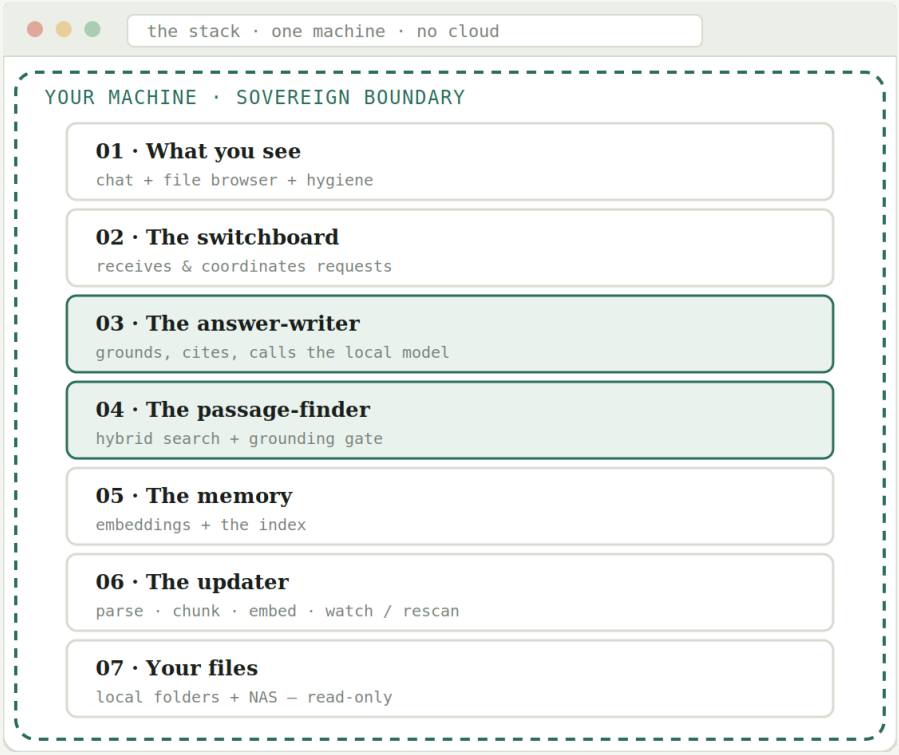
SIM RESEARCH INSTITUTE AG

A WHITEPAPER · 2026

The Tidy Machine

Building a sovereign, self-hosted AI assistant for your own documents — a local knowledge portal for private individuals and small businesses.



SOVEREIGN · LOCAL · CHAT-FIRST · GROUNDED

CONTENTS

What's inside

01 The problem: you can't find what you own
02 The core idea: a librarian for your files
03 Who it is for, and where it runs
04 Design principles
05 How it works
06 Architecture: one machine, layered
07 How a question flows
08 Technology stack
09 Deployment, operations & hardware
10 Data classification: the four levels
11 Privacy and data sovereignty
12 Security & governance for SMEs
13 Implementation: the prototype
14 Evaluation and quality
15 Limitations and honest boundaries
16 Roadmap / extension paths
17 Getting started
– Glossary

Finding what you already have is one of the most under-served problems in everyday computing. This paper describes a local knowledge portal: a small, self-built AI system that reads your own documents and lets you ask them questions in plain language — with answers grounded in, and linked back to, the source files. Critically, the model, the search, and the answering all run on your own hardware, so no file content ever leaves your control.

EXECUTIVE SUMMARY

Ask your files, on your own machine

Files accumulate across folders, drives, and devices faster than anyone can organise them, and conventional search only helps when you remember the exact words. A **local knowledge portal** reads your own documents and lets you ask them questions in plain language, with answers grounded in — and linked back to — the source files.

Three properties define the approach. First, it is **retrieval-augmented**: the language model answers only from passages retrieved from your files, never from its own general training. Second, it is **grounded**: when the evidence is too weak, it declines rather than guesses. Third, and most importantly, it is **fully local**: the model, the search, and the answering all run on your own hardware, so no file content ever leaves your control. The architecture is modest enough for one person to run, yet scales cleanly from a laptop hard drive to a NAS to a private server — equally suitable for an individual and for a small business.

PART 1

The problem: you can't find what you already own

Every long-lived collection of files tends toward disorder. Documents arrive constantly — downloaded, received, generated, copied "just in case" — and each takes seconds to file properly but a fraction of a second to dump anywhere. Over years, and across several people in an organisation, the result is predictable: duplicate versions

nobody trusts, scanned documents that aren't searchable, folders no one remembers creating, and knowledge that effectively disappears because no one can locate it.

The deeper cost is not storage; it is **lost knowledge**. A business that cannot quickly answer "what did we agree with this supplier?" or "where is the current version of this policy?" pays for that friction every day. An individual loses the same way, at smaller scale.

There are really two costs to a clean-up, and most people only count the second. The first is **knowing what you have** — before you can decide what to keep, merge, or discard, you must understand the contents of thousands of files. That is the genuinely impossible part, and it is why the project never starts. The second is **making the decisions** — keep this, drop that — which is real work but bounded, and the kind only you can do. A local knowledge portal collapses the first cost to almost nothing: the hard part of cleaning up was never the deleting, it was knowing what you had.

