### ****Document technique pour l'implémentation d'une plateforme avancée de gestion documentaire****

### ****1. Vue d'ensemble du projet****

Ce document expose une **conception technique avancée** pour le développement d'une **plateforme de gestion documentaire**.

* CRUD des documents
* Gestion des fichiers (ajout, visualisation)
* Versionnage avancé des documents
* Recherche basée sur les métadonnées
* Historique des modifications
* Gestion des dossiers hiérarchiques
* Validation dynamique des règles métiers via **DMN (Decision Model and Notation)**

### ****2. Structure des entités (Domain Model)****

Voici les **entités principales** nécessaires pour représenter les concepts métiers de la plateforme.

#### **2.1. Documentation des entités**

##### Document

Cette entité représente un document dans le système avec son contenu logique, ses métadonnées et des attributs spécifiques.

@Entity

public class Document {

@Id

@GeneratedValue

private Long id; // Identifiant unique du document

private String title; // Titre descriptif du document

private String description; // Description ou résumé du document

private String mimeType; // Type MIME du document (PDF, DOCX, etc.)

@Enumerated(EnumType.STRING)

private DocumentStatus status; // Statut du document (ACTIVE, ARCHIVED ou OBSOLETE)

@ElementCollection

private Map<String, String> metadata; // Métadonnées optionnelles (tags, auteur, etc.)

private String createdBy; // Créateur initial du document

private String lastModifiedBy; // Dernier utilisateur ayant modifié le document

private LocalDateTime createdOn; // Date de création du document

private LocalDateTime lastModifiedOn; // Date de dernière modification du document

@OneToMany(mappedBy = "document", cascade = CascadeType.ALL, orphanRemoval = true)

private List<DocumentVersion> versions; // Liste des versions associées au document

@ManyToOne

private Folder folder; // Le dossier auquel le document est associé

}

##### DocumentVersion

Chaque version d'un document est encapsulée dans cette entité.

@Entity

public class DocumentVersion {

@Id

@GeneratedValue

private Long id; // Identifiant unique de la version

private String fileName; // Nom du fichier original

private Long size; // Taille du fichier en octets

private String checksum; // Hachage pour garantir l'intégrité des données

private String fileUri; // URI ou chemin de stockage du fichier

private String createdBy; // Créateur de cette version du document

private LocalDateTime createdOn; // Date de création de cette version

@ManyToOne

private Document document; // Document auquel cette version appartient

@ManyToOne

private File file; // Contenu physique du fichier (lien vers une entité File)

}

##### File

Cette entité désigne le fichier physique associé à une version d'un document.

@Entity

public class File {

@Id

@GeneratedValue

private Long id; // Identifiant unique du fichier

@Lob

private byte[] data; // Contenu binaire du fichier (si stockage local)

private String storagePath; // Chemin de stockage (local ou distant)

private String storageType; // Type de stockage utilisé (CLOUD, LOCAL, etc.)

private Long size; // Taille du fichier

private String checksum; // Hachage SHA-256 pour validation d'intégrité

private LocalDateTime uploadedOn; // Date d'upload du fichier

}

##### Folder

Représentation d'un dossier contenant des documents ou des sous-dossiers.

@Entity

public class Folder {

@Id

@GeneratedValue

private Long id; // Identifiant unique du dossier

private String name; // Nom du dossier

private String createdBy; // Utilisateur ayant créé le dossier

private LocalDateTime createdOn; // Date de création du dossier

@ManyToOne

private Folder parentFolder; // Référencement du dossier parent (pour l'arborescence)

@OneToMany(mappedBy = "folder", cascade = CascadeType.ALL, orphanRemoval = true)

private List<Folder> subFolders; // Liste des sous-dossiers attachés

@OneToMany(mappedBy = "folder", cascade = CascadeType.ALL, orphanRemoval = true)

private List<Document> documents; // Liste des documents dans ce dossier

}

##### AuditLog

Traçabilité de toutes les actions effectuées dans le système.

@Entity

public class AuditLog {

@Id

@GeneratedValue

private Long id; // Identifiant unique du log

@Enumerated(EnumType.STRING)

private AuditAction action; // Type d'action (CREATE, UPDATE, DELETE, etc.)

private String user; // Nom de l'utilisateur ayant effectué l'action

private String details; // Détails supplémentaires de l'action

private LocalDateTime timestamp; // Date et heure de l'action

@ManyToOne

private Document document; // Document affecté par l'action

}

### ****2.2. Enumérations utilisées****

public enum DocumentStatus {

ACTIVE, // Document actif

ARCHIVED, // Document archivé

OBSOLETE // Document obsolète

}

public enum AuditAction {

CREATE, // Création d'un nouvel élément

UPDATE, // Modification d'un élément

DELETE, // Suppression d'un élément

CONSULT, // Consultation d'un élément

MOVE, // Déplacement d'un document entre dossiers

UPLOAD\_NEW\_FILE // Upload d'un fichier

}

#### **3. DTOs (Data Transfer Objects)**

Les DTOs permettent de simplifier les échanges de données entre la couche présentation et la couche service. Échantillon :

### ****1. DTO pour**** DocumentVersion

Le DTO pour les versions de documents inclut uniquement les informations nécessaires à l'utilisateur. Exemple : informations sur le fichier stocké et les métadonnées liées à la version.

