



INSTITUT SUPÉRIEUR D'INFORMATIQUE, DE
MODÉLISATION ET DE LEURS APPLICATIONS

PROJET ZZ2 F2
RAPPORT

OCULA

Élèves :

Mouad ISMAILI M'HAMDI
Brahim ID BENOUAKRIM
Achraf EL ALLALI

Enseignant :
Loïc YON

Table des matières

1 Introduction	4
1.1 Contexte et motivation	4
1.2 Problématique	4
1.3 Objectifs du projet	4
1.4 Pérимètre et hypothèses	4
1.5 Organisation du rapport	4
2 Organisation et gestion du projet	4
2.1 Répartition des responsabilités et approche collaborative	4
2.2 Méthodologie Agile itérative	4
2.3 Conception UX : maquettes et validation fonctionnelle	4
2.4 Approche DevOps et stratégie Docker-first	4
2.5 Planification et diagramme de Gantt	4
2.6 Processus de validation et revue de code	5
2.7 Difficultés rencontrées et gestion des risques	5
3 État de l'art et fondements théoriques	5
3.1 Analyse de contenu vidéo : enjeux et défis	5
3.1.1 Données non structurées et difficulté d'indexation	5
3.1.2 Contraintes de latence, coût et confidentialité	5
3.2 Reconnaissance automatique de la parole (ASR)	5
3.2.1 Principes généraux de la transcription audio–texte	5
3.2.2 Choix de modèle (Whisper)	6
3.3 Représentations vectorielles et recherche sémantique	6
3.3.1 Définition des embeddings	6
3.3.2 Indexation et recherche vectorielle	7
3.4 Modèles de langage (LLM)	7
3.4.1 Capacités et limites des LLM	7
3.4.2 Choix de modèle (gpt-oss-120b)	8
3.5 Retrieval-Augmented Generation (RAG)	8
3.5.1 Principe général	8
3.5.2 Forces et limites du RAG	9
4 Spécifications et conception de l'application OCULA	9
4.1 Présentation générale de la solution	9
4.2 Cas d'utilisation et fonctionnalités	10
4.2.1 Upload et gestion des vidéos	10
4.2.2 Transcription et génération de titre et de résumé	10
4.2.3 Chatbot de question–réponse sur la vidéo	11
4.3 Exigences fonctionnelles	12
4.4 Exigences non fonctionnelles	12
4.4.1 Performance et latence	12
4.4.2 Scalabilité	12
4.4.3 Sécurité et confidentialité	12
4.4.4 Robustesse et tolérance aux pannes	12
4.5 Architecture globale	12

4.5.1	Vue d'ensemble (Frontend, Backend, AI Service)	12
4.5.2	Choix technologiques et justification	13
4.6	Conception	14
4.6.1	Modèle conceptuel et logique	14
4.6.2	Modélisation des transcriptions, chunks et embeddings	16
4.6.3	Persistance : base relationnelle, stockage objet et base vectorielle	18
5	Implémentation	20
5.1	Frontend : application Angular	20
5.1.1	Principales pages et composants	20
5.1.2	Gestion des appels API et états (auth, vidéos, chat)	20
5.1.3	Contraintes UX et accessibilité	20
5.2	Backend : API Spring Boot	20
5.2.1	Architecture (controllers, services, repositories)	20
5.2.2	Gestion des utilisateurs et des vidéos	20
5.2.3	Endpoints REST : conception et validation	20
5.2.4	Sécurité : authentification et autorisation	20
5.3	Service IA : LangChain + Flask	20
5.3.1	Extraction audio et transcription (Whisper)	20
5.3.2	Chunking : stratégies et paramètres	20
5.3.3	Génération d'embeddings (MiniLM)	21
5.3.4	RAG : retrieval et construction du prompt	21
5.3.5	Appel au LLM externe et post-traitements	21
5.4	Stockage et (infrastructure)	21
5.4.1	Stockage vidéo et fichiers (Azure Blob)	21
5.4.2	Base de données : choix et schéma	21
5.4.3	Dockerisation	21
6	Évaluation et résultats	21
6.1	Protocole de test	21
6.1.1	Jeu de vidéos utilisé	21
6.1.2	Scénarios de test (upload, transcription, Q&A)	21
6.1.3	Critères : qualité, latence, stabilité	21
6.2	Analyse qualitative	21
6.2.1	Pertinence des résumés et titres	22
6.2.2	Qualité des réponses du chatbot	22
6.2.3	Comportement sur contenu bruité / long / multilingue	22
6.3	Analyse quantitative	22
6.3.1	Temps de traitement par étape (pipeline)	22
6.3.2	Temps de réponse du chatbot	22
6.3.3	Impact du chunking et du top-k retrieval	22
6.4	Discussion	22
6.4.1	Limites observées	22
6.4.2	Risques et biais potentiels	22
6.4.3	Comparaison : RAG vs LLM seul	22

7 Sécurité, conformité et considérations éthiques	22
7.1 Confidentialité des données et stockage	23
7.2 Gestion des accès et des permissions	23
7.3 Propriété intellectuelle et droits sur les contenus vidéo	23
7.4 Limites et usages responsables	23
8 Conclusion et perspectives	23
8.1 Bilan du projet	23
8.2 Améliorations techniques possibles	23
8.2.1 Optimisation de la recherche semantique	23
8.2.2 Streaming transcription	23
8.2.3 Amélioration de la segmentation (chunking adaptatif)	23
8.2.4 Analyse de video frame par frame	23
8.2.5 Infrastructure	23
8.3 Ouverture produit : industrialisation	24

1 Introduction

1.1 Contexte et motivation

À compléter.

1.2 Problématique

À compléter.

1.3 Objectifs du projet

À compléter.

1.4 Périmètre et hypothèses

À compléter.

1.5 Organisation du rapport

À compléter.

2 Organisation et gestion du projet

À compléter.

2.1 Répartition des responsabilités et approche collaborative

À compléter.

2.2 Méthodologie Agile itérative

À compléter.

2.3 Conception UX : maquettes et validation fonctionnelle

À compléter.

2.4 Approche DevOps et stratégie Docker-first

À compléter.

2.5 Planification et diagramme de Gantt

À compléter.

2.6 Processus de validation et revue de code

À compléter.

2.7 Difficultés rencontrées et gestion des risques

À compléter.

3 État de l'art et fondements théoriques

3.1 Analyse de contenu vidéo : enjeux et défis

À compléter.

3.1.1 Données non structurées et difficulté d'indexation

À compléter.

3.1.2 Contraintes de latence, coût et confidentialité

À compléter.

3.2 Reconnaissance automatique de la parole (ASR)

ASR (automatic speech recognition) est une technologie qui convertit le langage parlé en texte écrit. Elle traite les signaux audio, identifie les modèles de parole et les transcrit en texte avec une grande précision.

