Micro-service GED

4 répertoires seront nécessaires, ceux-ci seront défini par des variables dans les fichiers de configurations. “doc_dir“ : répertoire contenant les documents. “archive_dir“ : répertoire contenant les archives. “key_dir“ répertoire contenant les clefs de chiffrement.

Entités :

Media			
Champs	**Type**	**Nullable**	**Description**
uuid	uuid (char(36))	**false**	Identifiant de la ressource
name	varchar(256)	**false**	Nom du document
type	varchar(8)	**false**	Type du document
user	relation avec "User"	**false**	Utilisateur propriétaire
metadata	json	**true**	Données à enregistrer avec le document
group	relation avec "Group"	**true**	Group auquel appartient le document, tel un répertoire parent, si NULL le document est à la racine
createdAt	datetime	**false**	Date de création
updatedAt	datetime	**false**	Date de modification

Group			
Champs	**Type**	**Nullable**	**Description**
uuid	uuid (char(36))	**false**	Identifiant de la ressource
name	varchar (256)	**false**	Nom du groupe
user	relation avec "User"	**false**	Utilisateur propriétaire
parent	relation récursive avec "Group"	**true**	Group auquel appartient le group, NULL si à la racine
createdAt	datetime	**false**	Date de création
updatedAt	datetime	**false**	Date de modification

Archive			
Champs	**Type**	**Nullable**	**Description**
uuid	uuid (char(36))	**false**	Identifiant de la ressource
path	text	**false**	Chemin du document avant d'être archivé
metadata	json	**true**	Metadata du document avant d'être archivé
deletedAt	datetime	**false**	Date d'archivage

Gestion des fichiers et répertoires :

Tous les documents, qu'ils soient archivés ou non, devront être chiffrés. Les éléments du document en base de données : le nom, le type et les metadata du document, seront également chiffrés. De même, les éléments des archives : le path et les metadata, seront chiffrés. Ces données seront déchiffrées à chaque requêtes avant d'être envoyées à l'utilisateur.

Documents :

Chaque utilisateur possèdera un répertoire dans le répertoire "doc_dir", et n'aura connaissance que des documents s'y trouvant.

Tous les documents de l'utilisateurs seront stockés directement à la racine de ce répertoire et auront pour nom l'uuid de l'entité Media associé, sans extension.

Les groupes permettrons de créer une architecture hiérarchique tel qu'un système de répertoire. Ainsi, un document pourra appartenir à un groupe comme s'il serait dans un répertoire, groupe qui pourrait lui-même appartenir à un autre groupe, comme s'il serait lui-même dans un autre répertoire.
Les groupes et documents n'ayant pas de groupes seront considérés comme étant à la racine.

L'utilisateur verra donc le système hiérarchique ainsi créé, dans lequel chaque document aura pour nom le nom enregistré dans l'entité Media associé, et comme extension le type enregistré dans cette entité. La liaison avec le document enregistré dans le répertoire de l'utilisateur se fera par l'uuid de l'entité Media associé, puisqu'étant le nom du document.

Archives :

Il existera le répertoire "archive_dir" en parallèle du répertoire "doc_dir". Dans ce répertoire "archive_dir " il existera également un répertoire par utilisateurs, contenant chacun les différentes archives de l'utilisateur.
Chaque archive sera donc à la racine de ce répertoire et aura pour nom l'uuid de l'entité Archive associé.

Le champs "path" de l'entité Archive correspondra au chemin qu'avait le document avant d'être archivé. Ce chemin correspondra au chemin visible par l'utilisateur dans le modèle hiérarchique créé par les groupes.

Seuls les documents, donc les entités Media, pourront être archivés.

Endpoints :

Explorer				
Method	**URL**	**Body**	**Response**	**Description**
GET	/explorer /{path}		- Si le path correspond à un répertoire, ou est vide (correspond donc à la racine) : {"directories": [liste des répertoires dans le répertoire cible, avec pour chacun son nom et son uuid], "files": [liste des documents avec pour chacun son nom et son uuid]} - Si le path correspond à un document : {"uuid": uuid du document}	Permet de naviguer dans les répertoires stockés par le service

Media				
Method	**URL**	**Body**	**Response**	**Description**
POST	/file	{"content": document à upload en base64, "path": chemin du document dans l'arborescence, "metadata": metadata du document	{"uuid": uuid du document upload}	Permet d'upload un document sur le service
POST	/archive/{uuid}			Permet d'archiver un document
PUT	/file/{uuid}	{"content": document à upload en base64, "path": chemin du document dans l'arborescence, "metadata": metadata du document		Permet de modifier un document existant
GET	/file/{uuid]		Retourne une entité Media	Permet de récupérer une entité Media
GET	/file/bin/{uuid}		Retourne le document en binaire	Permet de consulter un document
GET	/file/base64/{uuid}		Retourne le document en base64	Permet de consulter un document
GET	/file		Retourne la liste de toutes les entités Media	Permet de récupérer la liste de tous les documents

