MAESTRO · IMPLEMENTATION PLAN

Crate — library scan and search v1

A producer points Crate at a sample folder and gets a searchable library: every audio file inventoried once, annotated with its detected BPM and musical key, and queryable offline in under 30ms.

crate Crate v1 maestro/cello-4 15 July 2026

DRAFT for review. Scoped to the scanner and the search index; the audition player is untouched.

Crate v1 · crate · maestro/cello-4

Crate — library scan and search v1

Context & goal

INTENDED OUTCOME

A producer points Crate at a sample folder and gets a searchable library: every audio file inventoried once, annotated with its detected BPM and musical key, and queryable offline in under 30ms. The index persists across restarts and survives the folder being moved or renamed, so a rescan is incremental rather than a cold start.

WHY

Search today re-walks the disk on every keystroke. On the 60k-file library we test against (`fixtures/library-60k/`), `src/ui/search.svelte` blocks the main thread for 4–9 seconds per query, which is why the search box was quietly hidden behind a feature flag in `src/ui/flags.ts:12`. The walk is not the slow part — re-reading and re-decoding every file's header on every query is. Nothing is cached between queries.

NON-GOALS

This plan does not touch the audition player (`src/player/`), does not add cloud sync or any network calls, and does not attempt automatic genre or instrument classification. Waveform thumbnails are explicitly deferred — they need a decode pass this plan does not schedule.

CONSTRAINT

Crate is offline-first and ships as a single binary. No server, no daemon, no background indexer process. Everything here runs in-process and must degrade to "search still works, just less annotated" if a seam fails.

Current state

Grounded in the files as they stand on `main`:

- `src/scan/walk.ts` — an async generator that yields absolute paths under a root, filtered by extension (`.wav`, `.aiff`, `.flac`, `.mp3`). Correct and reasonably fast (~2.1s over 60k files, warm cache). It has no caller other than the UI.
- `src/library/store.ts` — a `Map<string, SampleMeta>` rebuilt from scratch on every mount. Never written to disk. `SampleMeta` today is `{ path, name, size }` — no BPM, no key.
- `src/ui/search.svelte` — calls `walk()` directly inside a `$derived`, then `String.includes` over filenames. This is the 4–9s block.
- `src/audio/decode.ts` — wraps `symphonia` for the player. Exposes `decodeFull(path)` only; there is no header-only path, so anything that wants a sample rate today decodes the whole file.
- `src/db/` — does not exist. There is no persistence layer of any kind yet.

ASSUMPTION

`walk.ts` is sound and stays. This plan treats it as the inventory source and builds around it rather than rewriting it.

Work breakdown by seam

Ordered by dependency. Each phase lands on its own and leaves the app working.

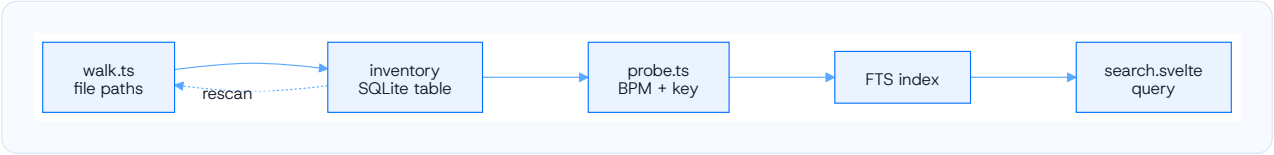

Figure 1: Scan → annotate → index → query.

Phase	Seam	What lands
1	Filesystem boundary	Inventory persists; a rescan is incremental
2	Decoder boundary	BPM and key read from headers
3	Query API surface	Search answers from the index, not the disk
4	UI data source	Flag comes off; the search box is live
5	Scheduling (opt-in)	Annotation moves off the critical path

Phase 1 — Persist the inventory

Seam: the filesystem boundary. The first thing that must stop being recomputed.