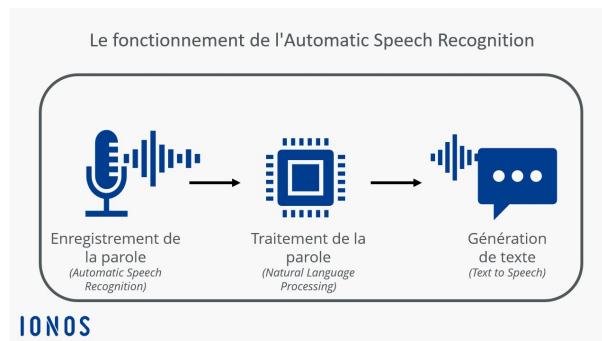


FIGURE 1 – Reconnaissance automatique de la parole

3.2.1 Principes généraux de la transcription audio–texte

Principalement on distingue entre deux approches pour la reconnaissance automatique de la parole :

- **Approche hybride classique** : est une méthode qui combine un modèle acoustique, un modèle de lexique et un modèle linguistique pour transcrire la parole en texte, en s'appuyant sur des données audio alignées afin d'associer précisément les segments aux phonèmes et aux mots.

- **Deep Learning** : utilise des réseaux de neurones pour convertir directement la parole en texte, sans séparer explicitement les modèles acoustique, lexique et linguistique. Elle apprend automatiquement les relations entre les sons et les mots à partir de grandes quantités de données audio.

3.2.2 Choix de modèle (Whisper)

Whisper est un système de reconnaissance automatique de la parole (ASR) basé sur l'intelligence artificielle, entraîné sur environ 680 000 heures de données audio multilingues provenant du web. Grâce à cette très grande diversité de données, il est particulièrement robuste face aux accents, au bruit de fond et au vocabulaire technique. Whisper est capable non seulement de transcrire la parole dans plusieurs langues, mais aussi de traduire directement ces langues vers l'anglais, ce qui en fait un outil polyvalent pour de nombreuses applications de traitement vocal et de recherche en intelligence artificielle.

TABLE 1 – Comparaison des Performance des Models ASR

Criteria	Whisper	AssemblyAI U3	Amazon	Microsoft
Accuracy (English)	92.4%	94.1%	92.4%	92.5%
Accuracy (Multilingual)	92.6%	91.3%	89.9%	88.9%
WER (English)	6.5%	5.9%	7.6%	7.5%
WER (Multilingual)	7.4%	8.7%	10.1%	11.1%

Le choix de Whisper a été évident , du à ça capacité impressionnante dans la transcription multilingue , et sont WER minimal par rapport aux autre modèle. De plus nous favorisons les modèles open-source, et Whisper étant le meilleur dans cette catégorie.

3.3 Représentations vectorielles et recherche sémantique

3.3.1 Définition des embeddings

Un embedding est une représentation vectorielle d'éléments comme les textes, les images, les vidéos ou les signaux audio. Cette transformation en valeurs numériques permet aux modèles de machine learning de comprendre les relations sémantiques et d'effectuer des tâches telles que la recherche, la similarité ou la classification.

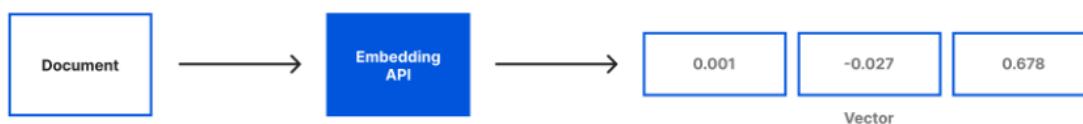


FIGURE 2 – Fonctionnement de embedding

3.3.2 Indexation et recherche vectorielle

L'indexation vectorielle est une technique qui consiste à organiser des données sous forme de vecteurs numériques afin de permettre une recherche rapide et efficace dans des espaces de grande dimension.

La recherche vectorielle est le processus qui consiste à comparer le vecteur d'une requête avec ceux stockés dans cet index afin d'identifier les éléments les plus proches sémantiquement.

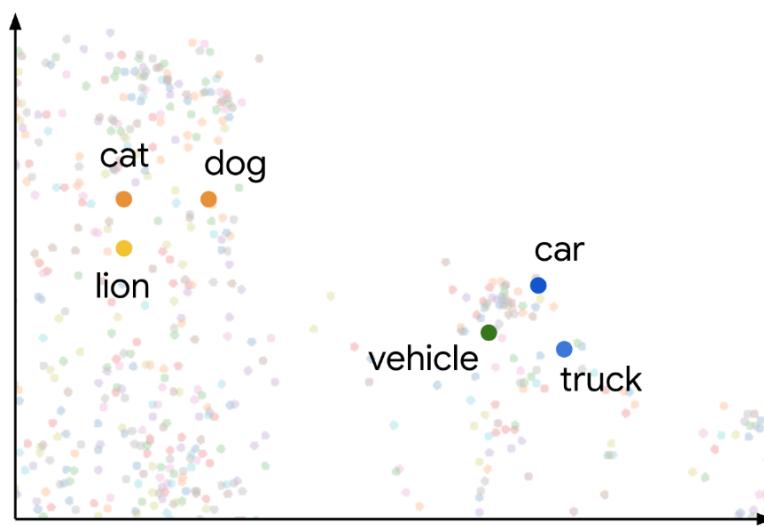


FIGURE 3 – Illustration de l'indexation vectorielle

La figure illustre le principe de l'indexation et de la recherche vectorielle. Chaque point coloré représente un objet (mot, image ou donnée) transformé en vecteur dans un espace numérique. Les éléments ayant un sens similaire se retrouvent proches les uns des autres, formant des groupes sémantiques. Par exemple, les mots “cat”, “dog” et “lion” sont regroupés car ils appartiennent à la même catégorie d’animaux, tandis que “car”, “truck” et “vehicle” forment un autre groupe lié aux moyens de transport. Lors d’une recherche vectorielle, le système consiste simplement à retrouver les points les plus proches du vecteur de la requête afin d’identifier les résultats les plus pertinents.

3.4 Modèles de langage (LLM)

3.4.1 Capacités et limites des LLM

Un LLM (Large Language Model ou Grand Modèle de Langage) est un système d’intelligence artificielle entraîné sur d’immenses quantités de données textuelles pour comprendre, générer et manipuler le langage humain de manière fluide et cohérente.

- **Capacités :**

- **Génération de contenu :** ils peuvent rédiger instantanément des textes.
- **Synthèse d’informations :** ils peuvent résumer des documents très long en extrayant les points clés.
- **Traduction et adaptation de style :** Ils peuvent traduire des langues et modifier le style d’un texte.

- **Limites**

- **Hallucinations** : ils peuvent confirmer des informations fausses.
- **Coût de calcul élevé** : ils sont très chers à entraîner et à faire fonctionner.

3.4.2 Choix de modèle (gpt-oss-120b)

Lors de notre recherche des modèles de LLM à utiliser, nous avons identifié deux candidats principaux : gpt-oss-120b et llama3.1-8b, tous deux accessibles via l'API fournie par Cerebras.

