

Love it. Here's a clean, **admin-first injection plan** with strong fallbacks: paste JSON by hand, full tag catalog picker, and "auto → curate" flows — all living in one admin page.

1) Admin ingestion modes (all supported on the same page)

1. Auto mode (OCR → LLM)

- Upload asset(s) → OCR → LLM proposes JSON.
- Shows a **diff/review panel** with all fields + proposed tags & categories.

2. Direct JSON mode (manual paste)

- Admin clicks "**Paste JSON**" → modal text area → paste LLM output from ChatGPT/wherever.
- Client validates schema + canonicalizes topics/tags → renders same review UI as Auto mode.

3. Minimal form mode (no JSON at all)

- For quick entries/edge cases: admin fills Subject, Paper, Topics, Skills, etc.
- As they type/select, the **Full Tag Catalog** (see §3) suggests completions.

All three converge to the **same validator + review** step before publishing.

2) Data contracts (TypeScript types)

```
// Canonical taxonomies
type SubjectId = "math-aa-h1" | "math-aa-s1" | "physics-h1" |
"physics-s1" | string;
type Paper = "Paper 1" | "Paper 2" | "Paper 3" | "Unknown";

interface TagCanonical {
```

```

    id: string;           // slug, unique
    label: string;        // display
    synonyms?: string[];  // for mapping pasted/LLM terms
    kind: "topic" | "skill" | "keyword";
    subjectId?: SubjectId; // optional scoping
    parentId?: string;     // build trees (e.g., Probability >
Counting)
    active: boolean;
}

interface IngestJSON {
    subject: string; // free text that maps to SubjectId
    paper: Paper;
    year_session?: string;
    topics: string[];
    skills?: string[];
    estimated_difficulty?: number; // 0..1
    marks?: number;
    question_type?: "mcq" | "short-response" | "structured" | "proof" |
"other";
    has_diagram?: boolean;
    time_recommendation_minutes?: number;
    is_calculator_allowed?: boolean;
    likelihood_to_reappear?: number;
    tags?: string[];
    quality_flags?: {
        low_ocr_confidence?: boolean;
        incomplete_capture?: boolean;
        needs_human_check?: boolean;
    };
}

interface NormalizedIngest {
    subjectId: SubjectId;
    paper: Paper;
    topicIds: string[]; // mapped from topics[]
    skillIds: string[]; // mapped from skills[]
    tagIds: string[];   // mapped from tags[]
}

```

```
meta: Omit<IngestJSON, "subject" | "topics" | "skills" | "tags">;
warnings: string[]; // e.g., "Unmapped tag: arrangements"
}
```

3) Full Tag Catalog (the “source of truth”)

- Firestore collection: `tag_catalog/{tagId}` with `TagCanonical`.
- Covers **topics**, **skills**, and **keywords**; organized in a tree for topics.
- Includes **synonyms** list to auto-map LLM/manual text to canonical tags.
- Admins manage this in a small **Taxonomy Manager** (add/merge/retire tags, edit synonyms, reorder tree).

Why:

- Guarantees consistency; prevents “Permutation” vs “Permutations” drift.
 - Makes manual curation fast (type-ahead + multi-select).
-

4) Mapping & validation pipeline

1. **Parse:** Ensure valid JSON; reject non-JSON early.
2. **Normalize:**
 - Subject string → `subjectId` (exact match or synonym table).
 - For each string in `topics/skills/tags`:
 - lowercase & trim → lookup in `tag_catalog` by `label` or `synonyms`.
 - If not found: create a “**Proposed (unmapped)**” chip in the UI.

3. **Validate:**

- Required: `subjectId`, at least one `topicId`, `paper`.
- Bounds: `marks >= 0`, `0 ≤ difficulty ≤ 1`, etc.
- Cross-checks: topic belongs to subject; paper allowed for subject.

4. **Warnings:**

- Show unmapped strings, out-of-range values, subject/topic mismatch.

UI behavior

- Validation chips (green = mapped; yellow = suggested; red = unmapped).
 - Clicking a red chip opens **Full Tag Catalog** drawer to pick the correct tag or add a quick synonym.
-

5) Admin review UI (one page, left→right flow)

Left column:

- Asset preview (image/PDF) with zoom & pages.
- OCR text (collapsible).
- If “Paste JSON” mode: text area + **Validate** button.