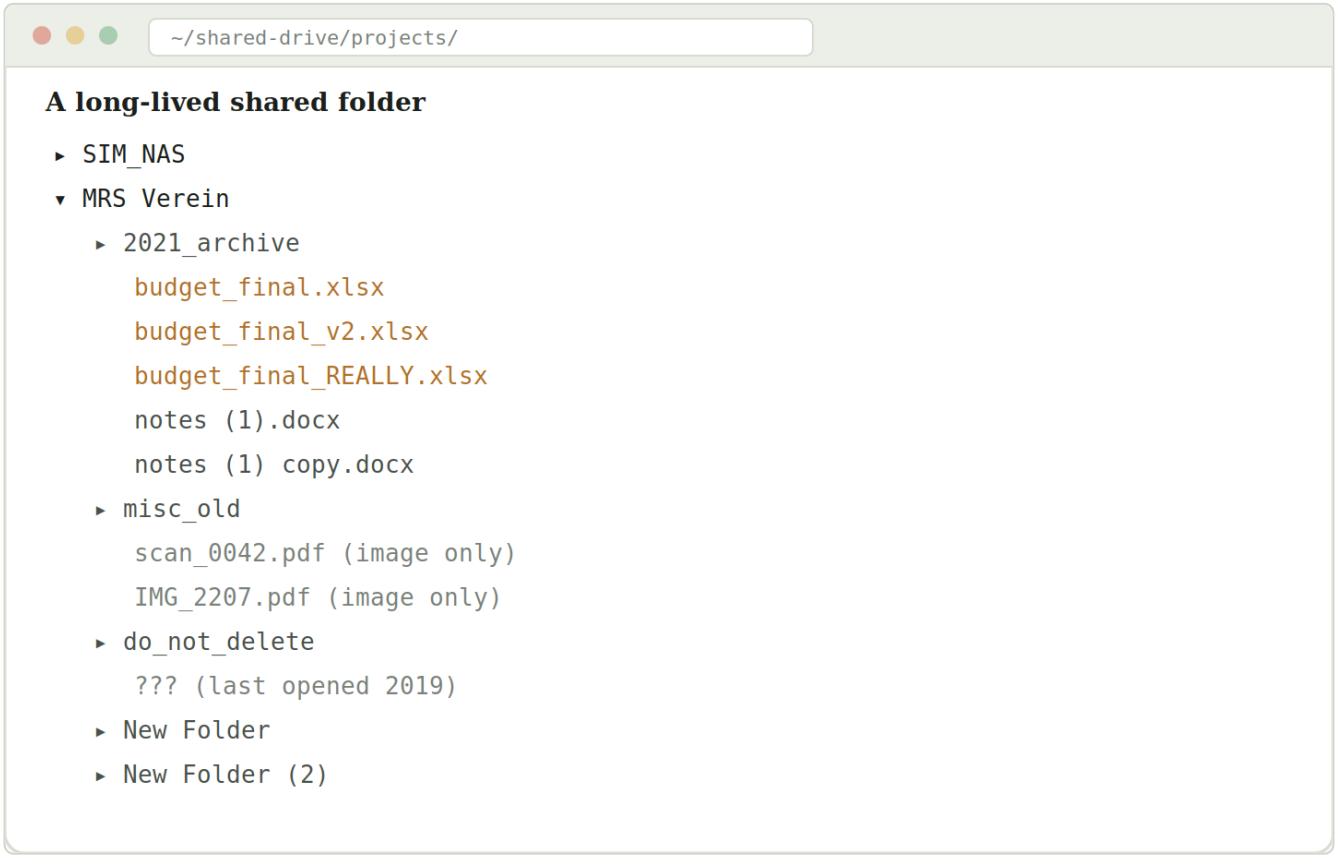


Fig. 1 — The natural state of a long-lived shared folder. Not a disaster anyone caused; just entropy doing its job. (Illustrative.)

Conventional tools fall short in specific, addressable ways: file search needs exact terms (it matches the words you typed, not the meaning you intended); it can't read images of text (a scanned page is invisible); and it can't answer in plain language (it returns a list of files, not an answer). The gap this paper addresses is the move from *searching for files* to *asking your files* — without sending anything to the cloud.

PART 2

The core idea: a librarian for your own files

Imagine a librarian whose only job is to read every document you own and remember what is in each one. You don't ask the librarian to throw anything away or reorganise your shelves. You ask questions — and the librarian finds the relevant passages, answers in plain language, and points to the exact documents and pages.

That is what a local knowledge portal does. You point it at your folders; it reads and indexes everything once, then keeps itself current in the background. From then on you have a single surface where you can:

- **Ask** — pose a question and get an answer grounded in your own files, with citations that link straight back to the source.
- **Search** — semantic *and* keyword search across everything indexed, for when you'd rather scan the hits yourself.
- **Browse** — the file tree, as a secondary view beside the chat.
- **Tidy** — a *Hygiene* view that scans a source and flags exact duplicates, versioned siblings, junk and scratch files, backup litter, oversized and obsolete-format files, and naming or folder problems — each with a count, reclaimable size, and safe actions (move to a review folder, or delete).

