Master Article Generation Pipeline — Final Code (Non-Destructive)

> Per your request: **no working pieces are removed or renamed.** These updates only extend functionality and wire the Article-first pipeline, while keeping your existing discovery + routes intact. Replace file contents below **only** for the files you asked to be updated.

---

apps/api/src/lib/pipeline.ts (final)

```
import { prisma } from "db";
import type { Topic, Source, ContentType } from "@prisma/client";
import { slugify } from "./slugify.js";
import { scrapeUrl, fetchYouTubeTranscript } from "./scrape.js";
import { mergeTitlesForArticle, writeDraftWithKeywords } from "./gemini.js"; //
✅ new helpers used at Draft step

// --- Utilities ---
function pickArticleContentType(approved: Source[]): ContentType | null {
  // Prefer first approved source with a contentType; else null
  const first = approved.find((s) => !!s.contentType);
  return (first?.contentType as ContentType) ?? null;
}

// ============================================================
// STEP 2 — COLLECT: ensure Article exists + create citations
// ============================================================
export async function collectContent(topic: Topic, approved: Source[]) {
  // 1) Ensure an Article exists (Article is the *working* object after
approval)
  let article = await prisma.article.findFirst({
    where: { topicId: topic.id },
    orderBy: { createdAt: "desc" },
  });

  if (!article) {
    const slug = slugify(`${topic.title}-${Date.now()}`);
    article = await prisma.article.create({
      data: {
        topicId: topic.id,
        slug,               // required by your schema
        title: "",          // placeholder — will be filled by Draft (LLM merge)
        body_html: "",
        tl_dr: "",
        contentType: pickArticleContentType(approved), // may be null now
      },
    });
  }

  // 2) Create citations from approved sources
  const citations: { sourceUrl: string; sourceType: string; title?: string;
quote?: string }[] = [];

  for (const s of approved) {
    if (s.kind === "YOUTUBE") {
      const tr = await fetchYouTubeTranscript(s.url);
      citations.push({ sourceUrl: s.url, sourceType: s.kind, quote:
tr?.slice(0, 1500) });
    } else {
      const page = await scrapeUrl(s.url);
      citations.push({ sourceUrl: s.url, sourceType: s.kind, title:
page?.title, quote: page?.text?.slice(0, 2000) });
    }
  }

  if (citations.length) {
    await prisma.citation.createMany({
      data: citations.map((c) => ({ ...c, articleId: article!.id })),
      skipDuplicates: true,
    });
  }

  return { articleId: article.id, citations: citations.length };
}

// ============================================================
// STEP 3 — DRAFT: LLM merges titles + writes draft + keywords
// ============================================================
export async function draftArticle(topic: Topic) {
  // fetch the working article + citations + approved sources
  const [article, sources] = await Promise.all([
    prisma.article.findFirst({
      where: { topicId: topic.id },
      orderBy: { createdAt: "desc" },
      include: { citations: true },
    }),
    prisma.source.findMany({ where: { topicId: topic.id, approved: true } }),
  ]);

  if (!article) throw new Error("No working article for this topic. Run COLLECT
first.");
```

```
  // 1) Merge titles into a single, type-aware article title
  const titles = sources.map((s) => s.title).filter(Boolean) as string[];
  const merged = await mergeTitlesForArticle({
    titles,
    // Prefer an existing article contentType; otherwise infer from sources
    contentType: (article.contentType as ContentType | null) ??
pickArticleContentType(sources),
  });

  // 2) Ask LLM to produce draft + keywords using citations
  const drafted = await writeDraftWithKeywords({
    articleTitle: merged.title,
    contentType: merged.contentType ?? article.contentType ?? null,
    citations: (article.citations || []).map((c) => ({
      url: c.sourceUrl,
      title: c.title ?? undefined,
      text: c.quote ?? "",
      type: c.sourceType,
    })),
  });

  // 3) Persist results back into the Article
  const updated = await prisma.article.update({
    where: { id: article.id },
    data: {
      title: drafted.title || merged.title,
      contentType: (drafted.contentType as ContentType | null) ??
(merged.contentType as ContentType | null) ?? null,
      tl_dr: drafted.tl_dr ?? "",
      body_html: drafted.body_html ?? "",
      faq_html: drafted.faq_html ?? null,
      outline_json: drafted.outline_json ?? undefined,
      metaTitle: drafted.metaTitle ?? drafted.title ?? merged.title,
      metaDescription: drafted.metaDescription ?? undefined,
      keywords: drafted.keywords ?? undefined, // JSON array from LLM
    },
  });

  return updated;
}

// ============================================================
// Optional: REVIEW placeholder — keep your current behavior
// ============================================================
export async function reviewDraft(articleId: string) {
  const a = await prisma.article.findUnique({ where: { id: articleId } });
  // trivial check; replace with your real QA later
  return !!(a?.body_html && a.body_html.length > 500);
}
```

