Ingest a Document

Admin Upload or link material, extract its text (with optical character recognition, OCR, if needed), and embed it: three stages, three visible statuses.

Screens Documents `documents.html` · Upload `upload-dialog.html` · Embeddings `embeddings-setup.html`

Writes the document `document` · the queue `embedding_job` · the vectors `embedding`

Key fields where it came from `document.source` · the text snapshot `extracted_text_path` · is the text usable? `extraction_status` · file type `mime` · job state `embedding_job.status · attempts · error` · the chunk `embedding.chunk_text · vector`

1 · Given / When / Then

Human wording first; the exact table/field in `mono`.

GIVEN	The signed-in person is an **admin**, and the embedding rules are configured in `embedding_ingest_registry`.
WHEN	On the **Documents Screen** they add material either by uploading a file or by pointing at a web address. Both land in the same single `document.source` field, because there is deliberately no separate “file” and “URL” column to get out of sync.
THEN	A `document` row appears immediately with `extraction_status = pending`. In the background the server extracts the text, running OCR if the file is a scan, writes the text to `extracted_text_path`, and flips the status to `ready`. An `embedding_job` is then queued. When it finishes, the document's chunks exist as `embedding` rows with vectors, and the document can be attached to a run as context. **writes** `document`, `embedding_job`, `embedding`

Variations and exceptions

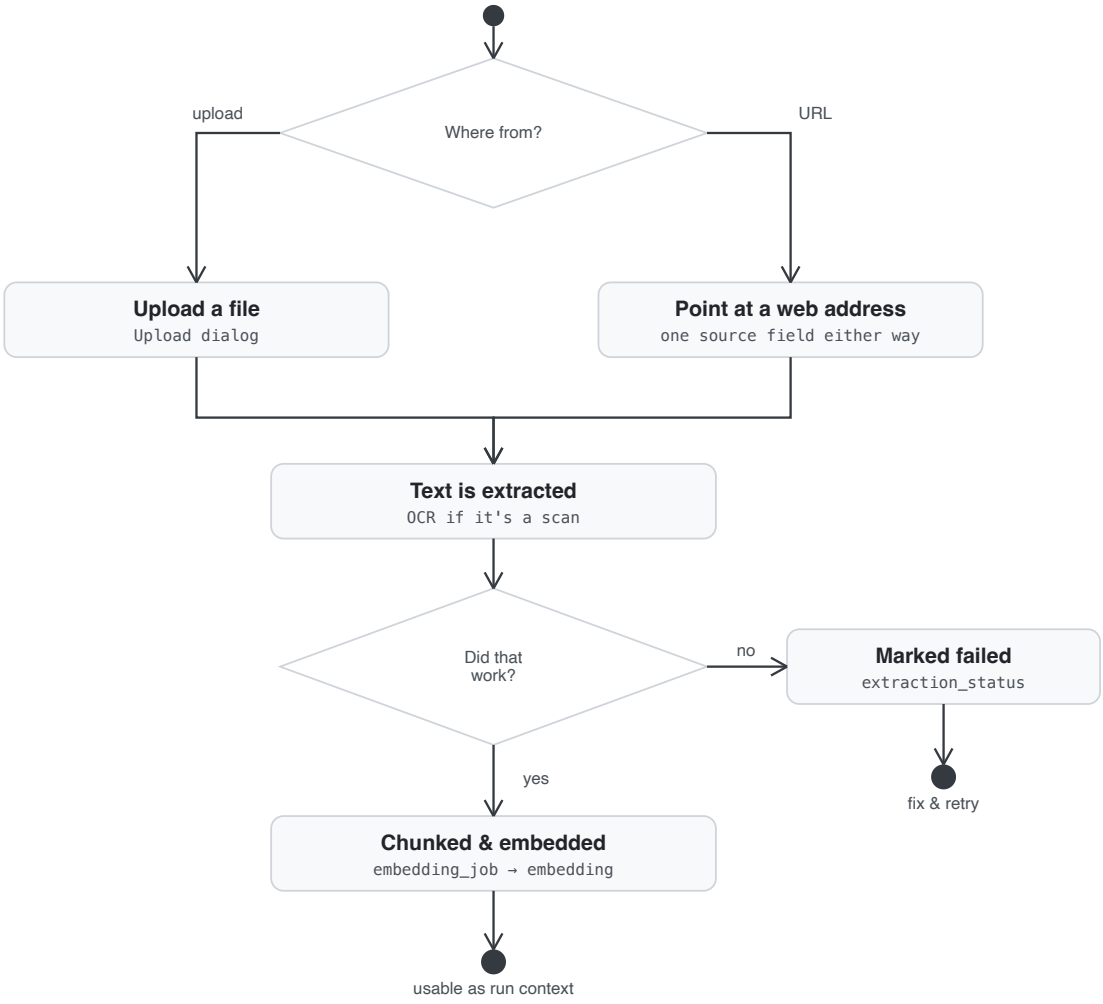
IF…	THEN…
Extraction fails — a corrupt file, a dead link, an unreadable scan	`extraction_status = failed`; the document is listed but not attachable, and the admin can fix the source and retry.
The source is replaced	Status goes to `stale` and a fresh extraction is queued. The old text stays live and searchable the entire time, and the new text becomes visible only at the atomic swap; if the re-extraction fails, the old text is still served.
A `running` or `paused` run attaches the document as context	The swap is deferred until the run finishes, so a run's retrieval corpus never shifts underneath it mid-study.
The embedding job fails	`embedding_job.status = failed` with `attempts` and `error` recorded — visibly stuck rather than silently missing.
Someone tries to attach a `pending`, `stale` or `failed` document as context	Refused: only `ready` documents attach.

