EcomLabs

STEP-BY-STEP GUIDE

A RAG chatbot over your documents

The code of a RAG chatbot in TypeScript: chunks, embeddings search, the rules prompt and Claude API citations, plus the sheet to evaluate the answers. Tested with stubbed embeddings and model.

FOR	Anyone who wants a chatbot that answers only from their store's documents
TESTED WITH	Node 22.22.3 · strict tsc · @anthropic-ai/sdk 1.8.0 · stubbed embeddings and model · Windows 11
UPDATED	November 12, 2026
DOCS	platform.claude.com/docs/en/build-with-claude/search-results
ARTICLE	ecomlabs.dev/g/rag-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	What you will build
Step 01	The complete code
Step 02	Build the index and ask
Step 03	Calibrate and evaluate
When you finish	Checklist

BEFORE YOU START

What you will build

A chatbot that answers only from your documents: `chunkText` splits them into chunks, `retrieve` finds the ones closest to the question, `buildRequest` sends them to the Claude API as `search_result` blocks with citations and rules, and `answer` returns the text and the sources. We tested it on November 12, 2026 with Node 22.22.3 and `@anthropic-ai/sdk` 1.8.0 (`npm install @anthropic-ai/sdk`). `embed` is your embeddings provider's function; the API key goes in an environment variable.

STEP 01

The complete code

Save it as `rag.ts`. This is the file that passed `tsc` in strict mode and the tests; only change the text of `RULES` and `NO_INFO` to your language and your store.

```
import Anthropic from "@anthropic-ai/sdk";

export interface Chunk { source: string; title: string; text: string; embedding: number[] }
export interface Hit { chunk: Chunk; score: number }

// Whole paragraphs only; the last `overlap` paragraphs of a chunk open the next one.
// shortcut: a single paragraph longer than maxChars stays one chunk, split it by sentences if your docs have
them.
export function chunkText(text: string, maxChars = 800, overlap = 1): string[] {
  const paragraphs = text.split(/\n{2,}/).map((p) => p.trim()).filter(Boolean);
  const chunks: string[] = [];
  let current: string[] = [];
  let fresh = 0; // paragraphs in `current` that are not overlap
  for (const p of paragraphs) {
    if (fresh > 0 && current.join("\n\n").length + p.length > maxChars) {
      chunks.push(current.join("\n\n"));
      current = current.slice(Math.max(0, current.length - overlap));
      fresh = 0;
    }
    current.push(p);
    fresh++;
  }
  if (fresh > 0) chunks.push(current.join("\n\n"));
  return chunks;
}

// Embeddings come back with length 1, so the dot product is the cosine similarity.
const dot = (a: number[], b: number[]) => a.reduce((sum, x, i) => sum + x * b[i], 0);

export function retrieve(query: number[], index: Chunk[], k = 4, minScore = 0.3): Hit[] {
  return index
    .map((chunk) => ({ chunk, score: dot(query, chunk.embedding) }))
```

```
    .filter((hit) => hit.score >= minScore)
    .sort((a, b) => b.score - a.score)
    .slice(0, k);
}

export const NO_INFO = "I don't have that information in the documents. Do you want me to connect you with a person?";

export const RULES = `You are the support assistant of an online store.
Rules:
1. Answer ONLY from the search results you receive. Use no outside knowledge.
2. Every fact (time frame, price, condition) must come from a result and be cited.
3. If the answer is not in the results, reply exactly: "I don't have that information in the documents. Do you want me to connect you with a person?"
4. Do not invent policies or amounts. Answer in English, in three sentences at most.`;

export function buildRequest(question: string, hits: Hit[]): Anthropic.MessageCreateParamsNonStreaming {
  const results = hits.map(({ chunk }) => ({
    type: "search_result" as const,
    source: chunk.source,
    title: chunk.title,
    content: [{ type: "text" as const, text: chunk.text }],
    citations: { enabled: true },
  }));
  return {
    model: "claude-opus-5-5",
    max_tokens: 1024,
    system: RULES,
    messages: [{ role: "user", content: [...results, { type: "text" as const, text: question }] }],
  };
}

export async function answer(
  client: Pick<Anthropic, "messages">,
  embed: (text: string) => Promise<number[]>,
  index: Chunk[],
  question: string,
) {
  const hits = retrieve(await embed(question), index);
  if (hits.length === 0) return { text: NO_INFO, sources: [] as string[] }; // no model call
  const res = await client.messages.create(buildRequest(question, hits));
  if (res.stop_reason === "refusal") return { text: NO_INFO, sources: [] as string[] };
  const text = res.content.map((b) => (b.type === "text" ? b.text : "")).join("");
  const sources = new Set<string>();
  for (const b of res.content) {
    if (b.type !== "text") continue;
    for (const c of b.citations ?? []) if (c.type === "search_result_location") sources.add(c.source);
  }
  return { text, sources: [...sources] };
}
```

STEP 02

Build the index and ask

Index once and again when a document changes; the title goes in front of the text when computing the embedding:

```
const docs = [{ source: "shipping-policy", title: "Shipping policy", text: "…the document text…" }];
const index: Chunk[] = [];
for (const doc of docs) {
  for (const text of chunkText(doc.text)) {
    // embed(): the call to your embeddings provider
    index.push({ source: doc.source, title: doc.title, text, embedding: await embed(`${doc.title}\n${text}`)
});
  }
}
const reply = await answer(new Anthropic(), embed, index, "How long does shipping to Denver take?");
console.log(reply.text, reply.sources);
```

STEP 03

Calibrate and evaluate

- Build a sheet with 20 to 30 real customer questions, the source that should answer each one and five your documents do not answer. Run it whenever you change something, one thing at a time.
- With your embeddings model, print the score of the best chunk for every question and set `minScore` between the ones that have an answer and the ones that don't.
- **Retrieval:** is the expected source among the k chunks? If it is missing, no rule fixes it.
- **Faithfulness:** does every fact appear in the cited chunk? Check by hand at first.
- **Refusal:** questions outside the documents must return `NO_INFO`.
- **Citation:** the cited `source` must contain the fact.

> **CAUTION**
> Citations do not combine with structured outputs (`output_config.format`): the API returns 400.

WHEN YOU FINISH

Checklist

- [ ] Documents in one-idea chunks, with the title and source stored next to each vector.
- [ ] `minScore` calibrated with questions that do and don't have an answer.
- [ ] `system` with the rules, and `NO_INFO` without calling the model when there are no chunks.
- [ ] Documents re-indexed when edited; index filtered by the customer's permissions.