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

public class DocumentVersionDTO {

private Long id; // Identifiant unique de la version

private String fileName; // Nom original du fichier

private Long size; // Taille du fichier en octets

private String checksum; // Hachage pour vérifier l'intégrité

private String fileUri; // URI du fichier (chemin de stockage ou URL)

private String createdBy; // Créateur de cette version

private LocalDateTime createdOn; // Date de création

}

### ****2. DTO pour**** File

Ce DTO est utilisé pour représenter les informations sur les fichiers associés aux versions des documents. Cela inclut les détails de stockage.

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

public class FileDTO {

private Long id; // Identifiant unique du fichier

private String storagePath; // Emplacement physique ou URL du fichier

private String storageType; // Type de stockage (CLOUD, LOCAL, etc.)

private Long size; // Taille du fichier en octets

private String checksum; // Hachage d'intégrité (SHA-256 ou MD5)

private LocalDateTime uploadedOn; // Date de téléchargement

}

### ****3. DTO pour**** Folder

Le DTO pour les dossiers inclut les informations nécessaires pour gérer la hiérarchie de dossiers et leur contenu (documents et sous-dossiers).

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

public class FolderDTO {

private Long id; // Identifiant unique du dossier

private String name; // Nom du dossier

private Long parentFolderId; // ID du dossier parent (s'il existe)

private String createdBy; // Utilisateur ayant créé le dossier

private LocalDateTime createdOn; // Date de création

private List<DocumentDTO> documents; // Liste des documents situés dans ce dossier

private List<FolderDTO> subFolders; // Liste des sous-dossiers

}

### ****4. DTO pour**** AuditLog

Le DTO pour les journaux d'audit est utilisé pour tracer et afficher les actions effectuées sur les documents et les dossiers.

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

public class AuditLogDTO {

private Long id; // Identifiant unique du log

private String user; // Nom de l'utilisateur ayant effectué l'action

private AuditAction action; // Type d'action effectuée (ex. CREATE, DELETE, UPDATE)

private String details; // Détails supplémentaires sur l'action

private LocalDateTime timestamp; // Date et heure de l'action

private Long documentId; // ID du document (s'il est concerné)

}

### ****5. DTO pour recherche avancée des métadonnées****

Ce DTO est utilisé pour filtrer et exposer les résultats d'une recherche avancée basée sur les critères de métadonnées.

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

public class MetadataSearchResultDTO {

private Long documentId; // Identifiant unique du document

private String title; // Titre du document

private String description; // Description du document

private String mimeType; // Type MIME du document

private DocumentStatus status; // Statut actuel (ACTIVE, ARCHIVED ou OBSOLETE)

private Map<String, String> metadata; // Métadonnées spécifiques au document

private LocalDateTime lastModifiedOn; // Dernière modification du document

}

Exemple pour les contrats de services ou usecases :

public interface DocumentService {

DocumentDTO createDocument(DocumentDTO documentDto);

DocumentDTO updateDocument(Long documentId, DocumentDTO documentDto);

void deleteDocument(Long documentId);

DocumentDTO getDocument(Long documentId);

}

### ****5. Diagrammes UML****

#### **Diagramme de Classes UML**

Voici le diagramme complet généré via PlantUML :

Liens pour visualiser :

<https://www.plantuml.com/plantuml/uml/bLF1Rjim3BtdAuGSqxNOha4Hr8rTB80R5ucJ7OQXK2qc5MPPXaKDo8Rz-monieHusL1VI9EF3u-aU6UGTsXLFQbgRWnBTMKLDCX-JLZ_vRevCYcSlyLEzdyeiOOu8C1KdMnHwYOEAwcWFBK4zRHRv6WDCywup2DlRwdY0va5Ku1SSEGnMzK1Hn3tfpZOAyP7BUH13fcqHtbmqOHGW0xnAvPBWxTUsXuwqxUmO3_8CM-W1rqBwDZ1cSdlIHXKN7PvNWTPmveh27aZVqACg5wX-cwiSh7TPkLATfqSzV-lBg_2wiIRunlesHnb4Ez2OyKl9uIltzZvx0tgZX_XYUFhE4mx_czlGRPjQys5qntqq8mlph3X2jup3K_OyWuQf9zuvnHPC6DVo3L_Un6BOP8cI4sia9ZhurWifP8AfMuOToRMR0rf26U3NDOcT81IWK6kselxx4L0OzNvDP6IP5ckzXcvc-Ndta_TJt6_BVAip8RgM2QLBZTPKbBfxYdrRfgvCWSerjjTNjB15diFpOiaVLvdNvuVLhdZ7oHFFqtPxFgwjpSti-deMERXoCvJeN9AptHERpTGVtIKscDeqtD_eFJlKWv2wA5w7dOwG0aoEoUC7iSTDCAg-Wy0>