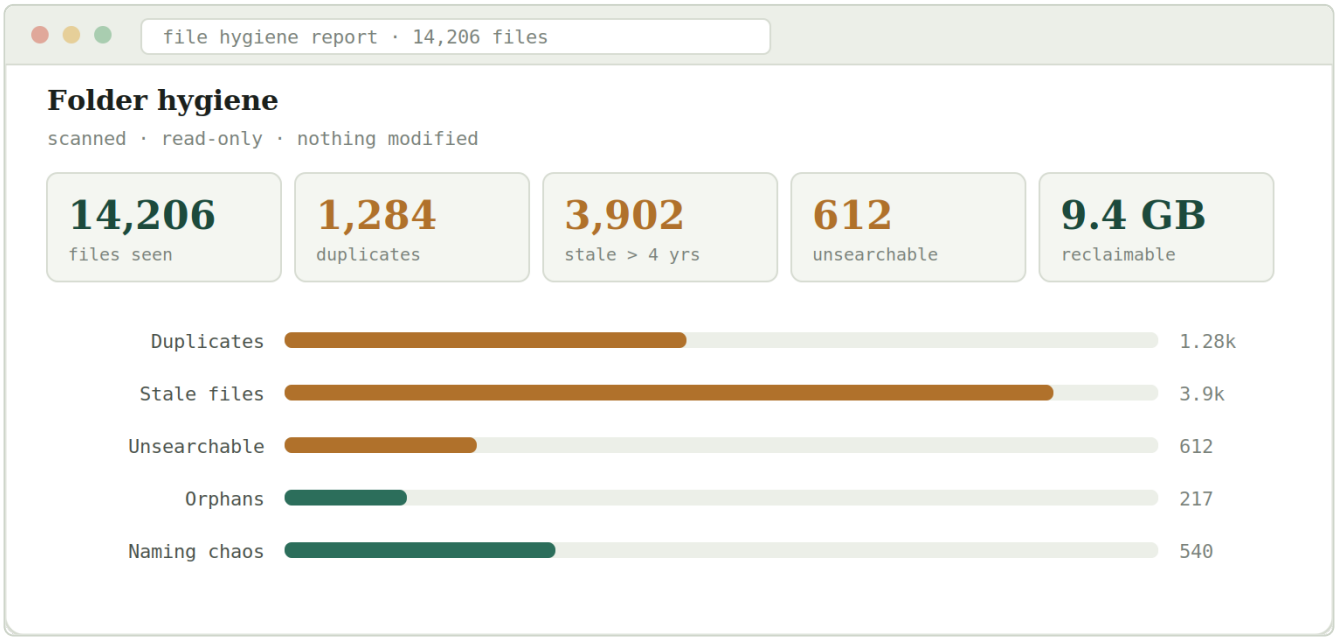


Fig. 2 — An illustrative hygiene report. The numbers are invented, but the categories are exactly the ones that make a file system rot. The point of the report is to turn a vague feeling of dread into a finite to-do list.

Why this is possible now. Two things matured together: the ability to turn text into a fingerprint of its *meaning* (so a question finds the right passage even when phrased differently), and capable language models small enough to run on an ordinary workstation — no data centre, no account, no fee. Together they let the whole librarian live on one machine.

> ONE-SENTENCE VERSION
>
> It is a new front door onto files you already have — not a new place to store them. Your files stay exactly where they are; the portal only reads and indexes them.

PART 3

Who it is for, and where it runs

The same system serves two audiences along one continuum: **private individuals** (a personal knowledge base over documents, correspondence, scans, and notes, on a laptop or home device) and **small businesses** (a shared knowledge

surface over contracts, policies, project files, and records, where institutional memory currently lives in scattered folders and people's heads).

What makes one design fit both is that it **scales with the storage, not the complexity**. The reference implementation uses a small NAS as the file source, but the architecture is identical across three deployment tiers:

Tier	Where files live	Typical user	Notes
Local hard drive	One machine's own disk	Individual	Simplest; everything on one device.
NAS (reference)	Network drive (SMB / NFS)	Individual / small team	Shared file source, mounted read-only.
Private server	An in-house server	Small business	More compute (e.g. GPU), larger corpus, several source folders.

In every tier the portal **runs locally** and the file source is **read-only**. Moving up a tier changes the amount of storage and compute, not the design.

PART 4

Design principles (non-negotiable)

1. **Sovereign / fully local.** All embedding and all inference run on your own hardware via Ollama. No cloud dependency, no exposure to foreign-jurisdiction data laws; no file content ever leaves the machine.
2. **Right-sized access.** For an individual: single-user, no accounts, every service bound to `localhost`. For a business: access governed at the file-system and network level rather than bolted on later.
3. **Chat-first.** The conversation is the primary surface; the file browser is secondary.
4. **Incremental and verifiable.** Built and adopted phase by phase, with a checkpoint at each step — no big-bang assembly, and every answer traceable to a source.

PART 5

How it works

Two journeys describe the whole system: documents travelling into an index, and a question travelling through that index to a grounded answer.

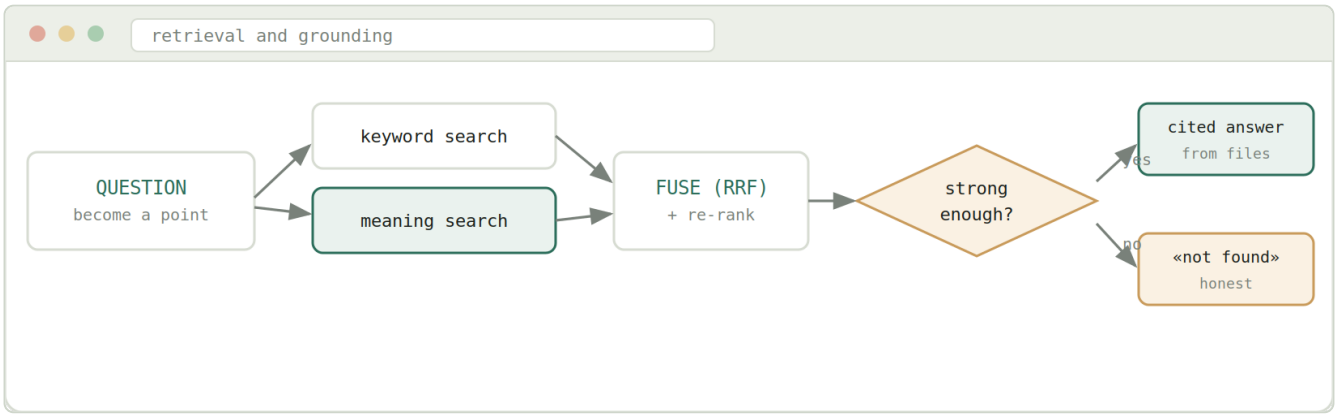


Fig. 5 — A question runs through both keyword and meaning search; the results are fused and re-ranked; then a grounding gate asks whether anything found is actually strong enough. If yes, the answer is written from those passages and cited. If no, the assistant says so.

