Ink and Insights Design Documentation

PANW Hackathon

Author: Aditya Manikonda

Background

Journalling is an activity that has been proven to improve one's mental health, as letting your thoughts fly is often the first step towards stress relief and self-growth. People want to journal consistently but often stall at the blank page and rarely see patterns in what they write. Heavy "wellness" apps feel prescriptive, while raw note apps lack gentle guidance. *Ink and Insights* is a local-first journaling platform that reduces the friction to write on a daily basis and reflects back meaningful trends—without compromising privacy. It pairs a short, context-aware writing prompt with lightweight, on-device NLP and a clear, weekly analytics view.

Goals

The major goals of the app are to:

1. Promote writing consistency:
    a. Provide a platform that makes the user want to come back the next day and write another entry. This is done through streaks on the Trends dashboard
2. Guide users but not too much:
    a. Offer a 1-2 customized writing prompts to promote reflection
3. Deliver actionable weekly insights:
    a. Uses sentiment analysis to score each user prompt, helping users learn what is working for them and what isn't
        i. The "mood scores" that each journal entry gets assigned is completely geared towards the user, as it uses the z-score of the user's previous mood scores as a baseline and sees how much better/worse the next entry scores relatively.
4. Be convenient for the user:
    a. There is Google Calendar integration, which is used for providing more context when generating the writing prompts. This feature is optional, and can be turned off with a toggle.
    b. Users are greeted with a calendar-like interface, making it convenient to go back and see their previous posts
    c. There's an option to export weekly insights as a CSV if desired.
5. Protect user data and privacy:
    a. Stores all necessary data locally in a SQLite database and uses a lightweight local LLM and NLP model to generate writing prompts/insights summaries and "mood scores" respectively. There is also secure user authentication

        using OAuth, where users must sign in to their Google account to use the app.

6. Gracefully handle any unexpected situations:
    a. The user should not be able to cause uncalled for behaviors
7. Be fast and dependable:
    a. LLM results are stored in a cache to reduce redundant API calls and improve user experience.
8. Be clear and accessible:
    a. Uses readable charts, AM/PM labels, tooltips for the 0–100 mood scale, and a calendar history that lets users jump to any day's entries.
9. Be easy to improve upon:
    a. Keeps LLM model configurable, isolates NLP/analytics/auth/storage into modules, and supports export to CSV for portability.

System Design

Overview

Ink & Insights is a local-first journaling app composed of a Streamlit UI, a thin services layer, and a SQLite store. The UI orchestrates OAuth sign-in, writing, analytics, and history views; services handle Google auth/calendar, NLP+LLM, and persistence. We mirror the clear, component-driven structure and flow from your prior report's "System Design" section.

Tech Stack

The UI is built with Streamlit, which allows us to produce a clean, interactive interface (tabs, forms, state) with minimal boilerplate. It integrates well with Python data structures and supports quick experimentation without standing up a separate frontend stack. Even though it's harder to create custom UIs and doesn't scale as well compared to fully-fledged tech stacks, it was a trade-off I was happy to make for the rapid prototyping and quick setup.

For charts, I chose Altair because its declarative grammar makes it easy to produce consistent, readable visuals (entries/day, mood/day, emotion mix, best-hour) with little code and sensible defaults. Streamlit's default charts are also based in Altair, so it was easy to integrate my custom graphs into the app.

I chose to store all necessary data in SQLite, a lightweight, file-based database that keeps all journal content local to the user's device. It requires no server setup, is fast enough for our workload, and aligns with our privacy-first approach. Scale isn't that big of a concern, so I was happy with this decision.

I also used Pandas to assemble weekly windows, compute day-level aggregates, and prepare data for charts and exports. Its dataframe operations are expressive and reduce custom code for analytics. Since we don't have that much data to handle, Pandas works for our use case. If we had a lot more users and data to handle, then I would probably choose something like PySpark as it leverages multiple cores to process data in a distributed fashion. Polars would also be a really good alternative because it uses columnar storage for really fast reads.

