SumizAI .com Your Second Brain, Powered by Intelligence.

https://sumizai.com/blog/sumizai-for-developers.html

USE CASES

What does SumizAI do for developers?

Developers already talk to models all day, and the output already vanishes all day. The specific loss is not code — code is in the repository. **It is the reasoning**: why this approach, what was rejected, what the measurement actually said.

Updated: 2026-08-14 **8** questions 3 min read

Q. What is actually worth capturing?

Decisions and their alternatives. Everything else is either in the code or not worth keeping.

A note that says "we use this queue" is worth almost nothing — the configuration says that. A note that says "we chose this queue over that one because redelivery semantics mattered more than throughput, and here is the measurement" is worth a great deal, and it is precisely what no artefact in the repository records.

The second category is debugging sessions that ended in an insight. The fix is a commit; the reason the bug was possible is a note.

Q. Why does the append-only source material matter here?

Because engineering claims decay, and you need to know when.

Every note keeps the complete original question and answer, and nothing later edits or removes them. When a merged note says an approach does not scale, you can read the original exchange and discover it was about a specific version, at a specific load, with a specific configuration.

A summary alone cannot tell you that, and a summary you cannot audit is how teams end up with received wisdom nobody can defend.

Q. How does this fit next to a repository?

Directly. A vault is a directory of `.md` files with relative links and a generated `base.md` index.

Commit it. Diffs are readable, review works, blame works, and the notes travel with the code. Nothing about the format fights version control, because it is the format version control was designed for.

It also means your existing tooling applies — grep, ripgrep, your editor's Markdown preview, a static site generator if you want an internal handbook.

Q. What does a good vault layout look like for engineering work?

One vault per system or per long-lived domain, not per ticket.

The boundary question is the useful one: when I ask about this, should the model see the other thing? For a service you own, yes — decisions about its storage, its API and its failure modes belong together. For an unrelated product, no.

The chapter tree then carries the internal structure, and it goes six levels deep if a subject needs it. Tickets are not chapters; subsystems are.

Q. Can I point it at a local model?

Yes. Ollama is one of the seven supported providers, and a custom server is another — anything speaking the OpenAI protocol works, because six of the seven providers share one client.

For a codebase you cannot send anywhere, that is the configuration that makes the app usable at all. Nothing leaves the machine.

One operational detail if you run the app in a container: an address like `localhost` for a provider on the host is rewritten to the host gateway, because Ollama's own documented address is unreachable from inside a container and that failure is confusing.

Q. What about the code the model writes?

Keep it in the repository. SumizAI is not a snippet manager and a note is a poor home for source that has to compile.

What belongs in the note is the reasoning around the code: what the approach is, what it assumes, what was tried first. Fragments in a note are fine as illustration and terrible as the canonical copy.

The line is roughly: if it needs to run, it goes in the repository; if it explains why the thing that runs looks like that, it goes in the vault.

Q. Does it help with onboarding?

In a limited but real way. There are no shared vaults, so this is not a team knowledge base — but the exports are shareable artefacts.

A vault exports as Markdown, a PDF or a deck in table-of-contents order. "Here is the reasoning behind this service, organised, forty pages" is a considerably better handover than a folder of links.

If you want a genuinely shared, versioned source of truth about what a system should do, that is a different tool with a different job.

Q. Where does it fall short for engineering use?

No sync between devices, so your laptop and your desktop are separate libraries.

No shared vaults, so this is a personal tool used inside a team rather than a team tool.

No semantic search — retrieval is lexical with trigram matching, and the semantic judgement happens in the model reading the shortlist. In practice this is fine and fast, but if you expected embeddings, there are none.