5.1 Ingestion — reading your files into an index

Each file makes a five-step journey. **Discover:** the portal notices what is new or changed since last time, comparing each file's modified-time and a content hash (SHA-256), so unchanged files are skipped. **Parse:** text is extracted from PDFs, word-processing files, spreadsheets, and presentations; when a file is a scanned image with no text layer, OCR reads the picture so it becomes searchable. **Chunk:** each document is cut into coherent passages, broken at natural seams, never mid-word, each tagged with a breadcrumb of where it came from. **Embed:** each passage is turned into a numerical fingerprint of its *meaning* by a local embedding model. **Index:** passages and their vectors are stored in a vector database, ready for fast nearest-neighbour lookup.

> WHAT AN EMBEDDING IS
>
> An embedding turns a passage of text into a list of numbers arranged so that passages about similar things sit close together — so "annual leave" can find a passage about "holiday entitlement" even with no shared words. The question and the documents must be measured with the *same* embedding model; change the model and you must re-index.

5.2 Retrieval — finding the right passages, honestly

A question is embedded and run through **hybrid search**: a dense (meaning) search and a sparse / keyword search at once. The two result lists are combined with Reciprocal Rank Fusion, which captures both the recall of meaning and the precision of exact tokens — references, codes, proper nouns that pure semantic search tends to lose. An optional re-ranker re-reads the top candidates and reorders them by true relevance.

Then the decisive step — the **grounding gate**. Every retrieved passage carries a similarity score; if the best one falls below a threshold (cosine ≥ 0.55 in the reference build), the system treats it as "no solid evidence found" and lets *empty* context into the prompt, which reliably forces a refusal rather than a guess.

5.3 Generation — a grounded, cited answer

If good passages survive the gate, the local language model writes an answer **only from those passages**, under a strict prompt: use the supplied extracts and nothing else, cite each claim, and refuse plainly when there is no basis. If nothing survives the gate, the portal returns an honest "not found". The refusal is a feature, not a failure: an answer about your files should come from your files — or not at all.

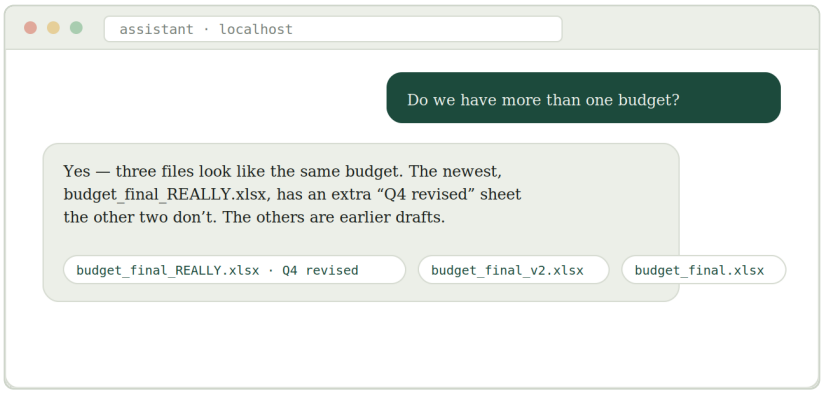


Fig. 6 — A grounded answer. Notice it answers in plain language but every claim is backed by a citation that links to the real file, so you can verify it in one click. (Illustrative.)

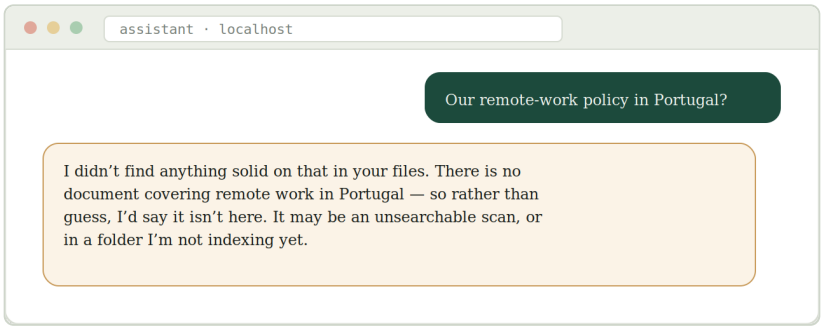


Fig. 7 — The refusal in action. It even suggests why the answer might be missing — an unsearchable scan, or a folder outside the index — which is often the next useful clue.

5.4 Keeping the index current

Local folders emit reliable change events, so a watcher updates the index the moment a file changes. Network shares are flaky for events, so the NAS is re-scanned on a schedule (and on demand), with change detected by modified-time and content hash so only new or changed files are reprocessed.

PART 6

Architecture: one machine, layered

The portal is a small stack of cooperating layers, each with one job, all inside one machine. A question travels *down* the stack; a cited answer travels back *up*; files travel *up* from the source into the index. Nothing crosses the outer boundary outward.

Layer	Job	Components
01 · Presentation	Split view — chat + file browser; citations open the real file.	Vite + React + Tailwind
02 · Application	The switchboard: receives requests, coordinates the layers, bound to localhost.	FastAPI + uvicorn
03 · Generation	Builds the prompt, enforces grounding, assembles citations, calls the local model.	Ollama (Qwen2.5 / Llama 3.x)
04 · Retrieval	Hybrid search, RRF fusion, re-rank, grounding gate.	bge-reranker-v2-m3 (optional)
05 · Embedding & index	Turns passages into vectors and stores them for fast search.	Ollama bge-m3, Qdrant

06 · Ingestion & sync	Parse · chunk · embed · watch / rescan.	PyMuPDF, python-docx, Tesseract, watchdog
07 · Source / data	Your folders and drives — read-only.	Local folders, NAS (SMB / NFS)