For sentiment analysis, I used VADER (vaderSentiment) because it is small, fast, and well-suited to short, informal text like journal entries. It produces an interpretable compound score in between -1 and 1, which I mapped to a 0–100 “mood” scale for dashboards. This was my first time using VADER and I was pleasantly surprised with how accurate the scores were :)

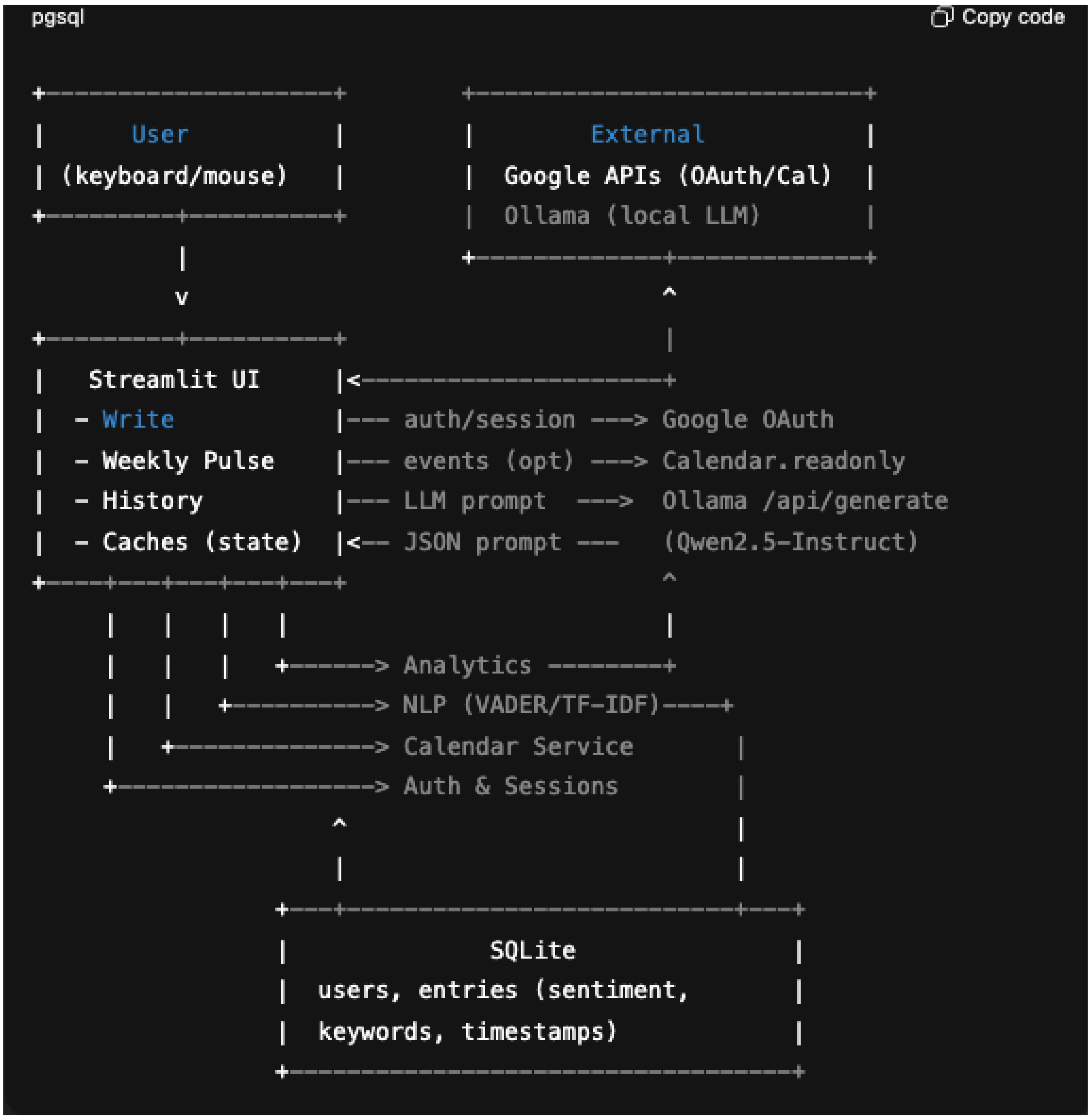
To extract themes, I opted for something like TF-IDF via scikit-learn’s TfidfVectorizer with 1–3-gram phrases. This on-device method is deterministic, requires no large model downloads, and surfaces concrete topics that are useful for prompts and weekly summaries.

