Project Documentation – File Overview and Usage

October 1, 2025

Overview

Dieses Projekt implementiert ein Backend für ein **semantisches Chat-/Quizsystem** mit folgenden Kernfunktionen:

- PDF-Dateien hochladen, in Vektoren umwandeln und mit FAISS indexieren.
- Benutzeranfragen über `LangChain` + `Ollama` gegen diese Wissensbasis beantworten.
- Quizmodus mit Zustandsverwaltung und Audioausgabe (TTS).
- Sprachaufnahmen aufnehmen und per Whisper transkribieren (STT).

Die **Steuerzentrale** ist die Datei:

- **server2.mjs** – Startet den Express-Server, bietet die API-Routen an, ruft die Python-Skripte auf und verwaltet Datenbank, Quiz-State und TTS.

Aktiv verwendet (Server-Laufzeit)

Folgende Dateien/Skripte werden direkt von `server2.mjs` aufgerufen oder indirekt benötigt:

- **db.js**, **db_memory.js**, **quiz_state.js** – Datenbank- und Zustandsverwaltung.
- **langchain_indexer.py**, **langchain_query.py** – Kernstücke für Vektorindizierung und semantische Abfragen.
- **text_to_speech.py** – TTS für Antworten.
- **whisper_stt.py** – STT für Audioeingaben.
- **record_audio.py**, **record.js** – Aufnahme von Nutzereingaben (Audio).

Nur Hilfs- oder Testskripte

Diese Dateien sind optional oder dienen Testzwecken:

- **vector_store.py**, **query_faiss.py** – Alternativer Weg zur Erstellung und Abfrage von Vektorstores (ohne LangChain).
- **reindex_all.py**, **auto_index.py** – Automatisches/Neu-Indizieren aller PDFs.
- **encode.py** – Hilfstool zum Umkodieren von Audiodateien.
- **test_vector_load.py** – Einfacher Test, ob ein FAISS-Index geladen werden kann.

Erklärung der Dateien

server2.mjs

Node.js Express-Server. Stellt REST-APIs bereit:

- `/api/upload-pdf` – lädt PDF hoch, ruft `langchain_indexer.py`.
- `/api/semantic-chat` – Chat/Quiz, ruft `langchain_query.py`.
- `/api/stt` – Whisper STT.
- `/api/record-and-process` – Komplettablauf: Aufnahme → Transkription → Query → TTS.

Speichert Nachrichten in DB, verwaltet Quiz-State, generiert TTS-Antworten.

db.js

Initialisiert die DB-Verbindung (MSSQL oder Postgres). Bietet Funktionen:

- `initDb()` – stellt Verbindung her.
- `getPool()` – liefert den DB-Pool.
- `getDbType()` – prüft DB-Typ.

db_memory.js

Dient zur Verwaltung der Konversationshistorie in der Datenbank:

- `getOrCreateConversation()` – erstellt oder lädt eine Konversation.
- `getConversationLog()` – liest alle Nachrichten.
- `saveMessage()` – speichert eine Nachricht.

quiz_state.js

Einfaches State-Management über JSON-Dateien im Ordner `quiz_memory/`. Funktionen: `getState()`, `setState()`, `clearState()`.

langchain_indexer.py

Python-Skript:

- Lädt PDF.
- Reinigt und zerlegt Text in Chunks.
- Erstellt Embeddings (HuggingFace).
- Baut FAISS-Vektorspeicher.
- Speichert Chunks zusätzlich als Pickle-Datei.

langchain_query.py

Python-Skript für semantische Abfragen:

- Lädt FAISS-Vektorstore.
- Führt Similarity Search für Nutzerfrage aus.
- Baut Prompt (normaler Chat oder Quiz).
- Ruft Ollama-LLM auf.
- Liefert Antwort und aktualisiert Konversationshistorie.

text_to_speech.py

Nutzt `pyttsx3`, um Text in WAV zu synthetisieren. Wird von Server aufgerufen, um Audioantworten zu erzeugen.

whisper_stt.py

Nutzt OpenAI Whisper (lokales Modell), um Audio nach Text zu transkribieren.

record_audio.py

Python-Skript, das mit `sounddevice` Mikrofonaufnahme (5 Sekunden) durchführt und als WAV speichert.

record.js

Alternative Node.js-Version für Audioaufnahme mit dem `mic`-Package.

encode.py

Hilfsskript: nimmt eine Audiodatei und re-encodiert sie in 16kHz, Mono, 16-bit PCM (kompatibel für STT/TTS).

vector_store.py

Eigenständiges Skript zum Erstellen von FAISS-Indizes (ohne LangChain). Pipeline: PDF → Text → Chunks → Embeddings → FAISS-Index + Pickle.

query_faiss.py

Einfacher Query-Client für FAISS-Index. Lädt Index + Chunks, encodiert Query und gibt Top-Matches aus.

reindex_all.py

Durchläuft alle Kursordner in `knowledge_base/`, löscht alte Indizes und erstellt neue über `langchain_indexer.py`.

auto_index.py

Ähnlich wie `reindex_all.py`, aber erzeugt nur fehlende Indizes (idempotent).

test_vector_load.py

Mini-Testskript: lädt einen FAISS-Index für einen Kurs und macht eine Similarity-Suche (funktioniert als Sanity Check).

Fazit

- **Im Produktivbetrieb** sind vor allem `server2.mjs`, die DB-Skripte, `langchain_indexer.py`, `langchain_query.py`, `text_to_speech.py`, `whisper_stt.py`, sowie die Audioaufnahme-Skripte relevant.
- **Die restlichen Skripte** sind nützlich für Tests, Debugging oder alternative Pipelines.