- **What changes.** Introduce `src/db/index.ts` (SQLite via `rusqlite`, file at `$DATA_DIR/crate.db`) and a `samples` table keyed by path with `size` and `mtime`. `walk()` output is upserted; rows whose path vanished are deleted. Rescan compares `(size, mtime)` and skips unchanged files.

```sql
-- src/db/migrations/001_samples.sql
CREATE TABLE samples (
  path  TEXT PRIMARY KEY,      -- absolute; re-pathed on move, never duplicated
  size  INTEGER NOT NULL,
  mtime INTEGER NOT NULL       -- with size, the whole change-detection key
) STRICT;
```

- **Files.** `src/db/index.ts` (new), `src/db/migrations/001_samples.sql` (new), `src/library/store.ts` (read from the table instead of rebuilding), `src/scan/walk.ts` (unchanged).
- **Verification.** `cargo test scan::incremental` — scan `fixtures/library-60k/`, touch one file, rescan, assert exactly one row updated. Cold scan under 5s, warm rescan under 300ms.
- **Review gate.** Does a moved root re-import everything? A skeptical reviewer should move `fixtures/library-60k/` to a sibling path, rescan, and confirm the rows are re-pathed rather than duplicated. If paths are the only key, this fails — that is the check.

Phase 2 — Header-only probe for BPM and key

Seam: the decoder boundary (`src/audio/decode.ts`). Independently verifiable against fixtures with known answers.

- **What changes.** Add `probeHeader(path) -> { sampleRate, frames, bpm?, key? }` alongside the existing `decodeFull`. BPM comes from the file's own metadata tag where present (most library packs write it), falling back to onset-interval estimation over the first 30s. Key is read from tags only — no estimation. Both fields are nullable and the UI must render without them.
- **Files.** `src/audio/probe.ts` (new), `src/audio/decode.ts` (extract the shared demuxer setup), `src/db/migrations/002_annotations.sql` (new columns), `src/scan/annotate.ts` (new — drives probe over unannotated rows).
- **Verification.** `cargo test probe::known_bpm` against `fixtures/annotated/` (24 files with hand-checked BPM/key). Assert tag-derived BPM is exact; assert estimated BPM lands within ±2 of the fixture value on the 8 untagged files.

- **Review gate.** Confirm a corrupt or truncated file yields `None` rather than panicking or poisoning the row — drop `fixtures/corrupt/truncated.wav` into the root and rescan. Annotation must be resumable: kill the process mid-annotate, restart, and confirm it continues rather than restarting.

Phase 3 — The query API

Seam: the search API surface. This is where the 4–9s goes away.

- **What changes.** SQLite FTS5 virtual table over filename and tags, plus indexed range predicates on `bpm` and `key`. Expose one function — `query(q: Query) -> Vec<SampleMeta>` — as the only supported read path into the library.
- **Files.** `src/library/query.ts` (new), `src/db/migrations/003_fts.sql` (new), `src/library/store.ts` (delegate to `query`).

```ts
// src/library/query.ts — the whole read surface. The UI gets nothing else.
export interface Query {
  text?: string; // FTS match over filename + tags
  bpm?: { min: number; max: number };
  key?: string; // exact, e.g. "Fmin"
  limit?: number; // default 200
}

export function query(q: Query): SampleMeta[];
```

- **Verification.** `cargo bench query::p95` over the 60k fixture: p95 under 30ms for a text query, under 50ms with a BPM range. Assert results are stable under repeated identical queries.
- **Review gate.** Check that `query` is genuinely the only read path — `rg "from samples" src/` should return `src/library/query.ts` and nothing else. If the UI has kept a side door into the table, this phase has not landed.

Phase 4 — Wire the UI to the index

Seam: the UI data source. Mechanical; no logic moves.

- **What changes.** `src/ui/search.svelte` calls `query()` instead of `walk()`. Remove the feature flag in `src/ui/flags.ts`. Add BPM/key filter controls and render both columns, blank when null.
- **Files.** `src/ui/search.svelte`, `src/ui/flags.ts`, `src/ui/SampleRow.svelte`.
- **Verification.** Playwright: type into the search box on the 60k fixture, assert first paint of results under 100ms and no main-thread block over 50ms.
- **Review gate.** Confirm the UI renders a library that has been scanned but not yet annotated — every BPM and key cell blank, no errors, search still working. That is the failure mode this whole plan has to survive.

