

SECTION A-A

BOX 03-02

LIMITATIONS.md

Write down what your system does not do

REV A · ISSUED

TITLE	SHEET	REV	DRWN	DATE
LIMITATIONS.md	03-02	A	D.KRAVCHUK	2026-10-03

BOX 03-02 REV A 2026-10-03 TEMPLATE

Write down what your system does not do

01 /

Why a file nobody asked for

Every system I ship comes with a file called `LIMITATIONS.md`. It sits next to the README and it is the second thing a reviewer opens. It lists, in plain words, what the system does not do, what it does in a simplified way, and what broke while I ran it.

I started writing them because software that only lists its strengths is an advertisement. A README says "handles leads from any source". The limitations file says "one lead per execution, batch inputs need a wrapper". The second sentence is the one that saves a week.

Three things happen when the file exists.

The buyer trusts the README more. Every claim on the front page now has a counterweight on the back page. If I was willing to write "dedupe is email or phone, not both", I probably was not inventing the part about the retry budget.

The next engineer starts from the right place. Half of the entries are decisions with a reason attached. "No reranking. The setting is reserved; turn it on when you can show the answer was sitting at position six." That is not a bug, it is a door with a sign on it.

I stop lying to myself. Writing "the webhook always returns HTTP 200" forced me to check why, and the answer (the form trigger refuses to coexist with a respond node) went into the file, with the workaround and its cost.

The file is not an apology and it is not a backlog. It is a drawing of the edges.

02 /

The template

Copy this into the root of the repo. Keep the four sections, delete the ones that stay empty, never rename them. A reader who has seen one of these files should be able to find the same things in the next one.

```
# What this does not do

Written before anyone asks, because software that only lists its
strengths is an advertisement. Most items are deliberate scope
decisions; the ones worth fixing for production say so.

## By design

**<Limit, as a noun phrase.>** <What it means in practice, one or
two sentences.> <What changing it would take, and why it was not
done here.>

**<Next limit.>** ...

## Operational
```

```
**<Limit that depends on the environment.>** <Where it bites:
localhost vs public, rate limits, auth, timeouts.> <What to change
first on a real deployment.>

## Found by running it

**<Something that broke.>** <Symptom as seen.> <Cause.> <Fix, or
"not fixed", and the date it was checked.>

## Known rough edges

- <Small thing, one line, no fix planned.>
- <Small thing, one line, no fix planned.>
```

Four sections, and they are not interchangeable.

SECTION	WHAT GOES HERE	TEST
By design	Scope you chose. Each entry is a decision.	Can you finish the sentence "because"?
Operational	Limits that depend on where it runs.	Does it change if the host, keys or network change?
Found by running it	Real failures from real runs, dated.	Did this cost you an afternoon?
Known rough edges	One-liners you will not fix.	Would a reader be annoyed, not blocked?

03 /

How to fill it

Lead with the limit, in bold, as a noun phrase. "One corpus, one audience." "No OCR and no images." "Thirty days of memory." The bold line is what people scan. If it reads like a feature name, you have it right. If it starts with "Unfortunately", delete that word.

Name the mechanism, not the feeling. Bad: "duplicate detection is not perfect". Good: "duplicate detection compares title word-sets; two reports of the same event with different headlines will both arrive". The second version tells the reader exactly what they will see and exactly where in the code it lives.

Say what changing it would take. This is the part that turns a weakness into a decision. "Per-user permissions are a filter on the search; the vector store supports it and the field is not there. Adding it means deciding who may see what, which is a conversation, not a feature flag." Now the reader knows the cost and knows you measured it.

Say which way it fails. A limit that fails safe is a different animal from one that fails loud, which is different again from one that fails silently. Write it down: "it errs towards a person, which is the safe direction" or "harmless, just a wasted enrichment call" or "the deleted text stayed answerable, the worst thing a support agent can do".

Date the checks. "Checked 18 Sep 2026 against two subreddits, both refused." An undated limit is folklore. A dated one is a measurement someone can repeat.