Our LLM runtime is Ollama, which lets us run models locally for privacy, predictable latency, and zero per-call cost. It exposes a simple HTTP API, supports JSON-formatted outputs, and fits our local-first architecture. If I wanted a super fast and accurate model with minimal token restrictions, I would’ve gone for a signature model like GPT-4o or Gemini, but for this project, privacy outweighs performance.

Additionally, the model I pulled from Ollama is Qwen2.5-3B-Instruct, which provides solid instruction-following quality at a small footprint suitable for laptops. It’s strong enough to generate concise, grounded prompts and weekly summaries without incurring heavy inference times. I went with this model in particular because it is optimized for Apple Silicon Macs (which I’m using for development) and it’s free to use.

Lastly, for authentication, I used the Google OAuth 2.0 device flow, which avoids password handling, works well in local apps (no hosted redirect needed), and conveniently provides the user’s profile name and avatar. Most people have a Google account they can sign into, so it felt like the most obvious choice for me, as opposed to handling password management myself.

System Architecture Diagram:



```
+--------------------+        +---------------------------+
|       User         |        |        External           |
| (keyboard/mouse)   |        |  Google APIs (OAuth/Cal)  |
+---------+----------+        |  Ollama (local LLM)       |
          |                   +-------------+-------------+
          v                                 ^
+---------+----------+                      |
|   Streamlit UI     |<---------------------+
|  - Write           |--- auth/session ---> Google OAuth
|  - Weekly Pulse    |--- events (opt) ---> Calendar.readonly
|  - History         |--- LLM prompt   ---> Ollama /api/generate
|  - Caches (state)  |<-- JSON prompt ---   (Qwen2.5-Instruct)
+----+---+---+---+---+                      ^
     |   |   |   |                          |
     |   |   |   +------> Analytics --------+
     |   |   +----------> NLP (VADER/TF-IDF)----+
     |   +--------------> Calendar Service      |
     +------------------> Auth & Sessions       |
                  ^                             |
                  |                             |
         +---+---------------------------+---+
         |              SQLite               |
         |  users, entries (sentiment,       |
         |  keywords, timestamps)            |
         +-----------------------------------+
```

Core components:

- Frontend (Streamlit). Three tabs: Write (pre-write prompt + editor), Trends (weekly “Wrapped” analytics + LLM summary), History (calendar picker + per-day details). Lightweight state is kept in st.session_state (e.g., cached prompts/summaries) and URL query param sid for session continuity.
- Auth & Sessions. Google OAuth for sign-in; upon success we persist (or lookup) the user and create a short sid stored in the URL. Profile picture is fetched using the user’s Google token.
- Google Calendar integration (toggleable). If enabled, we read “today/tomorrow” events and pass at most one salient event into the prompt context.
- NLP (on-device). VADER for compound sentiment; TF-IDF (1–3-grams with custom stoplist) for per-entry keywords; a simple rolling z-score gives a personalized mood label.
- LLM (local via Ollama). Qwen2.5-3B-Instruct generates:

1. a pre-write prompt (opener + 1–2 guiding questions, prioritizing Calendar events if available)
2. a concise weekly summary of the analytics.
   Small token budgets, strict JSON outputs, and post-validation keep responses short, grammatical, and grounded.

- Storage (SQLite).
  users(id, google_sub, email, display_name, picture_url, created_at)
  entries(id, user_id, ts, text, sentiment_label, sentiment_score, top_keywords_json)
- Analytics. Weekly scope only: entries/day, average mood/day (mapped to 0–100 for clarity), best journaling hour, top themes, emotion mix, streaks, and CSV export.

Key flows

A user signs in with Google; we upsert or fetch their account and issue a short sid that lives in the page URL so a simple browser refresh restores the session (a full app restart still requires sign-in). On the "Write" tab, we assemble a truth-based context for the pre-write prompt: the user's stated goal, a few recent themes (TF-IDF keyphrases from recent entries), a simple 7-day vs 30-day mood trend, "days since last entry," and—if the Calendar toggle is on—at most one salient event (past today > later today > tomorrow). That context is sent to the local LLM, which returns a tiny JSON object (opener, optional habit line, 1–2 questions). We validate and sanitize the output, then cache it under a deterministic key so it doesn't churn while the user types; we show a spinner and a clear error if the model times out.