Group				
Method	**URL**	**Body**	**Response**	**Description**
POST	/dir	{"path": chemin du répertoire à créer}	{"uuid": uuid du répertoire créé}	Permet de créer un répertoire
PUT	/dir/{uuid}	{"path": nouveau chemin du répertoire}		Permet de modifier un répertoire
GET	/dir/{uuid}		Retourne l'entité Group	Permet de récupérer une entité Group
GET	/dir		Retourne la liste de toutes les entités Group	Permet de récupérer la liste de tous les répertoires
DELETE	/dir/{uuid}			Supprime un répertoire, un répertoire peut être supprimé uniquement s'il est vide

ARCHIVE				
Method	**URL**	**Body**	**Response**	**Description**
GET	/archive/{uuid}		Retourne l'entité Archive correspondant à l'uuid	Permet de récupérer une entité Archive
GET	/archive/bin/{uuid}		Retourne le fichier de l'archive correspondant à l'uuid en binaire	Permet de récupérer une archive
GET	/archive/base64/{uuid}		Retourne le fichier de l'archive correspondant à l'uuid en base64	Permet de récupérer une archive
GET	/archive		Retourne la liste de toutes les entités Archive	Permet de récupérer la liste de toutes les archives
DELETE	/archive/{uuid}			Supprime définitivement une archive

Les requêtes PUT pourront modifier partiellement une entité. Il ne sera pas nécessaire de préciser tous les paramètres s'ils ne doivent pas tous être modifiés.

Le nom et l'emplacement des entités Group et Media pourront être modifiés via les requêtes PUT, en modifiant le nom et le chemin dans le paramètre "path". Le type d'un Media pourra être modifié par la même occasion en modifiant l'extension dans le "path".

Service :

Ce projet sera destiné à être consommé par un projet Symfony et devra donc y être intégré. Ce projet devra donc pouvoir être consommé par l'application Symfony en interne sans utiliser les endpoints et proposera donc un service ainsi qu'une interface permettant l'injection de dépendance.

Ce service se nommera GEDService, et proposera une méthode par endpoint, excepté pour les endpoints bin retournant les fichiers en binaires.

Service Encryptor :

Tous les document, archivés ou non, ainsi que les champs name, type et metadata de Media, path et metadata d'Archive et name de Group, devront être chiffrés.

L'algorithme de chiffrement utilisé sera AES-256-CTR, avec une passphrase de 92 caractères et un vecteur d'initialisation de 16 bytes.
Chaque utilisateur aura une passphrase et un vecteur d'initialisation différent. Ceux-ci seront générés et enregistrés dans le fichier texte portant le nom de l'uuid de l'utilisateur dans le répertoire "key_rep". Le format de ces fichiers sera :

«
---Begin Key---
{passphrase}
---End Key---
---Begin Vector---
{vecteur en base64}
---End Vector---
»

Le service Encryptor sera composé de la classe "Encryptor" et de l'interface "EncryptorInterface" :

- **generateKeyPair(string $uuid) : void**
  Génère la passphrase et le vecteur de l'utilisateur correspondant à l'uuid passé en paramètre.
- **loadKeyPair(string $uuid) : void**
  Charge la passphrase et le vecteur de l'utilisateur correspondant à l'uuid passé en paramètre.
- **encryptData(string $data) : $string**
  Chiffre la chaîne de caractère passé en paramètre avec la passphrase et le vecteur précédemment chargés.
- **decryptData(string $data) : string**
  Déchiffre la chaîne de caractère passé en paramètre avec la passphrase et le vecteur précédemment chargés.

Validators :

Il sera nécessaire de veiller à ce que plusieurs contraintes soient respectées.

Une première contrainte, "FileName", s'appliquera à Media et Group. Cette contrainte devra pouvoir être vérifié par l'objet Symfony Validator, et vérifiera qu'il n'existe pas déjà un Media ou un Group ayant le même chemin.

Une autre contrainte sera de vérifier, dans les contrôleurs, qu'un Group n'ait pas d'enfant avant que celui-ci ne soit supprimé, sinon une erreur 422 sera renvoyée.

DTO :

Dans chaque endpoint, le “path“ est visible dans son intégralité, ce n’est pas directement une référence au Group parent. Pour permettre d’envoyer cette information et pour formater les autres informations en ayant besoin, chaque entité aura un DTO contenant les informations tel qu’elles sont visibles par l’utilisateurs.

Pour permettre aux entités d’être transformées depuis le DTO puis chiffrées tout en permettant aux Asserts d’être effectués sur les données non chiffrées, chaque entité se découpera en 3 classes :

- Une entité DTO
- Une entité Decrypt
- Une entité finale telle qu’enregistrée en base de données

Le service “FileSystem“ permettra de passer d’une entité DTO à une entité Decrypt, puis d’une entité Decrypt à une entité finale. Ce service permettra également d’effectuer les validations sur les entités Decrypt, qui, ayant les données en claires, peuvent passer les validations.