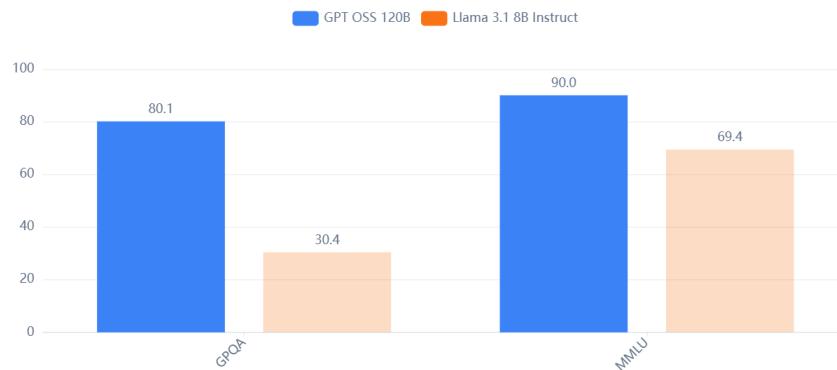


FIGURE 4 – Camparaison entre gpt-oss-120b et llama3.1-8b

En analysant les deux indices de performance, GPQA (Graduate-Level Google-Proof Q&A) et MMLU (Massive Multitask Language Understanding), nous pouvons clairement constater que le modèle gpt-oss-120b surpasse llama3.1-8b. Cette supériorité s'explique notamment par son nombre beaucoup plus élevé de paramètres (120 milliards) ainsi que par son architecture plus avancée, lui permettant d'obtenir de meilleurs résultats sur des tâches complexes de raisonnement et de compréhension.

3.5 Retrieval-Augmented Generation (RAG)

3.5.1 Principe général

Le RAG (Retrieval-Augmented Generation) est une technique qui permet d'optimiser les réponses d'un modèle d'IA en lui donnant accès à des données externes fiables, au-delà de ses connaissances d'entraînement initiales.

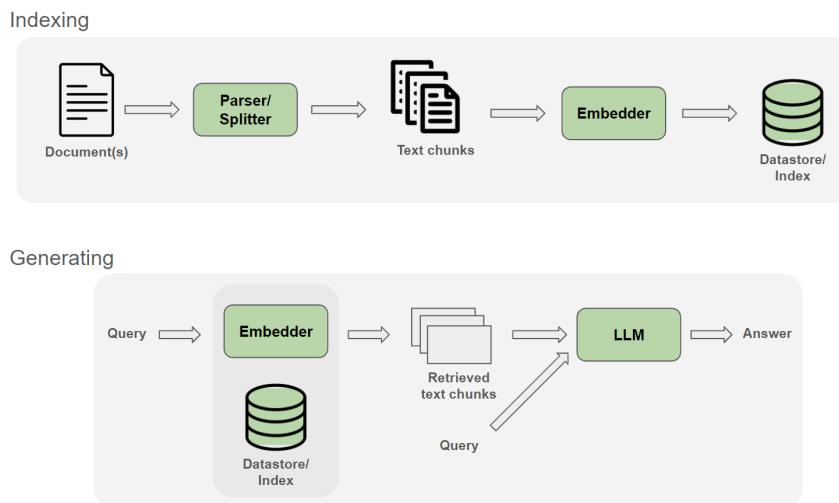


FIGURE 5 – Pipeline de RAG

Un pipeline de RAG fonctionne en trois étapes simples : d'abord, des documents sont collectés, découpés en petits morceaux puis transformés en vecteurs pour être stockés dans une base vectorielle. Ensuite, quand un utilisateur pose une question, elle est aussi convertie en vecteur afin de retrouver les passages les plus pertinents dans cette base. Enfin, ces informations sont envoyées avec la question à un LLM, qui s'en sert comme contexte pour produire une réponse plus précise et fiable.

3.5.2 Forces et limites du RAG

Le RAG est une approche qui combine la recherche d'informations externes avec la génération de texte par un LLM. Cette architecture permet d'améliorer la qualité des réponses, mais elle présente également certaines contraintes techniques.

- **Amélioration de la précision :** Le RAG permet de générer des réponses plus fiables en s'appuyant sur des documents externes, ce qui réduit fortement les hallucinations des modèles de langage.
- **Dépendance à la qualité de la recherche :** La performance du système dépend fortement de la pertinence des documents récupérés.
- **Latence et complexité :** L'étape de récupération des données ajoute du temps de réponse et augmente les coûts techniques et computationnels.

4 Spécifications et conception de l'application OCULA

4.1 Présentation générale de la solution

Notre application **Ocula** est une solution dédiée au traitement et à l'analyse de vidéos. Elle permet d'extraire automatiquement les transcriptions, puis de les exploiter pour générer des titres et des résumés pertinents. De plus, elle intègre une approche RAG (Retrieval-Augmented Generation) afin de répondre de manière précise et contextuelle aux questions des utilisateurs concernant le contenu des vidéos.

4.2 Cas d'utilisation et fonctionnalités

4.2.1 Upload et gestion des vidéos

Notre solution **Ocula** gère les vidéos téléversées en s'appuyant sur le service de **Azure Blob Storage**. L'utilisateur peut importer une vidéo depuis la partie client, puis une requête est envoyée au backend afin de créer automatiquement une entrée correspondante dans notre base de données. La vidéo est ensuite stockée dans notre object storage, où elle est associée à une clé unique. Cette clé permet de générer une URL sécurisée donnant accès à la vidéo lorsque cela est nécessaire.