When the user saves an entry, we run on-device NLP: VADER computes a sentiment score and label, TF-IDF extracts keyphrases, and a rolling comparison to the user's recent history yields a personalized mood indicator. We persist everything to SQLite and show a succinct confirmation. On the "Trends" tab, we scope analytics to the last seven days (entries/day, mood/day on a 0–100 scale with a neutral rule, best journaling hour, top themes, emotion mix, streaks), and generate a concise weekly summary via the LLM; that summary is cached for the current week with a "Refresh" button. The "History" tab offers a calendar picker to jump to any day with entries and browse that day's details. Throughout, failures are handled gracefully (timeouts → retryable errors, empty windows → friendly "no data" messages), and privacy stays intact: journals live in local SQLite, the LLM runs on the user's machine, and Calendar context is strictly optional and minimal.

Evaluation/Challenges

I think in terms of engagement, insightfulness, reliability, and privacy/trust, the app did a really good job. It encouraged steady daily writing and reflected back meaningful, truthful patterns in a way that felt safe and non-judgmental. The pre-write prompt usually nudged a quick start, the weekly view made mood and themes easy to grasp, and the local-only design with optional, minimal calendar context built trust throughout. I thought the calibrated z-score baseline and caching of the LLM responses were also a nice touch to make the user experience smoother.

However, I think AI quality and performance is where there could be a lot more improvements. Because we opted to use a smaller, lightweight model, it was a lot more prone to hallucinations and was a lot slower then I would've liked. In the earlier stages of development, inference times were almost a minute long, and through using less tokens, doing some of the sanitization post-generation, doing extensive hyperparameter-tuning (by tuning the temperature, smaller token budget, etc.), and adjusting the prompt countless times, I was able to get it down to < 20-25 seconds consistently. The prompts still hallucinate and get presented with improper grammar sometimes, but it's a significant improvement. It's also worth noting that for the weekly summaries, the model is able to do quite well. The summaries are almost always accurate based on the eye test, and the suggestions that it offers are mostly relevant. This is likely because they're grounded in hard numbers and narrow context (dates, counts, 0–100 mood), so the model has less room to drift. The pre-write prompt is a looser task with more open-ended language and therefore more prone to generic phrasing and format slips, especially on a compact model.

If I were to tighten this further, I'd use a slightly different hybrid approach and benchmark it against a series of local Ollama models. I would keep an event-first, rule-based template and let the LLM only fill one short, personal question, not the whole prompt. I'd enforce a strict JSON schema and add a tiny post-processor that guarantees sentence case, terminal punctuation, and removes stray quotes. I'd also reduce variability, prefer noun-phrase keywords over vague words, and stream responses to cut perceived latency. Finally, I'd pre-warm the model on sign-in and cache prompts by context, with an optional fast hosted fallback for low-power machines, so we keep privacy by default but deliver consistent, polished prompts when users need speed.

Another feature that I really wanted to integrate if I had more time is a chatbot that users can interact with after submitting their journal entry. It would use the entry and their record of sentiment-scores to ask deeper reflection questions. It would allow the user to introspect even further, and provide an even more immersive experience. However, the prerequisite to this feature is getting reliable and fast responses from one of the local LLMs, which is something that I need more time to figure out.

Lastly, even though the UI looks minimalistic and sleek as per Streamlit's interface, I think some color would've been a nice touch to make the app feel more welcoming. This is where using a fully fledged frontend framework would shine.

Conclusion

All in all, building *Ink and Insights* was a really enriching experience for me, as I got more experience with developing under strict constraints while still prioritizing a safe, informative, and immersive experience for the end user. I learned that the small things, like adding spinners, converting sentiment scores to a more human-readable format, or integrating caching all contribute towards making a smooth-running app. Even though there is still a lot of room for improvement, especially on the AI accuracy/speed front, I am very proud of the work I put out over the past 48 hours.

Note: Also sorry if this whole design document lost structure/felt informal. I tried to speedrun through some parts of it due to time constraints. Thanks for understanding!

Acknowledgements

I used ChatGPT and GitHub Copilot extensively to help me develop most of my codebase and provide some assistance on this design doc. Final architecture choices, integration work, and testing were done by me—and any mistakes are my own.