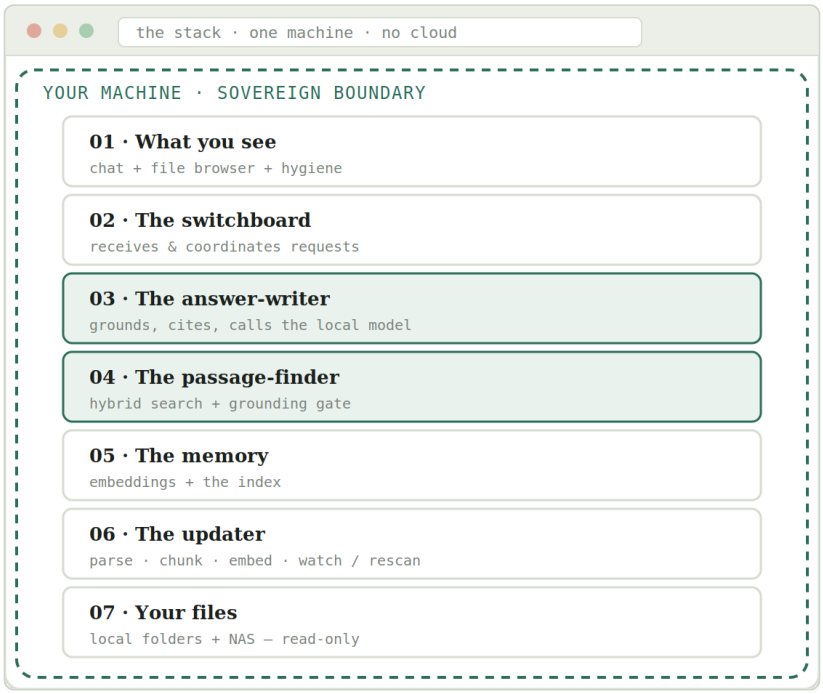


Fig. 8 — The seven layers. From the top: what you see, the switchboard, the answer-writer, the passage-finder, the memory, the updater, and your files at the base. The dashed line is the whole point — everything stays inside it.

What is striking is how small this is: a handful of programs on one workstation — modest enough for one person to run, understand, and switch off. For this use, that modesty is the advantage.

PART 7

How a question flows

1. You type a question in the **chat** (layer 01).
2. The **API** (02) receives it and starts the retrieve-and-answer routine.
3. The **retrieval layer** (04) embeds the question, runs hybrid search against the index (05), fuses and re-ranks the hits, and applies the grounding gate.
4. If good passages survive, the **generation layer** (03) builds a prompt from them, calls the local model, and assembles the answer with citations. If nothing survives, it returns "no grounded answer found".
5. The answer travels back **up** to the chat; each citation links to the real document.

Meanwhile, in the background, the **ingestion layer** (06) watches local folders and rescans the NAS (07), keeping the index current — an upward flow that never touches the query path.

PART 8

Technology stack

Concern	Choice	Why
Runtime / inference	**Ollama** on the host GPU	Runs the embedding and chat models locally; no API key, no cloud.
Embeddings	**bge-m3** (alt nomic-embed-text)	Multilingual (DE/EN); dense and sparse — pairs with Qdrant hybrid.
Vector store	**Qdrant** (Docker)	Native hybrid search + payload filtering; light enough for one user.
Reranker	**bge-reranker-v2-m3** (optional)	Local; lifts precision once base retrieval works.
Chat model	**Qwen2.5 (7B/14B)** or Llama 3.x	Sized to the available GPU.
Backend	**FastAPI + uvicorn** (Python 3.11+)	Bound to 127.0.0.1; endpoints /chat, /search, /files, /reindex, /health.
Parsing	**PyMuPDF, python-docx, openpyxl, python-pptx; Tesseract OCR**	OCR as the fallback for scanned PDFs and images.
File watching	**watchdog** + **APScheduler**	Reliable local events; scheduled rescan for the network share.
Frontend	**Vite + React + Tailwind**	Split view; brand tokens as CSS variables.
Packaging	**Docker Compose**	Qdrant + backend + UI as one stack; Ollama runs on the host.

> A NOTE ON INDEXING THROUGHPUT
>
> Turning a large corpus into embeddings is compute-bound. On a CPU-only machine this is slow for big collections; a GPU host makes full-corpus indexing practical. This is the main reason a business with a large archive moves to the private-server tier.

PART 9

Deployment, operations & hardware

Docker Compose bundles Qdrant, the backend, and the UI into one local stack; Ollama runs on the host. The NAS is mounted via SMB/NFS and indexed beside any local folders; it is a *file source only* and never runs inference. Network watching is unreliable, so the NAS uses a scheduled rescan plus a manual reindex button, with change detected by modified-time and content hash. All paths, model names, and thresholds live in a `.env` file, and a `/health` endpoint reports the stack's state.

9.1 Hardware — what you actually need

The honest headline: **this runs on far less hardware than you'd expect.** Because everything is local and modest, it fits on ordinary, portable machines — no server room required.

Tier	Machine	Key specs	Role

Entry	Lenovo ThinkPad X1 Nano	~0.9 kg · 13″ 2K · Intel Core i-series · integrated Iris Xe (no discrete GPU) · 16 GB RAM	Proof it runs on minimal hardware.
Power user	Lenovo ThinkPad X1 Extreme Gen 5	Core i9-12900H (14C/20T) · 32 GB DDR5 · RTX 3080 Ti 16 GB · 1 TB SSD · 1.88 kg	Faster, higher answer quality, still portable.
Business	NVIDIA DGX Spark (Founders Edition)	Blackwell GPU · Arm CPU (X925 + A725) · 128 GB unified · 4 TB NVMe · up to 100 Gb/s · ~1.2 kg · ≈ CHF 4,100	On-prem appliance for an archive.

The progression is real, not theoretical. The first working portal ran on a sub-1 kg ultraportable with only integrated graphics — slower, with a modest corpus, but genuinely answering grounded questions over its own files. A GPU laptop runs a 7–14B model and indexes a personal corpus quickly. For an organisation, the DGX Spark's 128 GB of *unified* memory lets one small appliance hold and run substantially larger models — and index a real corpus — entirely on-premises.