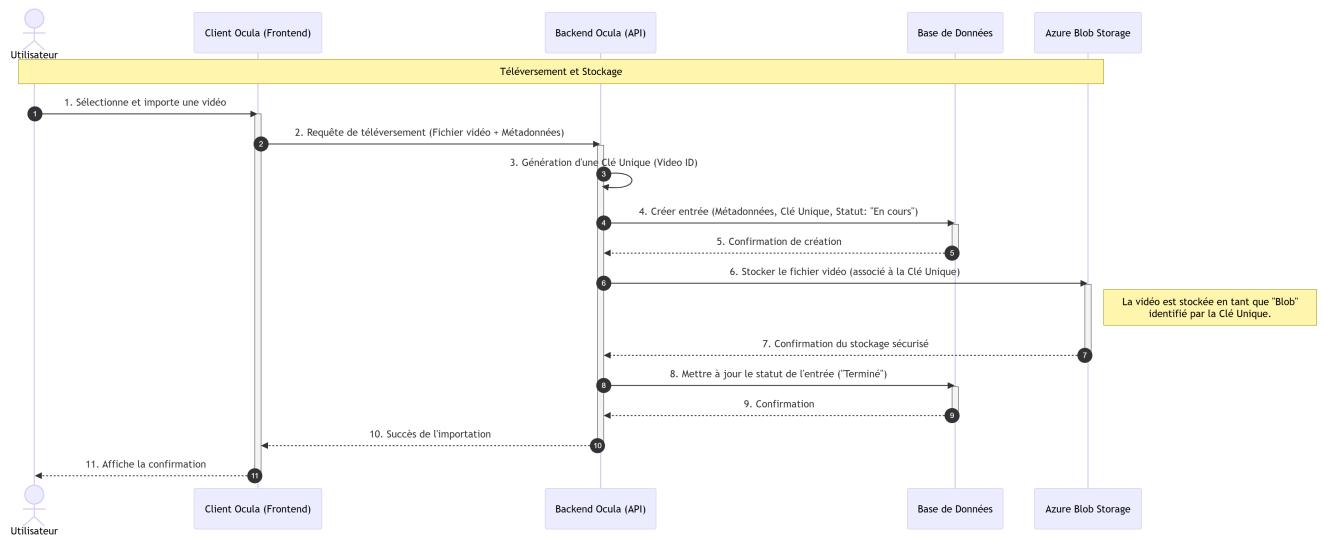


FIGURE 6 – Enter Caption

4.2.2 Transcription et génération de titre et de résumé

Notre solution **Ocula** permet d'extraire automatiquement la transcription des vidéos en exploitant les capacités de **Whisper**. Après le téléchargement d'une vidéo, une tâche de traitement est déclenchée afin de l'analyser. Une fois la transcription extraite, nous enregistrons ses différents segments, puis nous divisons le script en plusieurs chunks.

Ces chunks sont ensuite transformés en embeddings et stockés dans une base de données vectorielle, afin de pouvoir être exploités ultérieurement dans un système **RAG** (Retrieval-Augmented Generation). Enfin, grâce à une pipeline composée d'un prompt et d'un LLM (**gpt-oss-120b**), la solution génère automatiquement un titre pertinent ainsi qu'un résumé du contenu de la vidéo.

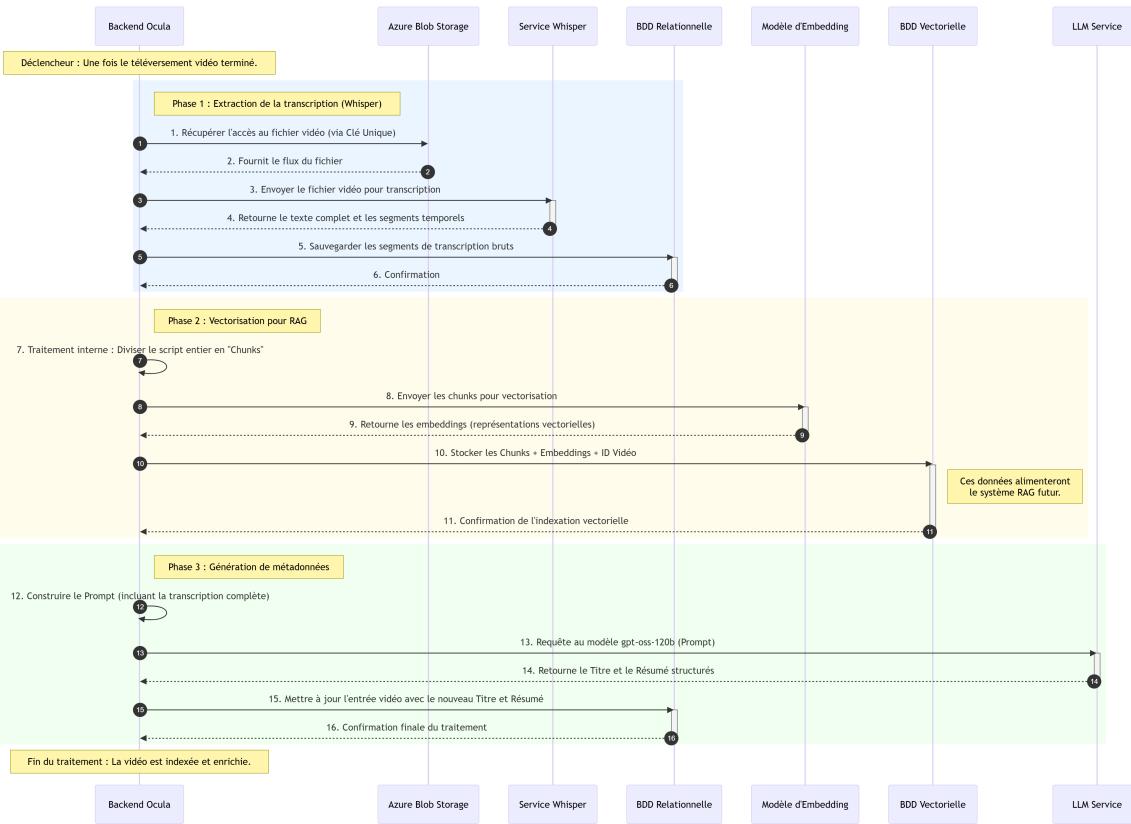


FIGURE 7 – Enter Caption

4.2.3 Chatbot de question-réponse sur la vidéo

Notre solution **Ocula** intègre un assistant **chatbot** basé sur l'intelligence artificielle. Cet assistant utilise un **LLM** afin de répondre aux questions des utilisateurs tout en s'appuyant sur une approche **RAG** (Retrieval-Augmented Generation) et sur les différents chunks générés lors du traitement des vidéos.

Lorsqu'un utilisateur envoie un message, le chatbot déclenche une pipeline RAG qui recherche les informations pertinentes dans les chunks de la transcription associés à la vidéo concernée. Le contexte ainsi récupéré est ensuite transmis au LLM (**gpt-oss-120b**), ce qui lui permet de fournir des réponses précises, pertinentes et adaptées au contenu réel de la vidéo.

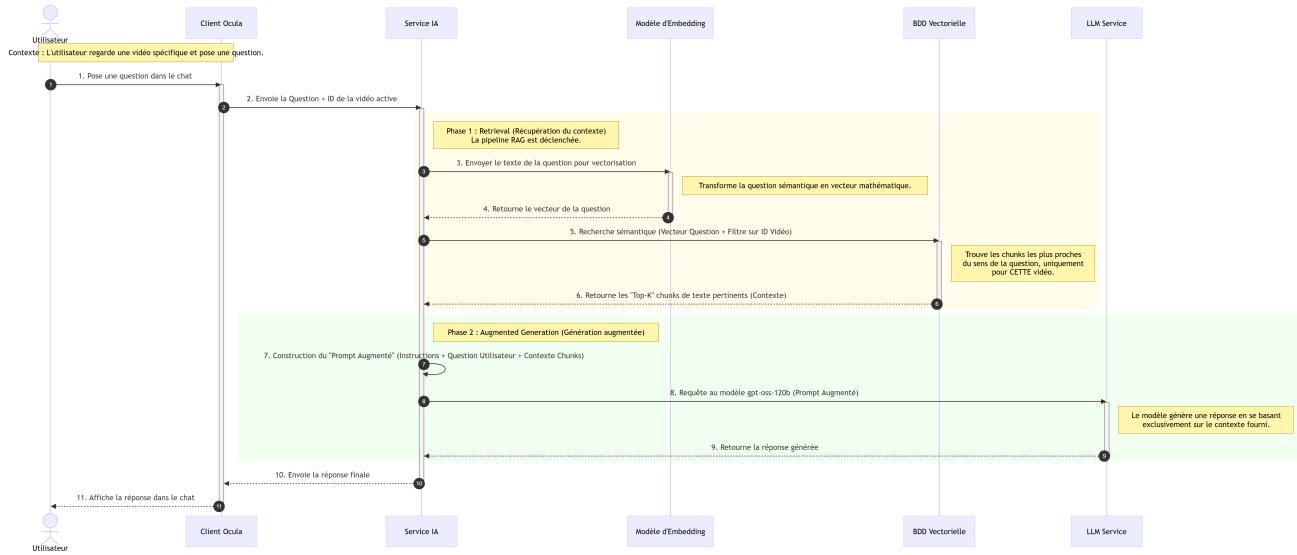


FIGURE 8 – Enter Caption

4.3 Exigences fonctionnelles

À compléter.

