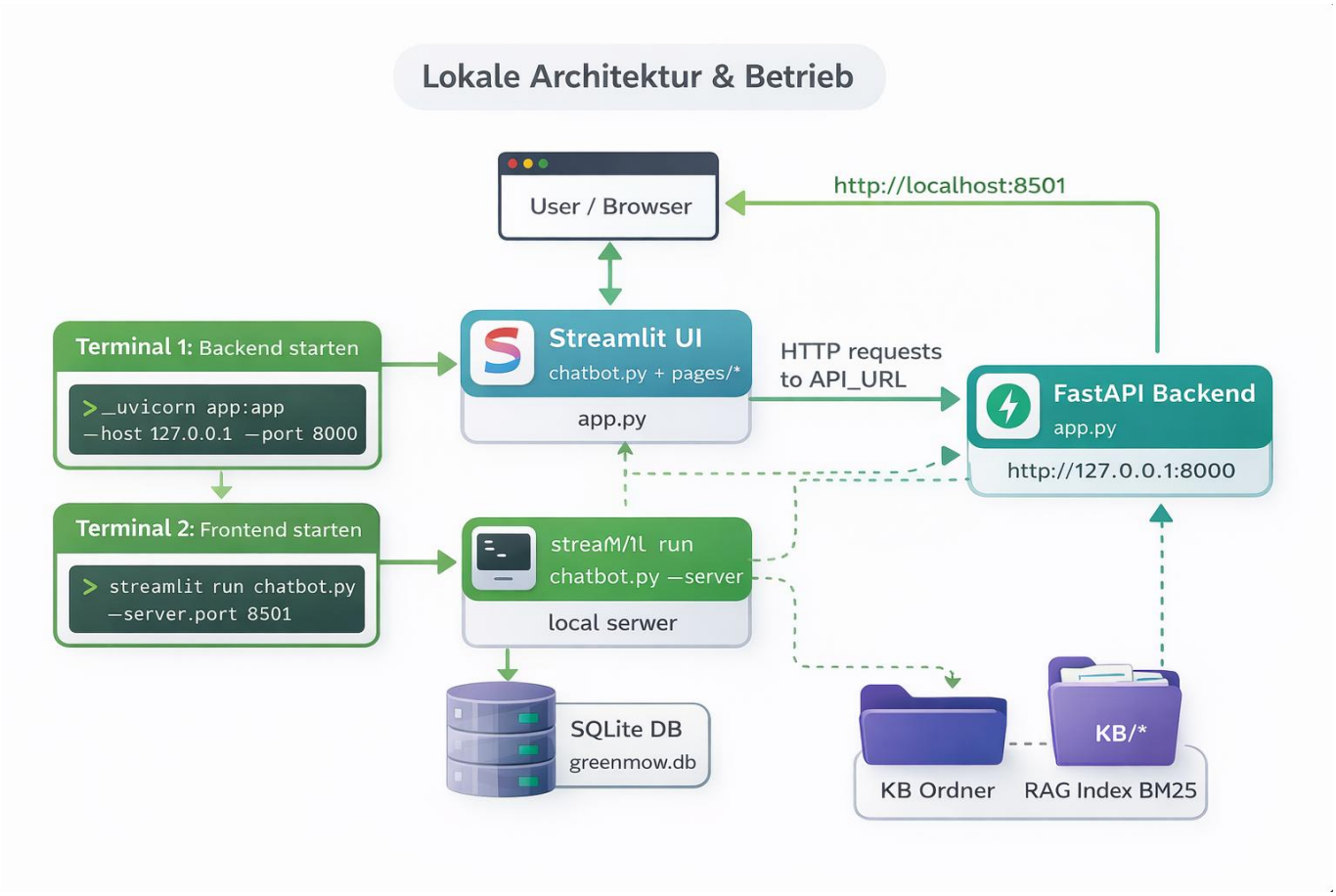


**1) Du startest das Backend (FastAPI) – Terminal 1**

**Befehl (Beispiel):**

uvicorn app:app --host 127.0.0.1 --port 8000

**Was passiert dabei?**

- Dein **FastAPI-Server** startet auf deinem PC.
- Er "lauscht" lokal unter: **http://127.0.0.1:8000**
- Dieser Server bietet deine API-Endpunkte an, z.B.:
  - POST /chat
  - GET /db/mowers
  - GET /db/work_orders (wenn du sowas hast/baust)
- Beim Start lädt er auch:
  - den **KB-Ordner** (RAG/BM25 Index)
  - die **SQLite DB** (z.B. greenmow.db)