> HARDWARE, SOFTWARE, OR THE WHOLE THING — READY TO RUN
>
> SIM Research Institute AG operates this exact stack on these machines. Depending on what you need, we can help you source the hardware, supply a ready-to-go DGX Spark, provide the software — or run the entire process end to end as a managed service. From a single laptop to a business appliance, at any tier.

PART 10

Data classification: the four levels

Sensitivity is not all-or-nothing. A simple, widely-used model gives individuals and businesses a shared vocabulary for *how careful* to be with a given document — and it is the reason a local architecture matters.

Level	Name	Protection need	In practice
1	**Public**	none	Freely accessible; no restrictions.
2	**Internal**	low	For you / the organisation only.
3	**Confidential**	raised	Sensitive personal data is at least confidential; access control mandatory.
4	**Secret**	highest	Handled only in approved environments; strictest access **and location** control.

The levels are cumulative. The jump that matters is the last one. **Levels 1–3 are largely about *who* may see the data. Level 4 adds a constraint about *where* it may physically be processed** — and that single requirement reshapes everything technical: no cloud model, no cloud embeddings, no outbound path at all.

> *A portal built to satisfy the strictest level satisfies all the others for free.*

An individual indexing holiday photos and a business holding confidential records run the *same* architecture; only the deployment tier and the access governance around it differ. Each chunk's metadata can carry its classification mark, so every answer's provenance is traceable down to the level.

PART 11

Privacy and data sovereignty

This is the heart of the design. With a cloud tool, privacy is a *policy* — a promise you didn't read. With a local system, privacy is *structural*: there is no outbound path, so there is no service that *could* receive your files. Three properties make it trustworthy: **no outbound data path** (nothing can leak because nothing is sent); **read-only sources** (the portal indexes but never alters your files); and **grounding as a confidentiality control** (the model speaks about your content only from your content, never filling gaps from its own training).

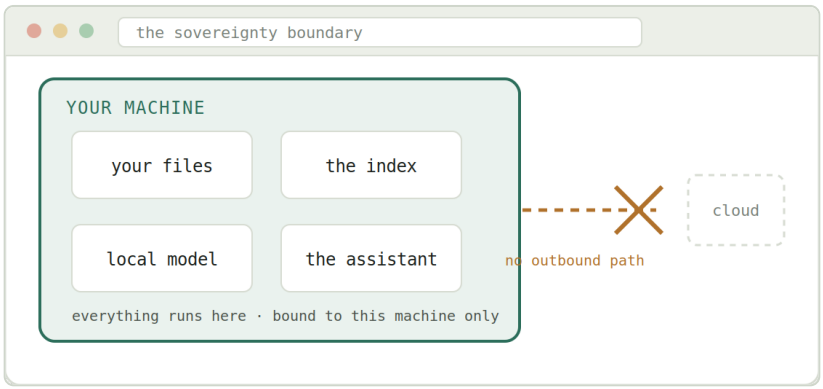


Fig. 10 — The sovereignty boundary. Every component lives inside one machine; there is simply no service to which your file contents could be sent. Privacy here is structural, not a promise on a settings page.

A practical caveat: encrypted documents can't be indexed, so the source layer must hold readable files whose access is governed at the file-system level.

PART 12

Security & governance for businesses

The base design is fit for individuals as-is. For a business handling regulated or highly sensitive material, a few additions harden it further, in the order they're worth building:

1. **Audit log.** An append-only record of who asked what, when, and what was cited — optionally hash-chained for tamper-evidence. Cheap and concretely useful.
2. **Classification metadata per chunk.** Tag each passage with its level so answers are filterable and traceable.
3. **Hardened network egress.** A custom Docker network with an explicit allowlist; trivial on an air-gapped machine, cheap to add anyway.
4. **Air-gap install path.** An offline bundle for a workstation with no internet at all.
5. **Approved-application governance.** The documentation — architecture, reproducible build, threat model, and test evidence — that supports formal approval where it's required.

PART 13

Implementation: the prototype

A working end-to-end portal runs on a single workstation: all seven layers, hybrid retrieval with the grounding gate, a strict citation prompt with a fixed refusal, local folders and a NAS as read-only sources bound to `127.0.0.1` with no outbound inference, an nginx-served frontend over a FastAPI backend with Qdrant and Ollama, all brought up by

Docker Compose. At the time of these captures the index held roughly a thousand chunks across three mounted sources.

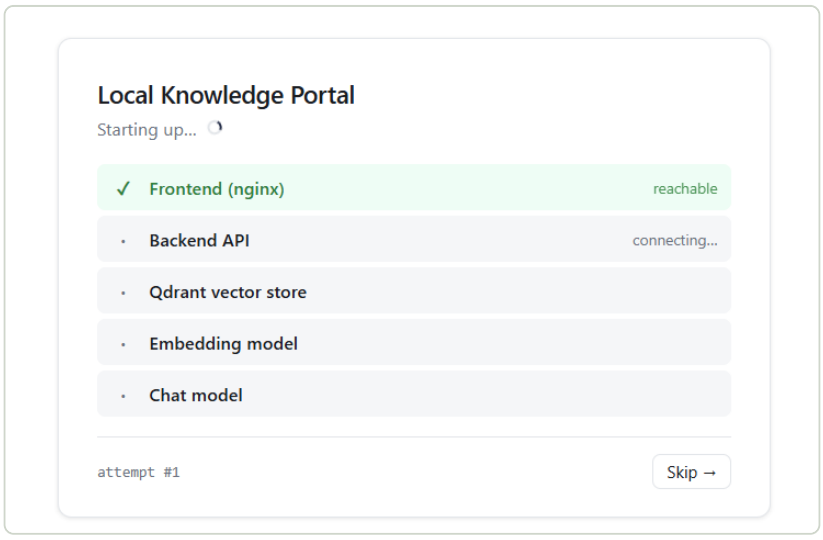


Fig. 13.1 — Starting the portal. The stack comes up layer by layer (frontend → backend → Qdrant → embedding → chat model); it reports "ok" only once the chain is reachable.