IF…	THEN…
They try to delete a document an experiment still attaches	Refused (`RESTRICT`). Retiring it (`retired_at`) hides it from new studies while it stays in place for the experiments referencing it; once detached everywhere it can be decommissioned, which also removes its `embedding` rows.

Acceptance test — *Given* an admin and an active ingest registry, *When* they upload a readable PDF, *Then* a `document` row moves `pending → ready` with a non-null `extracted_text_path`, an `embedding_job` reaches `done`, and at least one `embedding` row exists; *and* an unreadable file ends at `failed` without ever nulling the path.

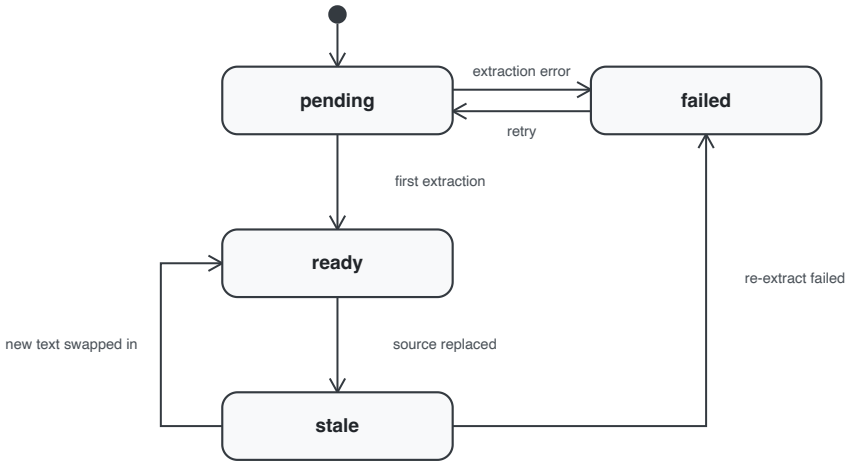
2 · The Journey, Screen by Screen

Two ways in, one pipeline out.



3 · Is the Text Usable?

The document's `extraction_status`, and why the path never goes null.

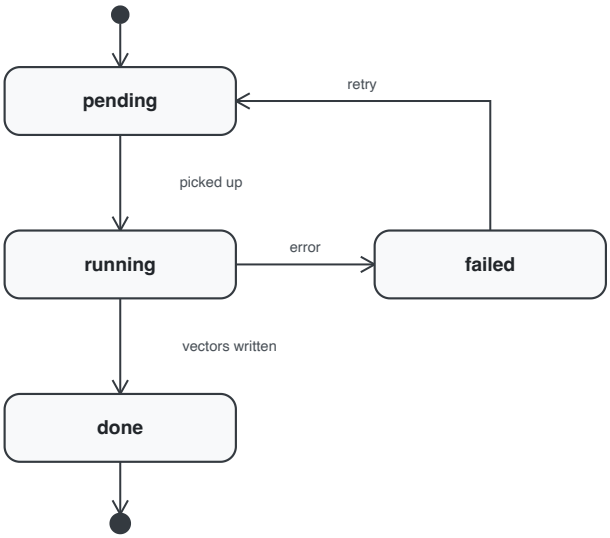


The cached text is READABLE in every state except pending — pending is the only state with nothing to read. A re-extraction is written to a STAGING file and swapped in atomically, so retrieval keeps serving the existing text for the whole time the new extraction runs, and keeps serving it even if that extraction fails. That is why stale never goes back to pending: a document that has once extracted is never empty again. While an active run uses the document the swap is DEFERRED (run-fidelity lock), so a live run's retrieval cannot shift under it.