4.4 Exigences non fonctionnelles

À compléter.

4.4.1 Performance et latence

À compléter.

4.4.2 Scalabilité

À compléter.

4.4.3 Sécurité et confidentialité

À compléter.

4.4.4 Robustesse et tolérance aux pannes

À compléter.

4.5 Architecture globale

4.5.1 Vue d'ensemble (Frontend, Backend, AI Service)

Notre application **Ocula** suit une architecture structurée en trois principales couches. D'abord, le frontend permet à l'utilisateur de téléverser une vidéo et de demander son analyse. Ensuite, le backend gère les requêtes, enregistre les métadonnées dans la base de données et stocke les fichiers dans un système de stockage objet. Enfin, la partie IA

prend en charge le traitement de la vidéo, elle génère la transcription, crée des embeddings stockés dans une base vectorielle, puis exploite ces données pour alimenter le LLM, qui produit des résumés, des titres et des réponses pertinentes aux questions des utilisateurs.

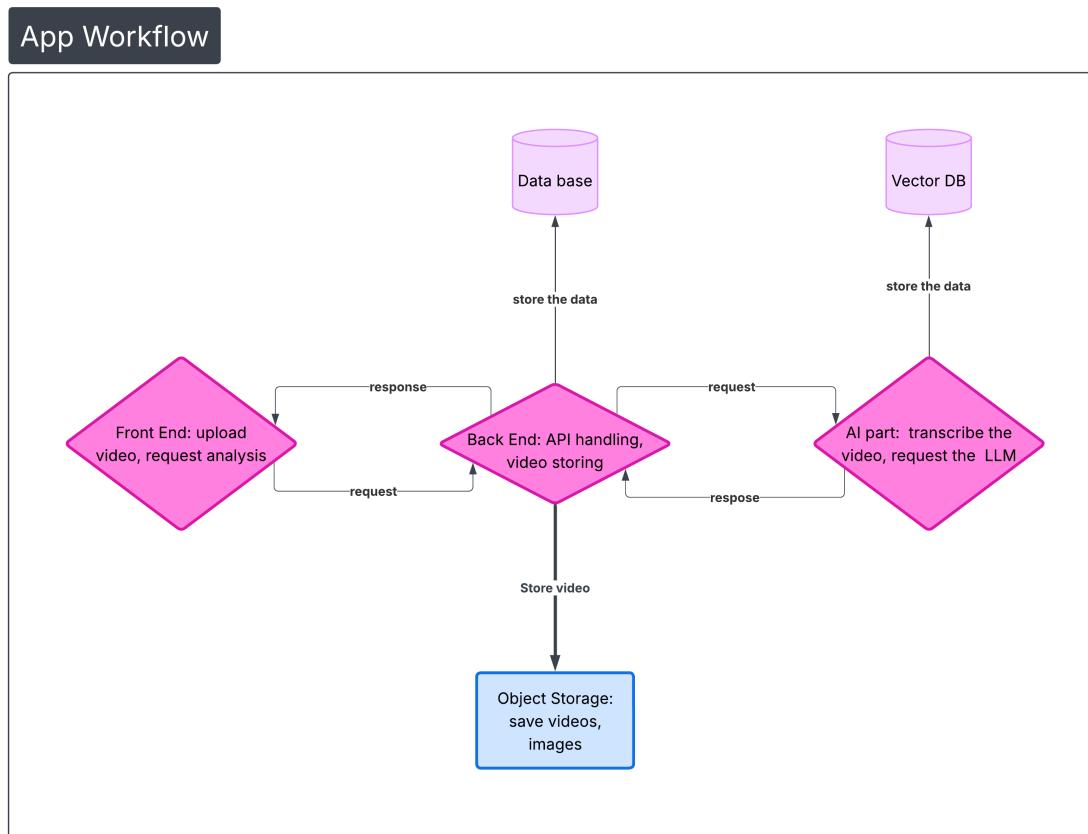


FIGURE 9 – Vue d'ensemble

4.5.2 Choix technologiques et justification

Pour le développement de l'application **Ocula**, nous avons choisi des technologies adaptées à une architecture moderne et scalable.

Le **backend**, nous avons choisi **Spring-Boot**, un framework Java largement adopté pour la conception d'API robustes et sécurisées. Il offre une intégration avec les bases de données relationnelles, ainsi qu'un écosystème facilitant le développement.

Le **frontend** a été développé avec **Angular**, un framework structuré et performant, particulièrement adapté aux applications web complexes. **Angular** permet une bonne organisation du code, une communication fluide avec les API REST, ainsi qu'une expérience utilisateur dynamique et réactive.

La **partie intelligence artificielle**, nous avons utilisé **Python** avec **Flask** pour exposer les services IA sous forme d'API légères et rapides. L'intégration de **LangChain** a permis de faciliter la mise en place des pipelines RAG.

Pour la **persistante des données**, nous avons choisi **PostgreSQL** comme base de données relationnelle, en raison de sa capacité à gérer des structures de données complexes. En complément, nous avons utilisé **ChromaDB** comme base de données vectorielle, spécialement conçue pour le stockage et la recherche d'embeddings.

4.6 Conception

4.6.1 Modèle conceptuel et logique

Le modèle conceptuel d'OCULA est structuré autour de l'entité *Video*, qui constitue le cœur fonctionnel du système. Bien que l'application soit multi-utilisateur, l'ensemble du cycle de traitement IA est vidéo-centré : les opérations d'analyse, de transcription, de génération d'embeddings et de question-réponse s'appuient systématiquement sur l'identifiant `video_id`. Ainsi, l'entité *User* possède les vidéos, mais c'est la *Video* qui orchestre le pipeline métier.

