomNet/DicomNet/data/dnetDictio
naryRanged.txt

Parsing file NO.1: /home/kaiwen/testdicom/test/888.888.1.19950627.121647.1.102
Get Info:

Parsing file NO.2: /home/kaiwen/testdicom/test/888.888.1.19950627.121647.1.101
Get Info:

 ----- Parsing succeed -----
Added to DB: 2
Documents exist: 15
Watched folders: 2
Wds exist: 2
```

Figure 3.12 – Création d'un répertoire contenant des fichiers DICOM avec Inotify

```
 [File: /home/kaiwen/testdicom/test/888.888.1.19950627.121647.1.101] -- File was deleted
 [File: /home/kaiwen/testdicom/test/888.888.1.19950627.121647.1.102] -- File was deleted
 [Directory: /home/kaiwen/testdicom/test/] -- Directory was deleted
Folder removed
File removed from DB
File removed from DB
Documents exist: 13
Watched folders: 1
Wds exist: 1
```

Figure 3.13 – Suppression d'un répertoire contenant des fichiers DICOM avec Inotify

4 CONCLUSION

Nous avons étudié dans ce rapport l'apport de l'utilisation d'une base de données de type NOSQL pour l'indexation et le stockage des objets DICOM. Ce type de bases de données devraient permettre de s'adapter au caractère semi-structuré des champs DICOM. Nous avons plus particulièrement étudié l'utilisation de MongoDB pour l'indexation d'objets DICOM.

BIBLIOGRAPHIE

[1] Simón J. Rascovsky, Jorge A. Delgado, Alexander Sanz, Víctor D. Calvo, and Gabriel Castrillón. Informatics in radiology : use of CouchDB for document-based storage of DICOM objects. *Radiographics : A Review Publication of the Radiological Society of North America, Inc*, 32(3) :913–927, June 2012.

[2] MongoDB. *Accessed June 2015*. URL : `https://www.mongodb.org/`.