Shown on the **Documents Screen** (`documents.html`) as each row's status badge. `stale` is what makes a replacement non-disruptive: it is a readable state, not an outage.

4 · Is It Embedded Yet?

The queue's `embedding_job.status`.

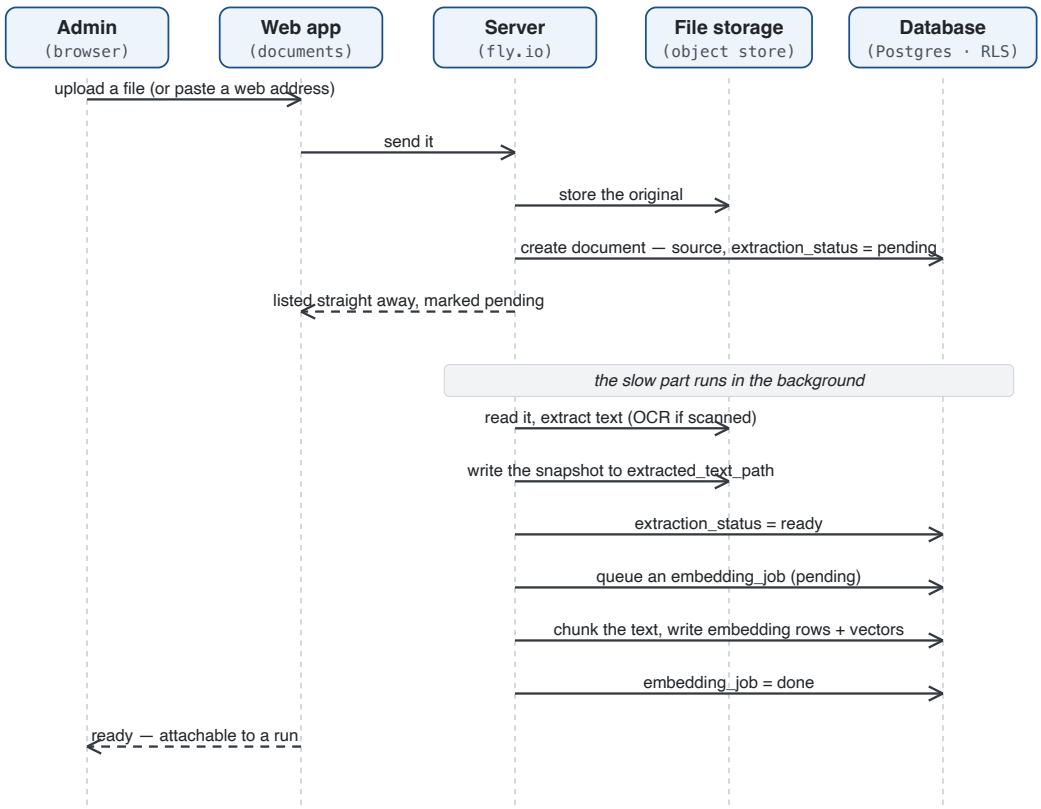


One job per source row. attempts counts the tries and error keeps the last message, so a stuck job is visible rather than silent. Re-embedding the same row replaces its embedding rows — vectors are derived data and can always be rebuilt from the source.

Watched on the **Embeddings Screen** (`embeddings-setup.html`). A failed job is a visible state with a retry path, never a silent gap.

5 · Behind the Scenes

The fast part is immediate; the slow part is queued.



The admin gets a row back at once and watches the statuses advance; nothing blocks on extraction or embedding.

6 · In Plain English

Documents are the material the AI is supposed to answer from. Getting them in has three stages, and each can fail on its own, which is exactly why each has its own visible status.

First the file arrives, by upload or from a web address. Either way it is recorded the same way, in one field, so there is never a question of which of two columns is the real source.

Then the text is pulled out, with OCR if it is a scan. There is a subtle but important design choice here: the extracted-text file keeps the same path forever and is never blanked out. When you replace a document, a fresh extraction is written alongside and swapped in at the last moment. The result is that the old text stays searchable the entire time the new one is being prepared, instead of the document dropping out of the system while it reprocesses.

Finally the text is cut into chunks and turned into vectors, so the AI can retrieve the relevant passages rather than being handed the whole document. That work is queued, and if it fails you can see that it failed, along with how many times it tried and what went wrong.

Fits the schema cleanly. The merged `document.source` and the never-nulled `extracted_text_path` and `extraction_status` pairing are exactly what these two state machines need.