Keep heuristics that were tried and reverted. The most useful entries are the ones that stop the next person from trying the same clever idea. "Turning a short refusal into a clarifying question helped on vague questions and hurt on short specific ones. Word count cannot tell vague from absent. The reasoning is left in the code so nobody tries it twice."

One paragraph per limit. If a limit needs three paragraphs it is two limits, or it belongs in the architecture doc with a one-line pointer here.

Numbers only from your own runs. If you measured a threshold, give the range you saw. If you did not measure it, say "not measured" rather than quoting a number from a blog post.

Where it links from

The README gets one line, near the top, not at the bottom under "Contributing":

```
What it does not do: see [LIMITATIONS.md](LIMITATIONS.md).
```

The architecture doc links to specific entries when it describes a component whose behaviour is bounded by one. The test list references the "Found by running it" entries that have a test, so a reader can see which ones are guarded.

04 /

Worked example 1: an AI intake pipeline

A workflow that takes an inbound lead from a form, an inbox or a webhook, scores it, writes a CRM row and drafts a reply. The model is called only when there is something to judge.

```
## By design
```

```
**One lead per execution.** Each trigger event is one lead. Code nodes read cross-node state from the first item, so a batch of several leads in one run is not supported. Batch inputs need a loop wrapper before normalization.
```

```
**Nothing is sent to the lead.** Every branch, including the cold auto-reply, creates a draft. A real auto-reply is one operation change and a decision the client has to make consciously.
```

```
**Enrichment is the public home page only.** A GET of the domain, then title, meta description and the first 600 characters. Sites rendered in JavaScript yield nothing. No paid data provider is called.
```

```
**Business hours are one time zone.** One constant in the rule scorer. Multi-region teams need a per-lead zone.
```

```
## Operational
```

```
**Webhook callers always get HTTP 200.** The form trigger refuses to coexist with a respond-to-webhook node, so the intake answers in
```

last-node mode. The body is structured; the status code cannot be 400. A separate intake workflow would restore real codes at the cost of a second workflow to maintain.

****Retry budget is small on purpose.**** Three attempts, backoff capped at 20 seconds, so a run stays under the webhook timeout. Long outages go to the fallback contract and a review flag, not to a queue.

****OAuth for the mail provider is manual.**** The consent screen cannot be scripted. Until the credential exists, deploy imports the mail nodes disabled and the pipeline runs without drafts.

Found by running it

****Two identical leads in the same second race on the upsert.**** The CRM has no unique constraint, so both become rows. Seen once during a replay test. A queue or a lock is out of scope here.

****The small model once filed a pricing request as "other"**** because the company was outside the target profile. The prompt now separates intent (category) from fit (score). Checked 18 Sep 2026.

Known rough edges

- The free-mail list is a fixed heuristic; an unlisted provider is treated as a company domain. Harmless, one wasted lookup.
- Categories are four plus "unclassified". Enough for routing, not a taxonomy.

What to notice: every "By design" entry ends with what it would take to change. "Operational" entries name the host behaviour that causes them. The race condition is dated and bounded ("seen once").

05 /

Worked example 2: a meal-planning app

A web app that builds a weekly plan from what is already in the fridge, in three languages, with a paid subscription. The limitations file here protects the marketing copy from drifting ahead of the product.

By design

****No photo recognition.**** The fridge is filled by tapping products from a catalog, not by pointing a camera at a shelf. Recognition would put a model between the shelf and the plan, and a wrong product there makes the plan wrong in a way the user cannot see.

****No calorie counter.**** Nutrient values are computed per plan from the catalog, deterministically. There is no per-meal logging and no daily target. Adding one means a different data model for intake, not a new screen.

****Three languages, one catalog.**** Product matching across languages is a mapping table maintained by hand. A product missing from the mapping shows in one language and vanishes in another. The weekly audit catches these; nothing catches them live.

****One household per account.**** No family roles, no shared lists

between accounts. Two people planning together share one login.

Operational

****Payments are live; the product is not in any app store.**** It is a web app installed to the home screen. A user who searches the store will not find it.