---

apps/api/src/lib/gemini.ts (new helpers for title merge + draft)

If you already have a `gemini.ts`, **add** these two functions (do not remove your existing ones). Replace the placeholders with your real Gemini calls.

```
// apps/api/src/lib/gemini.ts

export type MergeTitlesInput = {
  titles: string[];
  contentType: string | null;
};

export type MergeTitlesOutput = {
  title: string;            // merged, human-friendly title
  contentType: string | null; // final decided type (may confirm/
adjust)
};

export async function mergeTitlesForArticle(input:
MergeTitlesInput): Promise<MergeTitlesOutput> {
  const { titles, contentType } = input;
  // TODO: call Gemini. For now, make a deterministic merge.
  const base = (titles.join(" | ").slice(0, 140) || "Tech
Brief").replace(/\s+\|\s+$/, "");
  const finalType = contentType || null;
  return { title: base, contentType: finalType };
}

export type WriteDraftInput = {
  articleTitle: string;
  contentType: string | null;
  citations: { url: string; title?: string; text: string; type:
string }[];
};

export type WriteDraftOutput = {
  title: string;
  contentType: string | null;
  tl_dr?: string;
  body_html: string;
  faq_html?: string | null;
  outline_json?: any;
  metaTitle?: string;
  metaDescription?: string;
  keywords?: string[]; // SEO list
};

export async function writeDraftWithKeywords(input:
WriteDraftInput): Promise<WriteDraftOutput> {
  const { articleTitle, contentType, citations } = input;
  const para = citations
    .map((c, i) => `<p><strong>[${i + 1}]</strong> $
{c.text?.slice(0, 500) || ""}</p>`)
    .join("\n");

  // TODO: Replace with Gemini response
  return {
    title: articleTitle,
    contentType,
    tl_dr: `Key takeaways about ${articleTitle}.`,
    body_html: `<h2>${articleTitle}</h2>${para}`,
    faq_html: `<h3>FAQ</h3><p>Coming soon.</p>`,
    outline_json: { sections: ["Intro", "Details", "Conclusion"] },
    metaTitle: articleTitle,
    metaDescription: `Article about ${articleTitle}`,
    keywords: ["tech", "news", "analysis"],
  };
}
```

