Analyse Ligne par Ligne du Script FastAPI

Imports

■from datetime import datetime■import json■from typing import List, Dict■import uuid■from fastapi import FastAPI, HTTPExce

- Tu importes les modules nécessaires pour :
    - gérer les dates, UUID, JSON
    - créer une API avec FastAPI
    - faire des requêtes HTTP asynchrones avec `httpx`
    - gérer les modèles de données avec `pydantic`
    - autoriser les requêtes cross-origin (CORS)

Initialisation de l’application FastAPI

■app = FastAPI()■

- Tu crées une instance de l’application FastAPI.

Configuration CORS

■origins = [■ "http://localhost:4200",■ "http://127.0.0.1:4200",■]■■app.add_middleware(■ CORSMiddleware,■ allow_

- Tu autorises les requêtes provenant de ton frontend (probablement Angular ou React).
- CORS est essentiel pour permettre à ton frontend d’appeler ton API sans blocage navigateur.

Modèles de données (DTOs)

■class ModelDto(BaseModel):■ model: str■ version: str■■class Query(BaseModel):■ prompt: str■ model: str = "mistra

- Tu définis des schémas de données pour :
    - les modèles disponibles (`ModelDto`)
    - les requêtes utilisateur (`Query`)
    - les messages de chat (`ChatMessage`)
    - les conversations complètes (`Conversation`)

Stockage en mémoire des conversations

■conversations: Dict[str, Conversation] = {}■

- Tu utilises un dictionnaire Python pour stocker les conversations en mémoire (non persistant).

Endpoint `/generate` — génération en streaming

■@app.post("/generate")■async def generate_text(query: Query):■ async def stream_ollama():■ ...■ return Streaming

- Cet endpoint appelle un serveur Ollama/Mistral local pour générer du texte en **streaming**.
- Il utilise `httpx.AsyncClient` pour faire une requête POST avec `stream=True`.
- Il renvoie les lignes générées une par une.

Endpoint `/api/models` — liste des modèles disponibles

■@app.get("/api/models")■async def list_models():■ ...■

- Cet endpoint interroge le serveur Ollama pour obtenir la liste des modèles disponibles.
- Il transforme la réponse en objets `ModelDto`.

Endpoint `/api/conversation/start` — démarrer une conversation

■@app.post("/api/conversation/start")■async def start_conversation(conv_id: str):■ ...■

- Crée une nouvelle conversation avec un ID donné.
- Empêche la duplication d'ID.

Endpoint `/api/conversation/{conv_id}/message` — ajouter un message et générer une réponse

■@app.post("/api/conversation/{conv_id}/message")■async def add_message(conv_id: str, query: Query):■ ...■

- Ajoute un message utilisateur à une conversation existante.
- Envoie le prompt au serveur Mistral/Ollama.
- Gère deux formats de réponse :
    - JSON unique
    - JSONL (lignes JSON)
- Ajoute la réponse du modèle comme message assistant.

Endpoint `/api/conversation/{conv_id}` — récupérer une conversation

■@app.get("/api/conversation/{conv_id}")■async def get_conversation(conv_id: str):■ ...■

- Retourne tous les messages d'une conversation donnée.

Lancement de l'application

■if __name__ == "__main__":■ import uvicorn■ uvicorn.run("main:app", host="0.0.0.0", port=8000, reload=True)■

■ **Erreur ici** : `_name_` devrait être `__name__` et `_main_` → `"__main__"`

■ Corrige comme ceci :

```
if __name__ == "__main__":
    import uvicorn
    uvicorn.run("main:app", host="0.0.0.0", port=8000, reload=True)
```