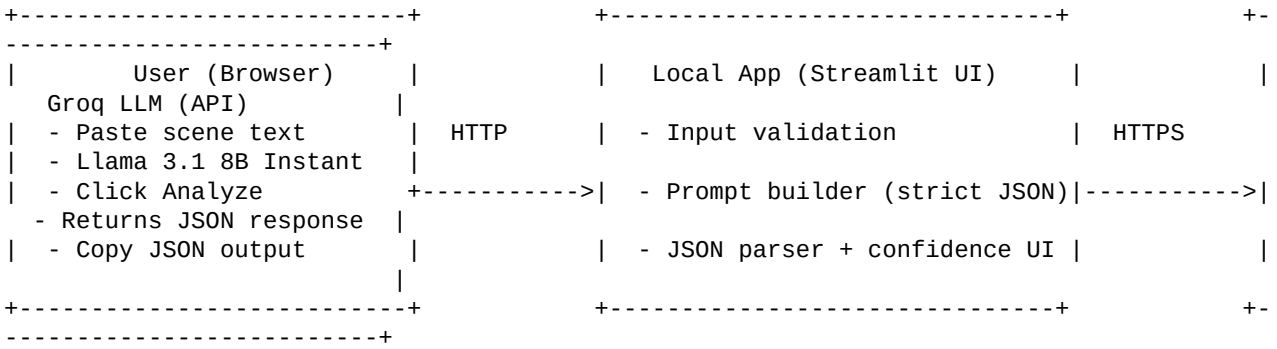
SceneSense AI

Architecture Explanation

Version: 1.0 | Focus: Hackathon MVP architecture + demo reliability

1. High-level architecture

SceneSense AI follows a simple, reliable 3-layer structure: a local web UI, an inference layer that builds a strict JSON prompt, and an LLM provider (Groq) that returns the structured scene intent.



2. Core components

- **UI Layer (Streamlit):** Collect scene input, allow loading example scenes, trigger analysis, render JSON output + confidence label.
- **Prompt & Inference Layer (Python):** Construct a strict schema prompt, call Groq chat completions, and parse returned JSON safely.
- **Model Provider (Groq):** Runs the LLM (e.g., Llama 3.1 8B Instant) and returns the structured response.
- **Config & Secrets (.env):** Stores API keys locally using environment variables (never hardcoded).

3. End-to-end request flow

- **Step 1 — Input:** User pastes a screenplay scene (or chooses an example scene).
- **Step 2 — Validation:** App checks that the scene is not empty and meets a minimum length (to avoid weak outputs).
- **Step 3 — Prompt build:** App builds a system+user prompt that forces strict JSON output with required keys.
- **Step 4 — LLM call:** App sends prompt to Groq chat completion endpoint with a low temperature for stability.
- **Step 5 — Parse:** App attempts to parse the response into JSON; on failure, shows a readable fallback.
- **Step 6 — Display:** UI renders JSON + confidence label (High/Medium/Low).

4. Prompting strategy (why JSON is stable)

The prompt is designed to reduce hallucinated formatting and enforce valid JSON. Key techniques: (1) explicit schema, (2) instruction to return ONLY JSON, (3) small key set, and (4) low temperature.

JSON schema enforced in prompt

```
{
  "emotion": "",
  "narrative_purpose": "",
  "visual_mood": "",
  "camera_style": "",
  "confidence": ""
}
```

5. Confidence scoring and labels

The LLM returns a numeric confidence score (0–1). The UI maps this into a simple label for humans:

Confidence score	Label shown in UI	Meaning
≥ 0.80	High	Strong signal; output likely matches scene intent
0.50 – 0.79	Medium	Reasonable; may need quick human review
< 0.50	Low	Weak signal; scene may be short/ambiguous or needs human interpretation

6. Error handling (demo reliability)

- **Empty/short scene:** show warning and do not call the API.
- **Missing dependency (Streamlit/dotenv):** documented install via requirements.txt; run using python -m streamlit.
- **Non-JSON output:** catch JSON parsing errors and show raw response in a safe fallback section.
- **Network/API latency:** keep screenshot-based backup outputs for offline presentations.

7. Security and secrets management

API keys are stored in a local **.env** file and loaded into environment variables at runtime. The repository includes a **.env.example** template and uses **.gitignore** to prevent committing secrets.

8. Local run (quick commands)

```
# (Windows PowerShell) inside project folder:
python -m venv venv
.\venv\Scripts\activate
python -m pip install -r requirements.txt
python -m streamlit run app.py
```

9. Future architecture upgrades (post-hackathon)

- **Batch processing:** multi-scene upload and parallel calls with caching.
- **Exports:** JSON → PDF/CSV; structured reports for teams.
- **Shot list module:** convert intent into suggested shots and transitions.
- **Role-based outputs:** Director Mode vs Writer Mode templates.

- **Optional local inference:** if offline requirement increases, add local model fallback.