Phase 5 — Move annotation off the critical path (opt-in, later)

Deliberately last and separable. Phases 1–4 annotate synchronously during scan, which is slower but simple and correct. If the cold scan proves too slow in practice, annotation becomes a worker pool that fills rows in behind a usable index. Do not start this until Phase 4 has real usage.

Option	Cold scan (60k)	Complexity	Verdict
Synchronous annotate (Phases 1–4)	~11 min est.	Low — one pass, no coordination	Ship this first
Worker pool, index-first (Phase 5)	~90s est., search live at ~5s	Medium — resumability, backpressure	Only if measured pain
Skip BPM estimation, tags only	~40s est.	Lowest	Fallback if estimation is a tarpit

Adversarial review

The assumption that sinks this if wrong. That BPM lives in the file's tags for most of a real library. The `fixtures/annotated/` set is 24 files curated by us and 16 of them are tagged — that is not evidence

about a producer's actual drive. If real libraries are mostly untagged, every scan falls through to onset estimation, cold scan goes from ~11 minutes to hours, and Phase 5 stops being optional. Validate this before starting Phase 2: run a tag-presence count over a real library and get an actual percentage. The whole shape of this plan hangs on that number.

The riskiest seam is Phase 2. Phases 1, 3 and 4 are well-understood plumbing over a schema; Phase 2 is signal processing with no ground truth beyond fixtures we wrote ourselves. "Within ±2 BPM" is a number chosen to be passable, not one derived from what a producer would call correct — and a sample that is 87 BPM but tagged 174 is not obviously wrong to a test. Expect this phase to take longer than the other four combined.

Where this plan is least sure. The `(size, mtime)` change-detection key in Phase 1. It is cheap and it is what everyone uses, but it misses in-place edits that preserve both, and it false-positives across filesystems that round mtime differently (which is every sync tool). Content hashing is correct and too slow to do on every rescan. This plan takes the cheap option knowingly; if rescans start missing edits, that is the first place to look.

> **RISK**
> Phase 3's FTS5 dependency assumes the bundled SQLite is compiled with it. If the shipped build is not, this is not a small fix — it is either a vendored SQLite or a hand-rolled index, and it changes the single-binary story in `Constraint` above. Check `PRAGMA compile_options` on the actual release artifact before Phase 3, not on a dev machine.

Counter-argument to the whole plan. The problem might be entirely in `src/ui/search.svelte` re-walking on every keystroke, not in the absence of an index. A 20-line debounce plus an in-memory cache could take 4–9s down to something acceptable and cost nothing. That would not deliver BPM or key search — which is the actual product goal — but if the goal were only "search feels fast", this plan is roughly four phases of over-engineering. It is worth being explicit that the index is justified by the annotation requirement, not by the latency number.

Rollout & verification

Land Phase 1 → 4 in order, each behind its own PR, each independently revertable. The progress signal is one number: p95 search latency on `fixtures/library-60k/`, recorded per phase.

Phase	p95 search	Library usable?
Today (flagged off)	4–9s	No — flag hides it
Phase 1	4–9s	No change — inventory only
Phase 3	under 30ms	Yes, unannotated
Phase 4	under 30ms	Yes, flag removed

Phase 4 is the point of no return — it deletes the flag and the old path. Everything before it is additive and can sit on `main` unused. If Phase 2 proves to be the tarpit this plan expects, land 1, 3 and 4 with null annotations and ship a working fast search with no BPM column; that is a coherent product on its own.

Open questions

1. **Where does `$DATA_DIR` point on each platform, and is it backed up?** A 60k-row index is disposable, but a rescan is 11 minutes. If it lands somewhere users sync, the mtime false-positive problem in the adversarial review gets worse. Needs a decision before Phase 1 writes the file.
2. **What is the actual tag-presence rate in a real library?** Blocking for Phase 2 — see the adversarial review. This is a measurement, not a decision, but nobody has taken it.

3. **Is `key` a free-text tag or an enum?** Libraries write `Fmin`, `F minor`, `Fm`, and `5A` (Camelot). Exact-match search across those is meaningless without normalisation, and normalisation is a scope decision this plan does not make.