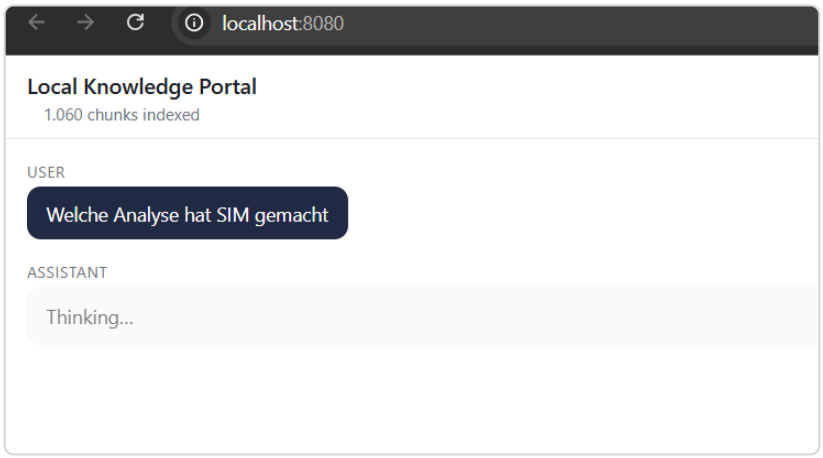


Fig. 13.2 — Asking a grounded question. A question is put to the portal; the assistant composes a citation-backed answer from the indexed files. The same answer shape works in German or English.

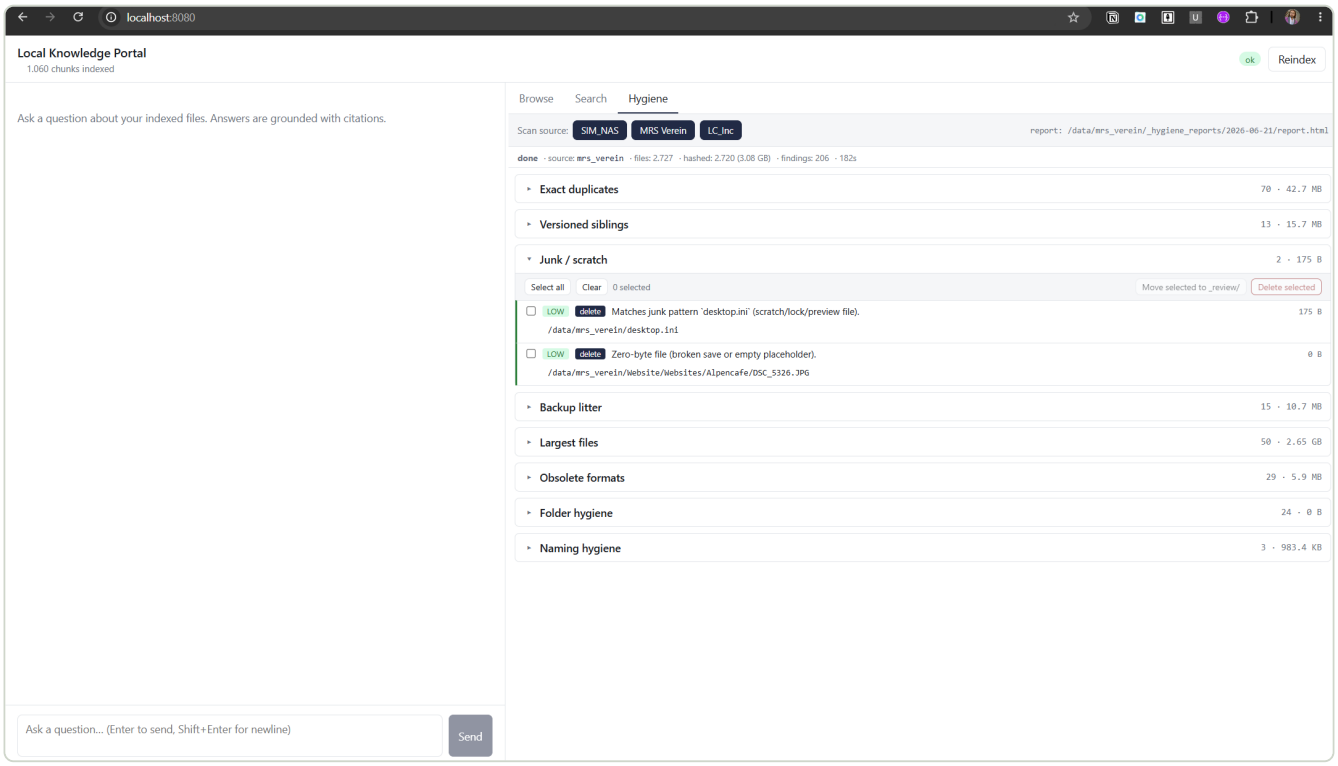


Fig. 13.3 — The Hygiene report. A scan over a source produces findings grouped into actionable categories (duplicates, versioned siblings, junk, backup litter, largest files, obsolete formats, folder and naming hygiene), each with counts, reclaimable size, and safe move/delete actions.