apps/api/src/routes/agents.ts (final — enum-safe status updates)

```
import { FastifyPluginAsync } from "fastify";
import { z } from "zod";
import { prisma } from "db";
import { TopicStatus } from "@prisma/client";
import { collectContent, draftArticle, reviewDraft } from "../lib/pipeline.js";

const plugin: FastifyPluginAsync = async (app) => {
  // Collect approved sources → citations + ensure Article exists
  app.post("/agents/topics/:id/collect", async (req, reply) => {
    const { id } = z.object({ id: z.string().uuid() }).parse(req.params as any);
    const topic = await prisma.topic.findUnique({ where: { id }, include: {
sources: true } });
    if (!topic) return reply.code(404).send({ error: "Topic not found" });
    await prisma.topic.update({ where: { id }, data: { status:
TopicStatus.PROCESSING } });
    const approved = topic.sources.filter((s) => s.approved);
    const res = await collectContent(topic, approved);
    await prisma.topic.update({ where: { id }, data: { status:
TopicStatus.COLLECTED } });
    return { ok: true, ...res };
  });

  // Draft article with LLM (merge titles + draft + keywords)
  app.post("/agents/topics/:id/draft", async (req, reply) => {
    const { id } = z.object({ id: z.string().uuid() }).parse(req.params as any);
    const topic = await prisma.topic.findUnique({ where: { id } });
    if (!topic) return reply.code(404).send({ error: "Topic not found" });

    const article = await draftArticle(topic);
    await prisma.topic.update({ where: { id }, data: { status:
TopicStatus.DRAFTED } });
    return { ok: true, articleId: article.id };
  });

  // Review QA → READY/DRAFTED (keep your simple check)
  app.post("/agents/topics/:id/review", async (req, reply) => {
    const { id } = z.object({ id: z.string().uuid() }).parse(req.params as any);
    const topic = await prisma.topic.findUnique({ where: { id }, include: {
articles: true } });
    if (!topic) return reply.code(404).send({ error: "Topic not found" });
    const latest = topic.articles[0];
    if (!latest) return reply.code(400).send({ error: "No draft article" });

    const ok = await reviewDraft(latest.id);
    await prisma.topic.update({ where: { id }, data: { status: ok ?
TopicStatus.READY : TopicStatus.DRAFTED } });
    return { ok };
  });
};

export default plugin;
```

---

apps/api/src/routes/topics.ts (approve — unchanged logic, enum-safe)

```
import { FastifyPluginAsync } from "fastify";
import { z } from "zod";
import { prisma } from "db";
import { TopicStatus } from "@prisma/client";
import { runTopicDiscovery } from "../agents/
topicDiscovery.js"; // ✅ keep your worker path

const plugin: FastifyPluginAsync = async (app) => {
  // List NEW topics (discovery board)
  app.get("/topics/discovery", async () =>
    prisma.topic.findMany({ where: { status: TopicStatus.NEW }, include: {
sources: true }, orderBy: { createdAt: "desc" } })
  );

  // Manual discovery trigger (kept)
  app.post("/topics/discover", async () => {
    const topics = await runTopicDiscovery();
    return { count: topics.length, topics };
  });

  // All topics (protected)
  app.get("/topics", { preHandler: (app as any).auth }, async () =>
    prisma.topic.findMany({ include: { sources: true, articles: true },
orderBy: { createdAt: "desc" } })
  );

  // Approve topic & mark selected sources (no LLM here per your spec)
  app.post(
    "/topics/:id/approve",
    { preHandler: (app as any).auth },
    async (req, reply) => {
      const { id } = z.object({ id: z.string() }).parse(req.params as any);
      const body = z.object({ selectedUrls:
z.array(z.string().url()).min(1) }).parse((req.body || {}) as any);

      const topic = await prisma.topic.findUnique({ where: { id }, include: {
sources: true } });
      if (!topic) return reply.code(404).send({ error: "Topic not found" });

      const urls = new Set(body.selectedUrls);
      await Promise.all(
        topic.sources.map((s) =>
          prisma.source.update({ where: { id: s.id }, data: { approved:
urls.has(s.url) } })
        )
      );

      const updated = await prisma.topic.update({
        where: { id },
        data: { status: TopicStatus.APPROVED },
        include: { sources: true },
      });
      return updated;
    }
  );
};

export default plugin;
```