Center column:

- **Structured form** prefilled from normalized JSON:
 - Subject (select)

- Paper (select)
- Topics (multi-select from catalog)
- Skills (multi-select)
- Keywords/Tags (multi-select)
- Marks, Difficulty slider, Time, Calc allowed, Type, Year/Session
- “Unmapped” bucket at top with pills you can drag into fields or map via catalog.

Right column:

- **Full Tag Catalog** tree + search (filter by kind & subject).
- **Suggested tags** from auto-detection (checkbox list).
- **Change log** (who edited what).
- Buttons: **Approve & Publish** / **Save Draft** / **Reject**.

Keyboard candy

- ⌘+Enter Approve; Shift+Enter Save Draft; Esc Cancel; ↑↓ to navigate suggestions.

6) Workflows

A) Auto → Curate (happy path)

- Upload → OCR → Auto JSON → Normalize → UI suggests Topics/Skills/Keywords → Admin ticks desired → Approve.

B) Paste JSON (fallback)

- Click **Paste JSON** → paste from ChatGPT → Validate → Normalize → Same curation UI → Approve.

C) Manual form (edge)

- Fill subject/paper → pick topics/skills from catalog → Approve.

D) Late tag edits & bulk retag

- Open question → **Edit tags** → use catalog to add/remove.
- Bulk: select N questions → **Bulk actions** → **Add/Remove tag**.

7) Publishing & versioning

- On Approve: create `questions/{qid}`; copy assets to `questions/...`; write `version: 1`.
- Later edits create `version: n+1` and append to `audit/{qid}/history/{versionId}` with diff (who/when/what).
- Keep **soft delete**: `isArchived: true` instead of deleting.

Audit example entry

```
{
  "by": "adminUId123",
  "at": "2025-11-08T10:22:00Z",
  "changes": {
    "topicIds": { "add": ["prob-counting"], "remove":
["prob-uncertain"] },
    "skillIds": { "add": ["factorials"] }
  }
}
```

8) API/Functions sketch

- `POST /admin/validate-ingest` → returns `NormalizedIngest` + warnings.
- `POST /admin/publish` → writes `questions/{qid}`, moves files, sets version, audit.
- `POST /admin/map-synonym` → add synonym to `tag_catalog/{tagId}.synonyms`.
- `GET /admin/tag-catalog?subjectId=&kind=` → for the right-side drawer.
- `POST /admin/bulk-retag` → add/remove tagIds on a set of qids.
- `POST /admin/ingest/paste-json` → server-side schema validation (optional; client does it too).

Keep these behind admin auth; also run server-side canonicalization to avoid trusting the client.

9) Tag suggestion engines (layered)

- **LLM suggestions** from OCR text.
- **Embedding nearest-neighbors** (pgvector/Pinecone) → suggest related topics/skills seen on similar questions.
- **Rule hints**: if “!” or “factorial” present → suggest Counting/Permutations.
- **Co-occurrence mining**: learned over time.

Admin sees them as a checklist: **Auto-detected (checked by default)** and can uncheck.

10) Error handling & guardrails

- **JSON schema validator** (AJV) with friendly errors.
 - **Unknown subject/topic** → don't block; just flag unmapped until admin resolves.
 - **Hard stop** if no valid `subjectId` OR no `topicIds`.
 - **Retry queue** for failed OCR/LLM; manual paste is always available.
-

11) Security & roles

- Roles via **custom claims**: `admin`, `editor` (can curate/publish), `taxonomist` (can edit catalog).
 - Firestore rules: write access only for admins; read access public for published questions (your app still checks entitlements).
-

12) Quick UI copy (buttons & labels)

- **"Auto-detect tags"** (runs again if admin tweaks OCR).
 - **"Paste JSON"** (modal).
 - **"Full Tag Catalog"** (drawer, searchable).
 - **"Apply selected suggestions"** (adds checked tags).
 - **"Approve & Publish"** / **"Save as Draft"** / **"Reject Ingest"**.
-

13) Why this covers your concerns

- **Admin controls quality** (no student uploads).
- **Fast throughput** thanks to auto-suggest + catalog picker.
- **Offline fallback**: paste JSON from ChatGPT if OCR/LLM is down.
- **Consistency** via canonical catalog & synonym mapping.
- **Scales** with bulk retagging and versioned edits.

If you want, I can draft:

- The **AJV JSON schema** for `IngestJSON`.
- A **React component** for the Tag Catalog drawer (tree + search + chips).
- A **server normalizer** that maps strings → canonical `tagIds` and returns `warnings[]`.