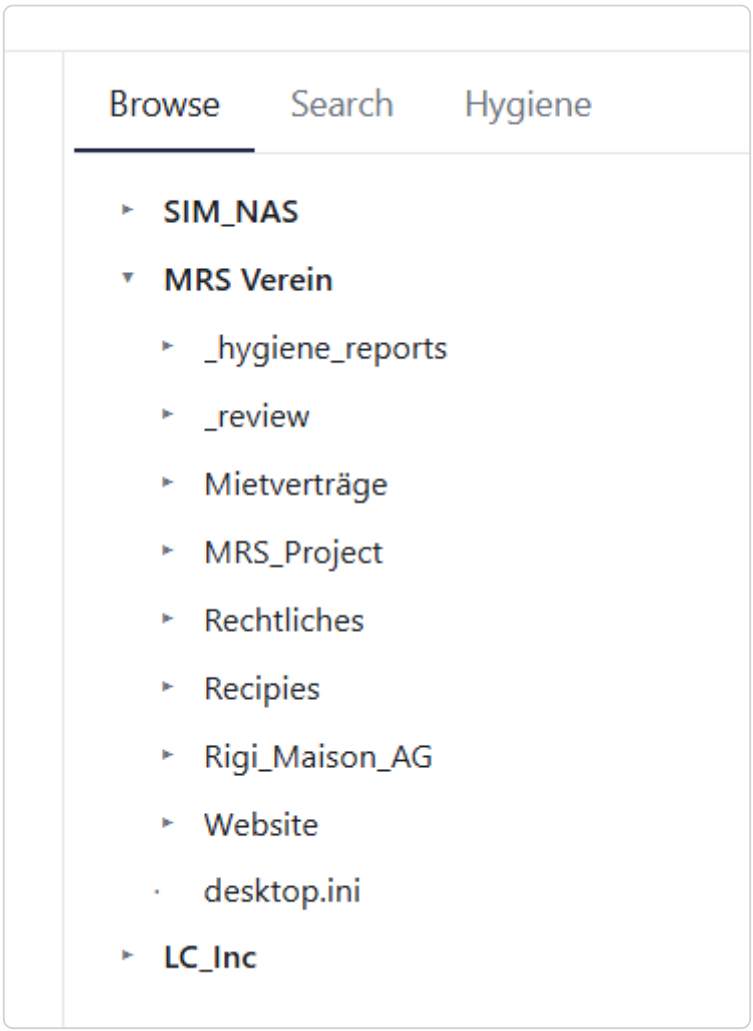


Fig. 13.4 — Browsing the indexed sources. The folder tree across the mounted sources, a secondary view beside the chat.

13.1 Working a clean-up with the Hygiene report

The report turns an unbounded chore into a finite, reviewable loop. Work one category at a time, rather than tidying everything at once:

1. **Start with duplicates.** Keep the canonical copy of each cluster and send the rest to a review folder — usually the biggest win at once.
2. **Make the invisible visible.** Run OCR over the unsearchable scans so they gain a text layer and become answerable after re-indexing.
3. **Sweep the stale and oversized files.** Review the years-untouched and largest files as a batch; most belong in an archive, not the live drive.
4. **Re-home orphans and obsolete formats.** Ask the portal what a file is when its name is useless; file the keepers, queue the rest.
5. **Tame the names last.** With volume already down, the remaining naming and folder issues are small enough to fix by hand.
6. **Re-scan and confirm.** Run the report again, watch the counts fall, then pick the next category.

> A GENTLE RULE

> **Move before you delete.** A "to-review" holding folder you empty after a month costs almost nothing and removes the one irreversible risk. The portal helps you *find* things; it should never be the thing that loses them.

PART 14

Evaluation and quality

A retrieval system that is never measured is optimised blind. The reference build keeps a small **evaluation set** — around 20–30 real questions, each with its expected source — and checks, after any change to the pipeline, whether the right source still lands in the top results. The same set calibrates the grounding threshold and decides whether optional refinements actually help before they are switched on. For a business, this set doubles as test evidence in any approval process.

PART 15

Limitations and honest boundaries

- **It won't make the decisions.** Judgement about what *matters* depends on context the files don't contain — that stays human.
- **It's only as good as what it can read.** Encrypted files, exotic formats, and folders you never pointed it at are invisible. A wrong or missing answer is often "that file isn't in the index".
- **It can be confidently wrong.** This is exactly why answers carry citations — for anything that matters, click through and confirm.
- **Maintenance is a habit.** Files keep arriving; a periodic re-scan keeps the index honest. The one-time setup is the hard part; upkeep is cheap.

PART 16

Roadmap / extension paths

Roughly in order of value per effort: an audit log; classification metadata per passage; hardened network egress; an air-gapped install path; GPU-host migration for large corpora; and, where regulation requires it, approved-application governance documentation. On the retrieval side: maintain the evaluation set, calibrate the grounding threshold against it, and add reranking or query refinements only where they demonstrably lift the hit-rate.

PART 17

Getting started

1. **Pick one folder you know well** — a small, familiar scope to check answers against your own knowledge.
2. **Run the inventory first, change nothing** — let the shape of the collection become visible before acting.
3. **Ask three real questions** — watching it answer correctly, and refuse honestly, is what builds trust.
4. **Widen the scope only once it's earned** — then point it at the folders you've forgotten.

> *The librarian has read every document in your folders. All that is left is to start asking.*

APPENDIX

Glossary

LLM — the component that writes the answer in plain language; here it runs locally. **RAG** (retrieval-augmented generation) — answering from passages retrieved from your own files. **Embedding** — a numerical fingerprint of a passage's meaning. **Hybrid search / RRF** — combining meaning-based and keyword-based search, fused by Reciprocal Rank Fusion. **Reranker** — a second pass that reorders top candidates by true relevance. **Grounding gate** — a quality floor; below it the system declines rather than guesses. **OCR** — reading text out of a scanned image. **Vector store** — a database built to return the nearest vectors to a query quickly. **NAS** — a network drive used as a read-only file source. **Air-gap** — a machine with no network path to the outside world.

© 2026 SIM Research Institute AG · The Tidy Machine — a whitepaper. Architecture diagrams are illustrative; prototype screenshots are from a working single-workstation build. Sovereign, local, grounded — by construction.