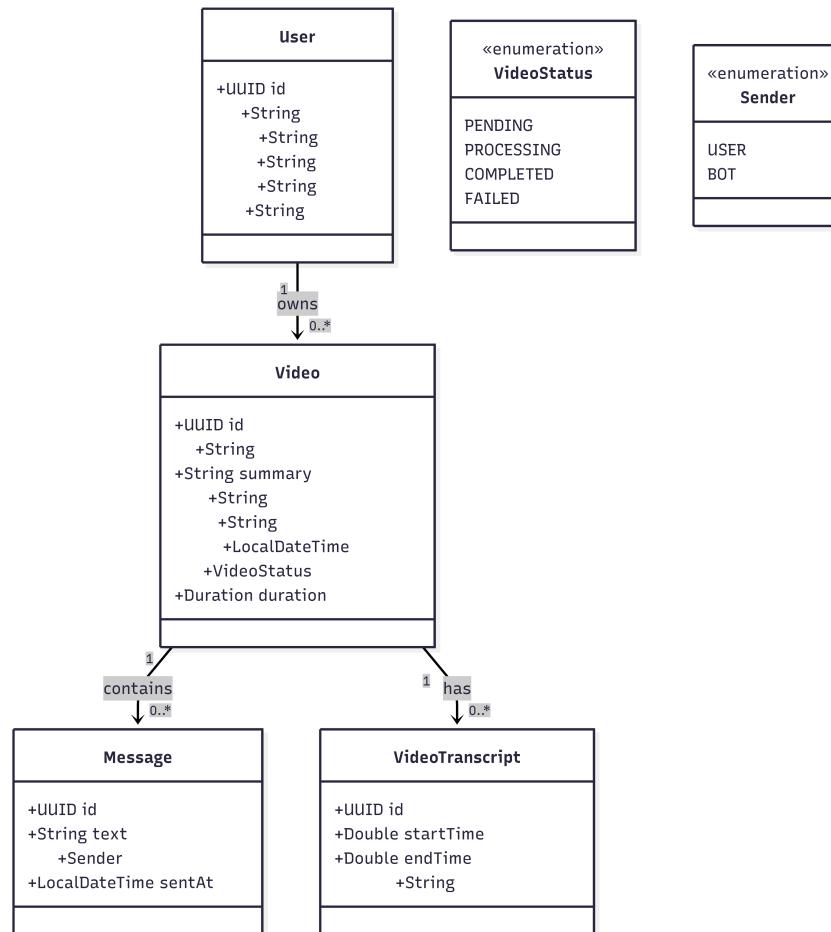


FIGURE 10 – Diagramme de classes du domaine métier d'OCULA

Relation User – Video Un utilisateur peut posséder plusieurs vidéos (1–N). Cette relation n'est pas uniquement logique, elle répond à plusieurs exigences architecturales :

- **Sécurité** : le contrôle d'accès repose sur le propriétaire de la vidéo ; un utilisateur ne peut accéder qu'à ses propres ressources.
- **Isolation des données** : chaque compte constitue un espace logique distinct.
- **Scalabilité** : les requêtes peuvent être filtrées et paginées par `userId`, améliorant les performances.
- **Architecture multi-tenant** : la base est partagée, mais le cloisonnement est assuré au niveau applicatif.

Relation Video – VideoTranscript Contrairement à une approche monolithique consistant à stocker une transcription complète sous forme d'un simple champ texte, OCULA adopte une modélisation segmentée. Une vidéo possède donc **0..*** entités *VideoTranscript*, chacune caractérisée par un **startTime**, un **endTime** et un **transcriptText**.

Ce choix répond à plusieurs objectifs :

- **Traçabilité temporelle** : chaque portion de texte peut être reliée à un intervalle précis de la vidéo.
- **Robustesse du pipeline** : en cas d'échec partiel du traitement, les segments déjà générés restent persistés.
- **Réindexation ciblée** : il est possible de retraiter ou ré-encoder certains segments sans recalcul global.
- **Granularité adaptée au RAG** : les segments constituent la base naturelle du découpage en chunks.

Relation Video – Message Chaque vidéo peut contenir **0..*** messages correspondant aux échanges entre l'utilisateur et le chatbot. La persistance de ces messages dépasse le simple besoin d'affichage :

- **Historique multi-session** : la conversation est conservée même après recharge-ment ou changement d'appareil.
- **Audit et explicabilité** : il est possible d'analyser les requêtes et réponses pro-duites.
- **Évolutivité** : ces données peuvent servir à des analyses ultérieures ou à l'amélio-ration du système.

Gestion d'état du traitement L'attribut **status** de l'entité *Video* (PENDING, PROCESSING, COMPLETED, FAILED) modélise explicitement le cycle de vie du traitement asynchrone.

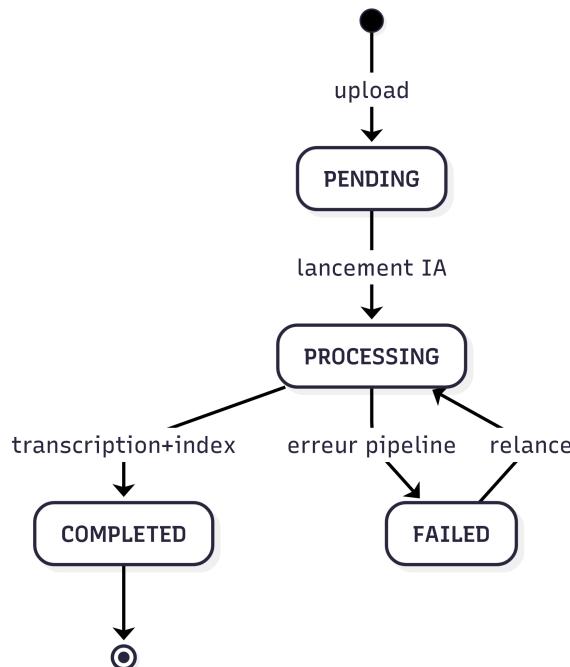


FIGURE 11 – Gestion d'état du traitement (VideoStatus)

Dans un système distribué où la transcription et l'indexation peuvent prendre plusieurs minutes, ce champ est essentiel pour :

- informer le frontend de l'avancement,
- permettre une mise à jour en temps réel via WebSocket,
- gérer proprement les erreurs et les tentatives de relance,
- éviter les états incohérents ou bloqués.

Ainsi, le modèle conceptuel ne se limite pas à une simple représentation des données : il structure le comportement global du système et soutient la robustesse de l'architecture.

4.6.2 Modélisation des transcriptions, chunks et embeddings

Si le modèle précédent décrit la structure relationnelle des données métier, le mécanisme de recherche sémantique repose sur une modélisation complémentaire adaptée au RAG.

1. De la transcription segmentée aux chunks

La transcription est initialement stockée sous forme de segments temporels (*Video-Transcript*). Toutefois, une recherche sémantique efficace ne peut s'appuyer ni sur un texte monolithique complet, ni sur des segments trop courts et isolés.

Le système applique donc une étape de *chunking*, consistant à regrouper et découper les segments afin de produire des blocs textuels cohérents et exploitables pour l'indexation vectorielle.

Cette granularité intermédiaire permet :

- d'améliorer la précision lors de la récupération de contexte,
- de réduire le bruit informationnel,
- d'optimiser l'utilisation de la fenêtre de contexte du LLM,
- de maintenir un équilibre entre cohérence globale et pertinence locale.