****AI cost is measured per plan, not capped per user.**** The weekly analytics call has a token budget; the planner does not. A user regenerating a plan thirty times in a day costs thirty calls.

****Trial requires a card.**** Seven days, card up front, no free tier. This is a conversion decision and it is written here so support does not promise otherwise.

Found by running it

****Recipes in one language against a fridge in another matched zero ingredients.**** The matcher compared display names, not catalog ids. Fixed by matching on id; a cross-language audit now runs weekly.

****Swapping a dish rebuilt the whole plan.**** Slow and heavy for a one-slot change. Replaced with a single-slot swap; the rest of the week is left as it was.

Known rough edges

- Expiry dates are entered by hand; nothing reads a label.
- The shopping list does not know store layouts or prices.

What to notice: the first two entries are features a visitor might assume exist. Writing "no" here is what keeps the landing page honest. The "Operational" section includes a pricing decision, because support will be asked about it and the file is where the answer lives.

06 /

Worked example 3: a RAG support bot

A support agent that answers only from your documents, quotes the sentence it used, and hands the question to a person when the documents do not cover it.

By design

****One corpus, one audience.**** Every question is answered from every active document. A corpus that mixes public help pages with internal notes will quote the internal ones. Per-user permissions are a filter on the search; the field is not there.

****The "send to a person" rule is keywords.**** A regular expression decides which questions never reach the model: someone's own money, account or legal position. Keywords are blunt. It errs towards a person, which is the safe direction.

****Vector search only.**** No keyword index. Embeddings are poor at exact tokens: error codes, SKUs, surnames. A corpus full of those wants hybrid search, which is a second index and a merge step.

****One step of memory.**** The session remembers the last question and the documents the last answer stood on. "And monthly?" works. Five turns deep does not.

Operational

****The similarity threshold is corpus-specific.**** It was measured against this corpus and this embedding model. Yours will sit somewhere else. Measure before trusting.

****A threshold cannot tell "absent" from "adjacent".**** A question about a topic the corpus never mentions can score higher than questions that have answers. No number separates those two. The citation check does, because there is no sentence to quote.

****The vector store runs without authentication.**** Normal inside one container network, not on a public address. The compose file publishes the port for convenience; change that first.

Found by running it

****A document could hold two versions of itself.**** On re-index, unchanged chunks kept the old version stamp. Deleted text stayed answerable. A document is now unchanged only when all chunks agree. There is a test for it.

****Finding the right page is not finding the answer.**** A billing question matched the first half of the billing page while the answer was in the second. The fix pulls the rest of the best document into the context whether the search asked for it or not.

****A heuristic that was tried and reverted.**** Turning a short refusal into a clarifying question. Helped on vague questions, hurt on short specific ones. Word count cannot tell vague from absent. The reasoning stays in the code.

Known rough edges

- Scanned PDFs are skipped and said to be skipped.
- Cost in the run row is characters divided by four. Right order of magnitude, not a bill.

What to notice: the two "Operational" entries about thresholds are the most valuable lines in the file, and they are measurements, not opinions. The reverted heuristic is kept so it is not tried twice.

07 /

Checklist before shipping

Run this against the file, not against the code.

- ☐ The file exists at the repo root and the README links to it within the first screen.
- ☐ Every entry under "By design" can finish the sentence "because".
- ☐ Every entry under "Operational" names the host condition that causes it.
- ☐ Every entry under "Found by running it" has a date or a test name.

- ☐ Every bold lead is a noun phrase, not a sentence starting with "unfortunately" or "currently".
- ☐ Every limit says which way it fails: safe, loud, or silent.
- ☐ Every "we could fix this by" has a cost next to it: a day, a second index, a conversation.
- ☐ Numbers come from your own runs. Anything else says "not measured".
- ☐ No entry is longer than one paragraph.
- ☐ Nothing in the README contradicts anything here. Read both in one sitting.
- ☐ At least one entry is embarrassing. If none is, you have not run it enough.

The last item is the real test. A limitations file with nothing embarrassing in it was written from the spec, not from the logs.

END OF SHEET · DIMAK.APP/VAULT