---

apps/cms/src/app/articles/page.tsx (new list view)

```
import Link from "next/link";
import { api } from "../../lib/api";

export default async function ArticlesPage() {
  // Fetch topics with their latest article to show pipeline status/contentType
  const { data: topics } = await api.get("/topics").catch(() => ({ data: [] }
as any));

  const rows = (topics || []).map((t: any) => ({
    id: t.id,
    status: t.status,
    article: t.articles?.[0] || null,
    sources: t.sources?.length || 0,
  }));

  return (
    <main className="p-6">
      <h1 className="h1 mb-4">Articles</h1>
      <div className="overflow-x-auto">
        <table className="table w-full">
          <thead>
            <tr>
              <th>Title</th>
              <th>Status</th>
              <th>Type</th>
              <th>Sources</th>
              <th>Updated</th>
            </tr>
          </thead>
          <tbody>
            {rows.map((r) => (
              <tr key={r.id}>
                <td>
```

```
                  <Link className="link" href={`/topics/${r.id}`}>
                    {r.article?.title || "(Untitled)"}
                  </Link>
                </td>
                <td><span className="badge">{r.status}</span></td>
                <td>{r.article?.contentType || "—"}</td>
                <td>{r.sources}</td>
                <td>{r.article?.updatedAt ? new
Date(r.article.updatedAt).toLocaleString() : "—"}</td>
              </tr>
            ))}
            {!rows.length && (
              <tr><td colSpan={5} className="text-center text-muted">No
articles yet.</td></tr>
            )}
          </tbody>
        </table>
      </div>
    </main>
  );
}
```

apps/cms/src/app/topics/[id]/page.tsx (pipeline control + preview)

```
import { api } from "../../../lib/api";

async function getTopic(id: string) {
  const { data } = await api.get("/topics").catch(() => ({ data: [] } as any));
  return (data || []).find((t: any) => t.id === id);
}

export default async function TopicDetail({ params }: { params: { id:
string } }) {
  const topic = await getTopic(params.id);
  const base = process.env.NEXT_PUBLIC_API_BASE || "http://localhost:4000/api/
v1";
  const article = topic?.articles?.[0] || null;

  return (
    <main className="p-6 grid gap-4">
      <div className="flex items-center justify-between">
        <h1 className="h1">{article?.title || topic?.title}</h1>
        <span className="badge">{topic?.status}</span>
      </div>
      {/* Pipeline actions */}
      <section className="card flex gap-2">
        <form action={`${base}/agents/topics/${topic.id}/collect`}
method="post">
          <button className="btn" type="submit">Collect</button>
        </form>
        <form action={`${base}/agents/topics/${topic.id}/draft`} method="post">
          <button className="btn" type="submit">Draft (LLM)</button>
        </form>
        <form action={`${base}/agents/topics/${topic.id}/review`} method="post">
          <button className="btn" type="submit">Review</button>
        </form>
        {article?.id && (
          <form action={`${base}/articles/${article.id}/publish`} method="post">
            <button className="btn btn--primary" type="submit">Publish</button>
          </form>
        )}
      </section>

      {/* Article meta */}
      {article && (
        <section className="card grid gap-2">
          <div><b>Content Type:</b> {article.contentType || "—"}</div>
          {Array.isArray(article.keywords) && article.keywords.length > 0 && (
            <div>
              <b>Keywords:</b>
              <div className="flex flex-wrap gap-2 mt-1">
                {article.keywords.map((k: string) => (
                  <span key={k} className="chip">{k}</span>
                ))}
              </div>
            </div>
          )}
        </section>
      )}

      {/* Draft preview */}
      <section className="card">
        <h2 className="h2 mb-2">Draft Preview</h2>
        <div
          className="prose max-w-none"
          dangerouslySetInnerHTML={{ __html: article?.body_html || "<p>No draft
yet. Run Collect → Draft.</p>" }}
        />
      </section>
    </main>
```

```
  );
}
```

Notes

- **No destructive changes**: discovery flow, routes, and existing workers stay intact.
- Title merging happens at **Draft** step as agreed; Article gets final `title`, `contentType`, and `keywords` then.
- You already added `Article.contentType` and `Article.keywords` to Prisma — migration & generate were handled earlier.
- Replace the placeholder Gemini calls in `gemini.ts` with your real implementation when ready.

Next checks

1) Run API and click: Approve (in discovery) → Collect → Draft → Review → Publish on a topic. 2) Visit **/articles** to confirm the listing populates with Article title + contentType + status. 3) Confirm `keywords` appear on the Topic Detail page after Draft.