2. Génération des embeddings

Chaque chunk est transformé en vecteur numérique à l'aide du modèle `sentence-transformers/all-` chargé via *HuggingFaceEmbeddings*.

Ce modèle encode les textes en vecteurs de dimension fixe permettant une comparaison sémantique par mesure de similarité ou de distance vectorielle. Chaque chunk correspond ainsi à un triplet logique :

- texte source,
- embedding haute dimension,
- métadonnées associées.

Cette représentation vectorielle permet de comparer la question de l'utilisateur avec le contenu vidéo et d'identifier les passages les plus pertinents.

3. Organisation dans la base vectorielle

Les embeddings sont stockés dans ChromaDB selon une organisation par vidéo. Pour chaque vidéo, une collection dédiée (`video_<video_id>`) est utilisée ou créée lors de linitialisation du flux d'indexation.

Chaque document vectoriel contient :

- le texte du chunk,
- lembedding associé,
- des métadonnées : `video_id`, `start_time`, `end_time`.

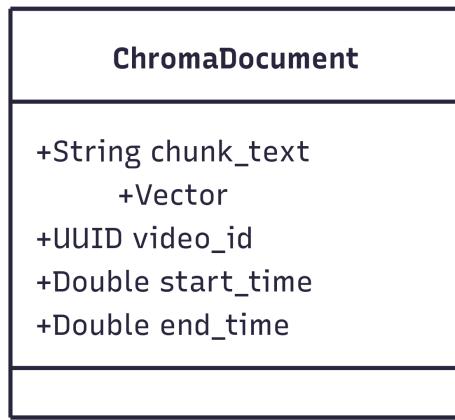


FIGURE 12 – Structure d'un document vectoriel dans ChromaDB

Lors d'une requête utilisateur, le mécanisme RAG :

- effectue une recherche par similarité dans la collection de la vidéo concernée,
- sélectionne les k passages les plus proches,
- applique un filtrage et un éventuel *reranking* afin d'améliorer la pertinence du contexte transmis au LLM.

Cette organisation garantit :

- l'isolation des recherches à une vidéo donnée,
- la traçabilité des sources utilisées dans la réponse,
- une correspondance directe entre passage récupéré et intervalle temporel.

Le modèle relationnel assure la cohérence transactionnelle des données métier, tandis que la couche vectorielle optimise la récupération contextuelle pour le RAG ; les deux couches sont donc complémentaires et synchronisées par `video_id`.

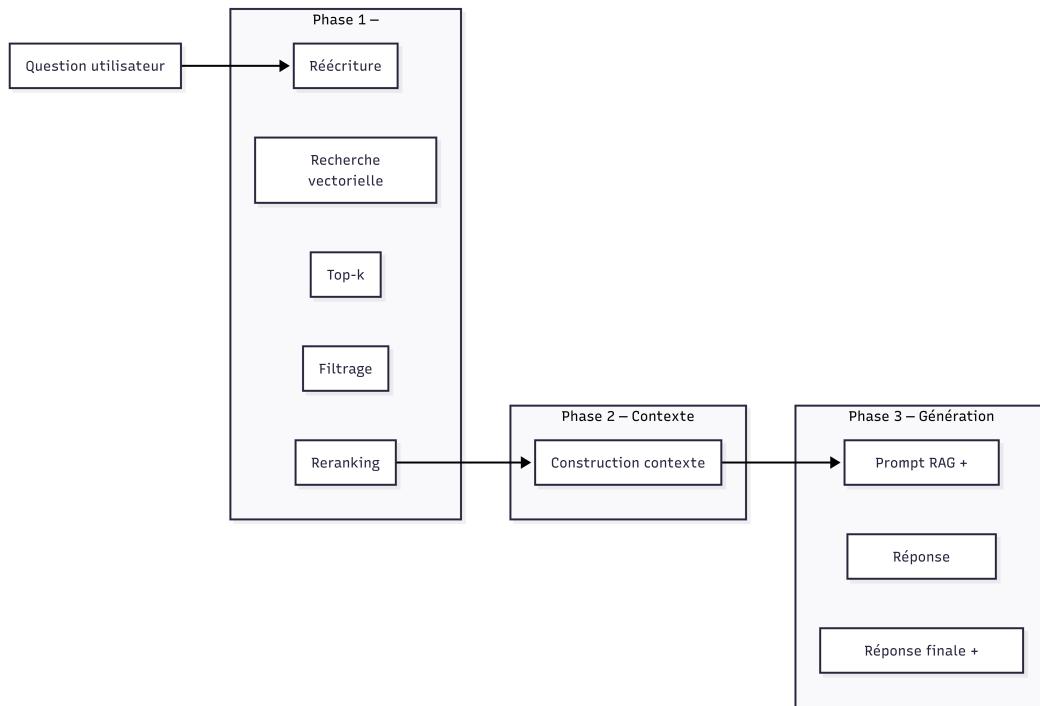


FIGURE 13 – Pipeline RAG : phases de retrieval, construction du contexte et génération

4.6.3 Persistance : base relationnelle, stockage objet et base vectorielle

La nature hétérogène des données manipulées par OCULA impose une stratégie de persistance adaptée à chaque type d'information. Le système distingue ainsi trois couches de stockage complémentaires : relationnelle, objet et vectorielle.

Base relationnelle (PostgreSQL) Les entités métier structurées (*User*, *Video*, *Message*, *VideoTranscript*) sont stockées dans une base relationnelle.

Ce choix permet :

- de garantir l'intégrité référentielle entre les entités,
- d'assurer la cohérence transactionnelle au niveau des données métier,
- de faciliter les requêtes applicatives (filtrage, pagination, jointures),
- de maintenir une structure normalisée et contrôlée.

Il est toutefois important de noter qu'il n'existe pas de transaction globale entre PostgreSQL, le stockage objet et la base vectorielle : l'architecture repose sur une coordination applicative entre services.

Stockage objet (Azure Blob Storage) Les fichiers volumineux tels que les vidéos, les miniatures et les avatars sont stockés dans un service de type objet.

La base relationnelle conserve principalement la *clé d'objet* (object key) associée à chaque ressource (*content*, *thumbnail*, *avatar*). Une URL signée est générée dynamiquement lors de l'accès au fichier.

Ce choix permet :

- d'optimiser le stockage des contenus binaires,
- de réduire la charge sur la base relationnelle,

- d'assurer une meilleure scalabilité des médias,
- de séparer clairement données structurées et fichiers lourds.

Base vectorielle (ChromaDB) Les embeddings générés à partir des chunks sont stockés dans une base vectorielle dédiée (ChromaDB).

Chaque vidéo dispose d'une collection nommée `video_<video_id>`.

Cette organisation assure :

- l'isolement des recherches par vidéo,
- la traçabilité temporelle des passages récupérés,
- une recherche efficace par mesure de similarité ou distance vectorielle.