➡️ Ergebnis: Backend läuft und wartet auf Anfragen von der UI.

---

**2) Du startest das Frontend (Streamlit) – Terminal 2**

**Befehl:**

streamlit run chatbot.py --server.port 8501

**Was passiert dabei?**

- Streamlit startet die Web-Oberfläche.
- Streamlit läuft lokal unter: **http://localhost:8501**
- Du öffnest diese URL im Browser.

➡️ Ergebnis: Du siehst deine App (Sidebar + Pages + Buttons etc.).

---

**3) Der User nutzt die UI im Browser**

Beispiele:

- Chatbot-Frage stellen
- "Requirement Refinement"
- "Requirements → Testcases"
- "Test Data Request" (Work Orders anlegen/anzeigen)

**Wichtig:** Streamlit ist nur UI/Frontend.
Die eigentliche "Intelligenz" / DB-Abfragen passieren im Backend oder über OpenAI.

---

**4) Streamlit schickt Requests ans Backend (API_URL)**

In deinem Streamlit-Code steht sowas wie:

API_URL = os.getenv("API_URL", "http://127.0.0.1:8000")

Wenn du z.B. auf **"Refine Requirement"** klickst:

- Streamlit macht einen HTTP Request an:
  - POST http://127.0.0.1:8000/chat
- Im JSON steht z.B.:
  - message (dein Prompt)
  - use_rag und top_k
  - session_id

➡️ Ergebnis: Streamlit wartet auf die Antwort vom Backend und zeigt sie dann an.

---

**5) FastAPI verarbeitet die Anfrage (RAG + Tools + OpenAI)**

Wenn /chat ankommt, läuft im Backend grob:

**5.1 Sprache/Session verwalten**

- Er merkt sich pro session_id, ob **DE oder EN** genutzt wird.

**5.2 Optional: RAG (KB Ordner)**

Wenn use_rag=True:

- Backend sucht passende Textstellen aus deinem **kb/** Ordner (BM25).
- Das wird als "Kontext" an das Modell mitgegeben.

**5.3 Tool-Calling (SQLite DB)**

Wenn das Modell merkt: "Ich brauche DB-Daten"

- Dann ruft es Tools auf wie:
  - list_mowers

    - get_mower
    - update_mower_status
- Diese Tools lesen/schreiben in **greenmow.db**

**5.4 LLM Antwort erzeugen**

- Backend ruft Azure OpenAI auf
- bekommt die Antwort zurück
- sendet an Streamlit:
    - reply
    - sources (KB chunks)
    - lang
    - session_id

➡️ Ergebnis: Streamlit zeigt dir die Ausgabe + optional Quellen.

---

**6) SQLite DB & Work Orders (Test Data Request)**

Deine DB (greenmow.db) ist lokal eine Datei.

- **Streamlit "Test Data Request"** kann direkt in die DB schreiben/lesen (wenn du es so gebaut hast)
- oder du lässt **FastAPI** die DB machen (sauberer, wenn alles zentral sein soll)

Typischer DB-Inhalt:

- mowers (Mäher)
- work_orders (Test Data Requests)

➡️ Ergebnis: Daten bleiben persistent, solange die DB-Datei existiert.

---

**7) Was bedeutet "2 Terminals" praktisch?**

- **Terminal 1 (FastAPI)** muss laufen, sonst bekommt Streamlit API-Fehler.
- **Terminal 2 (Streamlit)** muss laufen, sonst gibt es keine UI.
- Beide müssen gleichzeitig laufen.

---