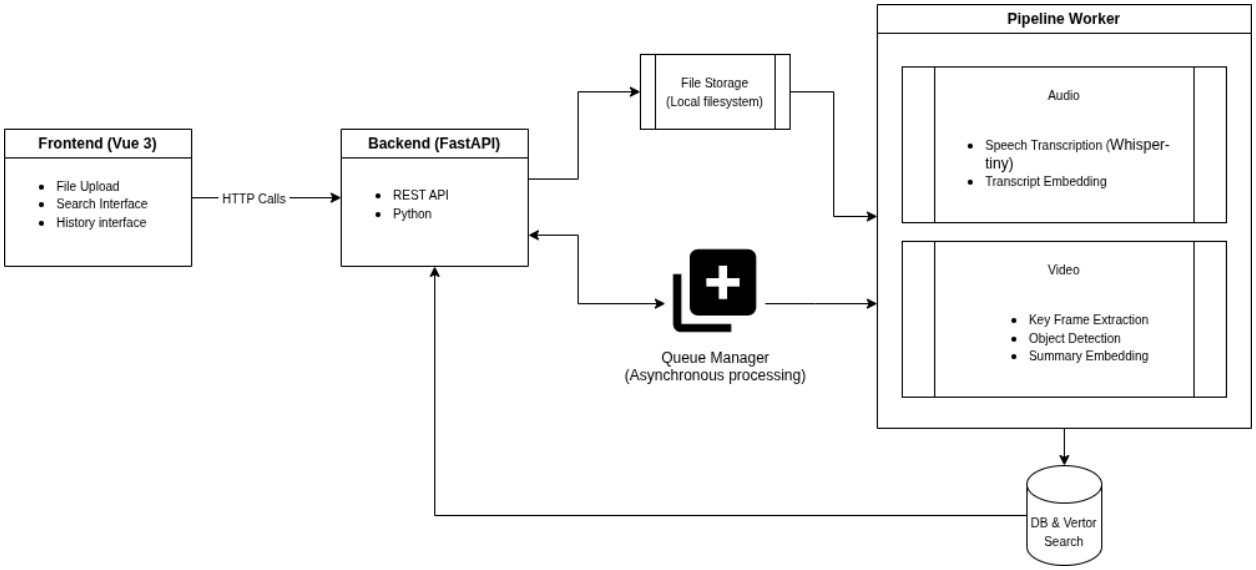

Frontend (Vue 3)

The frontend provides the user interface for uploading audio and video files, performing search queries, and viewing processing history. It communicates with the backend exclusively through HTTP REST calls and does not perform any media processing locally.

Backend (FastAPI)

The backend exposes RESTful APIs for media ingestion, retrieval, and search. It validates incoming requests, stores uploaded files, and coordinates media processing and search operations. The backend also serves as the integration point between the frontend, file storage, processing pipeline, and database.

File Storage

Uploaded audio and video files are stored on the local filesystem for the scope of this assignment. This separates file ingestion from processing and avoids keeping large binary payloads in memory or the database. In a production system, this component could be replaced with object storage such as S3.

Queue Manager

The queue manager represents a decoupling mechanism between API requests and compute-intensive processing. While processing may be performed synchronously for this assignment, this component illustrates how long-running audio and video jobs could be handled asynchronously to improve scalability and reliability.

Pipeline Worker – Audio

The audio pipeline performs speech transcription using the *Whisper-tiny* model, followed by timestamp and confidence extraction. The transcribed text is embedded using the same text embedding model as video (*all-MiniLM-L6-v2*) to enable unified vector search across media types.

Pipeline Worker – Video

The video pipeline extracts key frames from uploaded videos, performs object detection, and generates a textual summary of visual content. This summary is embedded using the same embedding model as audio, ensuring that audio and video embeddings share a common semantic space.

DB & Vector Search

The SQLite database stores metadata, timestamps, and embeddings for both audio and video. Embeddings are stored as serialized binary blobs (*Search Option 3*). During search, embeddings are deserialized into NumPy arrays and cosine similarity is computed directly in Python to rank results across both media types.

Design Considerations

This architecture balances simplicity and extensibility. Lightweight models prioritize processing speed over maximum accuracy, while the separation of storage, processing, and search provides clear extension points for GPU acceleration, larger models, or dedicated vector search infrastructure if more computational resources are available.