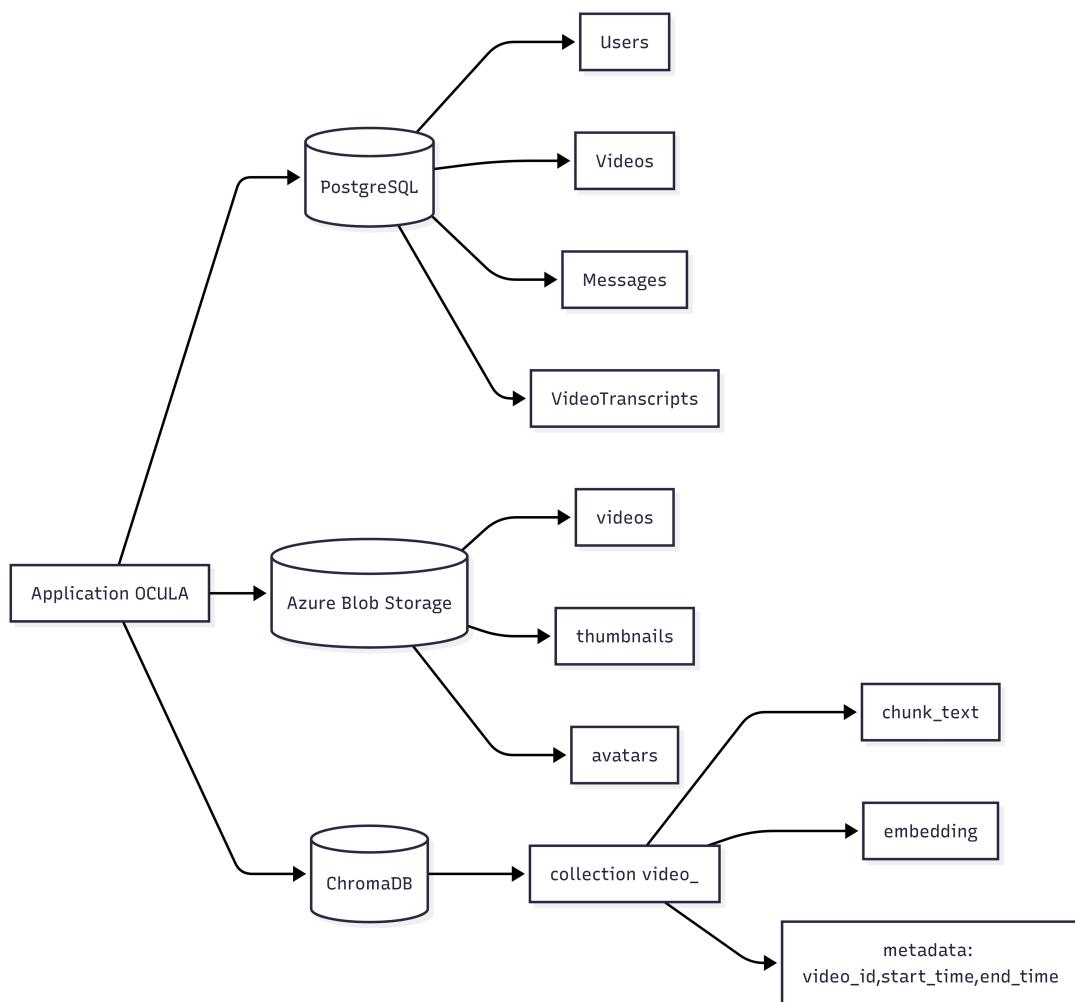


FIGURE 14 – Architecture de persistance hybride d'OCULA : base relationnelle, stockage objet et base vectorielle

Le modèle relationnel assure la cohérence des données métier, le stockage objet prend en charge les contenus binaires via des clés d'objet, et la base vectorielle optimise la récupération sémantique pour le RAG. Ces trois couches sont complémentaires et interconnectées principalement par `video_id` (et `user_id` pour les ressources utilisateur).

5 Implémentation

À compléter.

5.1 Frontend : application Angular

À compléter.

5.1.1 Principales pages et composants

À compléter.

5.1.2 Gestion des appels API et états (auth, vidéos, chat)

À compléter.

5.1.3 Contraintes UX et accessibilité

À compléter.

5.2 Backend : API Spring Boot

À compléter.

5.2.1 Architecture (controllers, services, repositories)

À compléter.

5.2.2 Gestion des utilisateurs et des vidéos

À compléter.

5.2.3 Endpoints REST : conception et validation

À compléter.

5.2.4 Sécurité : authentification et autorisation

À compléter.

5.3 Service IA : LangChain + Flask

À compléter.

5.3.1 Extraction audio et transcription (Whisper)

À compléter.

5.3.2 Chunking : stratégies et paramètres

À compléter.

5.3.3 Génération d'embeddings (MiniLM)

À compléter.

5.3.4 RAG : retrieval et construction du prompt

À compléter.

5.3.5 Appel au LLM externe et post-traitements

À compléter.

5.4 Stockage et (infrastructure)

À compléter.

5.4.1 Stockage vidéo et fichiers (Azure Blob)

À compléter.

5.4.2 Base de données : choix et schéma

À compléter.

5.4.3 Dockerisation

6 Évaluation et résultats

À compléter.

6.1 Protocole de test

À compléter.

6.1.1 Jeu de vidéos utilisé

À compléter.

6.1.2 Scénarios de test (upload, transcription, Q&A)

À compléter.

6.1.3 Critères : qualité, latence, stabilité

À compléter.

6.2 Analyse qualitative

À compléter.

6.2.1 Pertinence des résumés et titres

À compléter.

6.2.2 Qualité des réponses du chatbot

À compléter.

6.2.3 Comportement sur contenu bruité / long / multilingue

À compléter.

6.3 Analyse quantitative

À compléter.

6.3.1 Temps de traitement par étape (pipeline)

À compléter.

6.3.2 Temps de réponse du chatbot

À compléter.

6.3.3 Impact du chunking et du top-k retrieval

À compléter.

6.4 Discussion

À compléter.

6.4.1 Limites observées

À compléter.

6.4.2 Risques et biais potentiels

À compléter.

6.4.3 Comparaison : RAG vs LLM seul

À compléter.

7 Sécurité, conformité et considérations éthiques

À compléter.

7.1 Confidentialité des données et stockage

À compléter.

7.2 Gestion des accès et des permissions

À compléter.

7.3 Propriété intellectuelle et droits sur les contenus vidéo

À compléter.

7.4 Limites et usages responsables

À compléter.

8 Conclusion et perspectives

À compléter.

8.1 Bilan du projet

À compléter.

8.2 Améliorations techniques possibles

À compléter.

8.2.1 Optimisation de la recherche semantique

À compléter.

8.2.2 Streaming transcription

À compléter.

8.2.3 Amélioration de la segmentation (chunking adaptatif)

À compléter.

8.2.4 Analyse de video frame par frame

À compléter.

8.2.5 Infrastructure

À compléter.

8.3 Ouverture produit : industrialisation

À compléter.