📘 Documentation Technique Détaillée

📚 Contexte du Projet

Ce projet a pour but de créer un système conversationnel avancé reposant sur des LLMs (Claude Sonnet, OpenAI, ou locaux via LM Studio) et enrichi par un mécanisme RAG (Retrieval-Augmented Generation). Il intègre également un module de gestion de notes personnelles vectorisées et activables.

L'application doit être utilisable **en local**, avec des options hybrides vers des API payantes si configurées.

---

1. 🗄️ Base de données (PostgreSQL)

1.1 Schéma ERD (Entity Relationship Diagram)

- Entités principales : `conversations`, `messages`, `rag_corpus`, `documents`, `document_chunks`, `notes`, `note_chunks`, `llm_configs`, `conversation_context`
- Relations :
  - `conversation` 1<->n `messages`
  - `rag_corpus` 1<->n `documents`
  - `documents` 1<->n `document_chunks`
  - `notes` 1<->n `note_chunks`
  - `llm_configs` 1<->n `conversations`

1.2 Détails des tables

Table: `conversations`

Champ	Type	Contraintes
id	SERIAL	PK
title	TEXT	NOT NULL
created_at	TIMESTAMP	DEFAULT now()
llm_config_id	INTEGER	FK -> llm_configs(id)

Table: `messages`

Champ	Type	Contraintes
id	SERIAL	PK
conversation_id	INTEGER	FK -> conversations(id), ON DELETE CASCADE
role	TEXT	CHECK ('user', 'assistant')
content	TEXT	NOT NULL

Champ	Type	Contraintes
created_at	TIMESTAMP DEFAULT now()	

Table: `document_chunks`

| chunk_text | TEXT | NOT NULL |
| embedding | VECTOR | Index vectoriel (CHROMADB) |

1.3 Index

- GIN/IVFFlat sur `embedding`
- Index temporels sur `created_at`
- Index FK pour optimisations de jointures

---

2. 📊 Architecture logicielle

2.1 Vue d'ensemble

```
graph TD;
  UI[Interface utilisateur] --> API[API Backend - FastAPI];
  API --> DB[PostgreSQL];
  API --> VS[Vector Store (CHROMADB/Chroma)];
  API --> LLM[LLM (LM Studio ou API Claude)];
```

2.2 Structure des dossiers Backend

```
/backend
├── api/
│   └── routes/                # Endpoints REST
├── rag/
│   ├── loader.py              # Lecture PDF
│   ├── chunker.py             # Split texte
│   ├── embedder.py            # Embeddings
│   └── store.py               # CHROMADB index
├── llm/
│   ├── router.py
│   └── providers/
│       ├── anthropic.py
│       ├── openai.py
│       └── local.py
├── conversations/
│   ├── controller.py
│   └── context_manager.py
```

---

3. 🔍 RAG & Notes Vectorielles

3.1 Pipeline RAG

1. Upload PDF
2. Extraction du texte
3. Chunking (par paragraphe, ou 500 tokens avec overlap)
4. Embedding local (`e5-base`, `Instructor`, `all-MiniLM-L6-v2`)
5. Indexation CHROMADB
6. Utilisés dans les conversations activées par l'utilisateur

3.2 Notes personnelles

- CRUD sur notes (titre, contenu)
- Même pipeline : chunking + vectorisation
- Stockées dans un index CHROMADB à part ou mêlé
- Utilisées dans les conversations activées par l'utilisateur

---

4. 🔄 Appels LLM

4.1 Abstraction dynamique

- Configuration dans `llm_configs`
- Choix : local (LM Studio) ou distant (Claude, GPT-4)
- Construction du prompt :
    - Prompt utilisateur
        - Ecrire…
        - Top-k chunks RAG + notes activées
        - Ecrire…
        - Instructions système

4.2 Exemple de requête API

```
{
  "model": "claude-3-sonnet",
  "prompt": "Voici le contexte ...\nQuestion: ...",
  "temperature": 0.7,
  "max_tokens": 1024
}
```

---

5. 📊 Format des données JSON

Requête utilisateur

```
{
  "conversation_id": 42,
  "content": "Que dit ce document ?",
  "llm_config_id": 1,
  "active_rags": [2],
  "active_notes": [5, 8]
}
```

Réponse du LLM

```
{
  "role": "assistant",
  "content": "Voici un résumé...",
  "sources": [
    {"type": "rag", "id": 2, "score": 0.89},
    {"type": "note", "id": 8, "score": 0.72}
  ]
}
```

---

6. 🤞 Tests recommandés

Tests unitaires

- Vectorisation correcte (mock embedding)
- Routes FastAPI
- Chunking / parsing PDF

Tests fonctionnels

- Upload de document + RAG
- Activation de notes + recherche contextuelle
- Commutation LLM local/API

---

7. 🛎️ Technologies / Coûts

Composant	Outil	Gratuit ?
LLM local	LM Studio + Mistral	Oui
Embedding	SentenceTransformers	Oui
Vector Store	CHROMADB	Oui
API payante	Claude / OpenAI	Optionnel
DB	PostgreSQL	Oui
Backend	FastAPI	Oui
UI	Next.js / React	Oui

---

8. 📃 Instructions d'installation (local)

```
# Installer les dépendances Python
pip install -r requirements.txt

# Démarrer CHROMADB + PostgreSQL (Docker possible)

# Lancer LM Studio en local : http://localhost:1234
```

```
# Lancer le backend FastAPI
uvicorn main:app --reload
```

---

9. ⚖️ Bonnes pratiques

- Toute configuration LLM modifiable dans `llm_configs`
- Structuration stricte des prompts via `system + context + user`
- Modularité des providers pour en ajouter facilement
- Logs et erreurs centralisés dans `/logs`
