**Assignment: AI Meeting Intelligence Platform**

**Objective:** Build a web application that extracts actionable insights from meeting recordings. Users should be able to upload audio/video files and receive structured information about action items, decisions, and participant interactions.

**Core Requirements:**

**Frontend:**

* File upload interface for audio/video
* Dashboard displaying meeting insights
* Action items and tasks view
* Basic analytics visualization

**Backend**:

* Audio/video file processing
* Speech-to-text conversion
* Information extraction using LLMs
* Simple data persistence
* Search functionality

**Technical Stack (Suggested):**

* Frontend: React + Tailwind
* Backend: Python FastAPI
* Database: SQLite
* AI Components:

1. Whisper.cpp (transcription)
2. Ollama (LLM)
3. ChromaDB (vector search)

**Expected Deliverables:**

* A github repository with source code for the application.
* A README with setup instructions and required configurations.
* Brief technical write-up explaining the following:

1. Architecture decisions
2. AI pipeline implementation
3. Challenges and solutions
4. Evaluation Criteria
5. Code quality and organization
6. API design
7. AI/ML pipeline implementation
8. Error handling
9. Documentation

* (Optional) Live url to the deployed application.

**🧠 AI Meeting Intelligence Platform - Solution Design (Updated for New Tech Stack)**

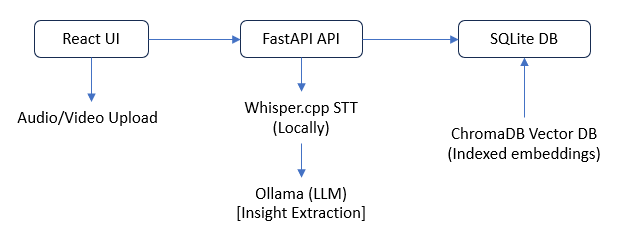
**1. Objective**

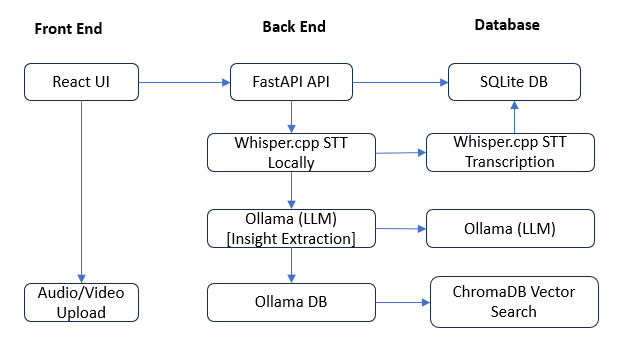
Build a web application that extracts actionable insights from meeting recordings.  
Users upload audio/video files and receive structured information about:

* ✅ Action items
* ✅ Key decisions
* ✅ Participant interactions

**2. Architecture Diagram**

**High-Level Architecture Overview:**





**3. Architecture Decisions**

**Frontend**

* **React + Tailwind CSS**: Clean and responsive UI using component-based structure.
* Features:
  + File upload interface for audio/video.
  + Insight dashboard for decisions, actions, and speaker summaries.
  + Visual analytics (e.g., pie charts for talk time, decision categories).

**Backend**

* **Python FastAPI**:
  + Lightweight, fast, async-capable API layer.
  + REST endpoints for file upload, job status, results, and search.
* Background processing using asyncio, Celery, or FastAPI BackgroundTasks for long-running tasks.

**Storage and Compute**

* **SQLite**: Lightweight, file-based relational DB for storing meeting metadata, transcripts, and insights.
* **Local or cloud-based file storage**: Temporary storage for uploaded files and transcripts.

**AI Components**

* **Whisper.cpp**: Lightweight, on-device transcription of audio/video files to text.
* **Ollama (LLM)**: Processes transcript using prompts to extract:
  + Action items
  + Decisions
  + Participants
  + Summary
* **ChromaDB**:
  + Stores embeddings of transcripts or chunks.
  + Enables semantic search over meeting content.

**4. AI Pipeline Implementation**

1. **User Upload**  
   React frontend uploads audio/video → FastAPI backend → stored locally or temporarily.
2. **Transcription (Whisper.cpp)**  
   Backend calls Whisper.cpp via subprocess or Python bindings to get the transcript.
3. **Insight Extraction (Ollama)**  
   Transcripts chunked and passed to Ollama LLM with custom prompt templates.
4. **Persistence (SQLite)**  
   Extracted JSON structure stored in SQLite tables (meetings, insights, participants, etc.).
5. **Vector Search (ChromaDB)**  
   Indexed transcript chunks allow search via semantic similarity.
6. **Frontend Visualization**  
   REST APIs expose insights to React dashboard:
   * Timeline view of meeting
   * Action item cards
   * Search bar for Q&A over meeting data

**Technology Stack**

|  |  |
| --- | --- |
| **Layer** | **Technology** |
| Frontend | React + Tailwind CSS |
| Backend | Python FastAPI |
| Database | SQLite |
| AI Tools | Whisper.cpp, Ollama, ChromaDB |

**5. Challenges and Solutions**

| **Challenge** | **Solution** |
| --- | --- |
| Large audio files & slow transcription | Chunk audio; optimize Whisper.cpp use or switch to GPU-enabled execution |
| Prompt tuning for structured insights | Use few-shot prompt examples and post-processing |
| Search relevance in vector DB | Chunk transcript intelligently; use title + timestamp metadata |
| Scaling for future users | Containerize services (Docker) and plan for migration to cloud-hosted DB |

**6. Evaluation Criteria**

* ✅ Functional flow: upload → transcribe → extract → search → view
* ✅ Modular and clean code (React, FastAPI)
* ✅ Integration of local LLMs and vector DB
* ✅ Search and insight quality
* ✅ Error-resilient backend and clear UX in frontend

**7. Code Quality & Organization**

**Backend (FastAPI)**

* Structured into routers, services, and utility modules
* pydantic models for data validation
* Async-friendly endpoints and background jobs
* Logging via loguru or standard Python logging
* Easy to run via uvicorn

**Frontend (React)**

* Components: Upload, Dashboard, InsightCard, SearchBar, ChartPanel
* Tailwind for consistent UI
* Axios for API calls
* Hooks and context for global state (like insights)

**8. API Design**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| POST | /api/upload | Upload meeting file |
| GET | /api/meeting/{id} | Get extracted insights |
| GET | /api/search?q=term | Semantic search from ChromaDB |
| GET | /api/analytics | Fetch insight stats and visual data |

**9. Error Handling**

* **Frontend**:
  + Snackbar for upload/transcription failure
  + Form validations on file type and size
* **Backend**:
  + Exception middleware using FastAPI
  + Input validations using pydantic
  + Error logging with stack trace

**10. Documentation**

**README Includes:**

* 📦 Project setup (React + FastAPI + SQLite + Whisper.cpp + Ollama)
* 🔧 Local environment setup with virtualenv or Docker
* 🧠 Instructions to run transcription and insight extraction
* 🌐 API usage with curl/postman samples
* 🔍 Sample queries for search endpoint
* 🛠️ Troubleshooting common setup issues