GUÍA PASO A PASO

Chatbot RAG con tus documentos

El código de un chatbot RAG en TypeScript: fragmentos, búsqueda por embeddings, el prompt con reglas y las citas del API de Claude, más la hoja para evaluar las respuestas. Probado con embeddings y modelo simulados.

PARA	Quien quiere un chatbot que conteste solo con los documentos de su tienda
PROBADO CON	Node 22.22.3 · tsc estricto · @anthropic-ai/sdk 1.8.0 · embeddings y modelo simulados · Windows 11
ACTUALIZADA	12 de noviembre de 2026
DOCUMENTACIÓN	platform.claude.com/docs/en/build-with-claude/search-results
ARTÍCULO	ecomlabs.dev/g/rag
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Qué vas a montar
Paso 01	El código completo
Paso 02	Armar el índice y preguntar
Paso 03	Calibrar y evaluar
Al terminar	Checklist

ANTES DE EMPEZAR

Qué vas a montar

Un chatbot que contesta solo con tus documentos: `chunkText` los parte en fragmentos, `retrieve` busca los más cercanos a la pregunta, `buildRequest` los envía al API de Claude como bloques `search_result` con citas y reglas, y `answer` devuelve el texto y las fuentes. Lo probamos el 12 de noviembre de 2026 con Node 22.22.3 y `@anthropic-ai/sdk` 1.8.0 (`npm install @anthropic-ai/sdk`). `embed` es la función de tu proveedor de embeddings; la clave del API va en una variable de entorno.

PASO 01

El código completo

Guárdalo como `rag.ts`. Es el archivo que pasó `tsc` en modo estricto y las pruebas; solo cambia el texto de `RULES` y de `NO_INFO` a tu idioma y tu tienda.

```
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 = "No tengo esa información en los documentos. ¿Quieres que te pase con una persona?";

export const RULES = `Eres el asistente de atención de una tienda en línea.
Reglas:
1. Responde SOLO con los resultados de búsqueda que recibes. No uses conocimiento externo.
2. Cada dato (plazo, precio, condición) debe salir de un resultado y citarse.
3. Si la respuesta no está en los resultados, responde exactamente: "No tengo esa información en los
documentos. ¿Quieres que te pase con una persona?"
4. No inventes políticas ni cantidades. Responde en español, en tres frases como máximo.`;

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] };
}
```

PASO 02

Armar el índice y preguntar

Indexa una vez y de nuevo al cambiar un documento; el título va delante del texto al calcular el embedding:

```
const docs = [{ source: "politica-envios", title: "Política de envíos", text: "…el texto del documento…" }];
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, "¿Cuánto tarda el envío a Monterrey?");
console.log(reply.text, reply.sources);
```

PASO 03

Calibrar y evaluar

- Arma una hoja con 20 a 30 preguntas reales de tus clientes, la fuente que debería responder cada una y cinco que tus documentos no contestan. Córrela cada vez que cambies algo, de una cosa por vez.
- Con tu modelo de embeddings, imprime la puntuación del mejor fragmento de cada pregunta y pon `minScore` entre las que sí tienen respuesta y las que no.
- **Recuperación:** ¿la fuente esperada está entre los k fragmentos? Si falta, ninguna regla lo arregla.
- **Fidelidad:** ¿cada dato aparece en el fragmento citado? Revísalo a mano al principio.
- **Rechazo:** las preguntas fuera de los documentos deben devolver `NO_INFO`.
- **Cita:** el `source` citado debe contener el dato.

> **CUIDADO**
> Las citas no se combinan con structured outputs (`output_config.format`): el API responde 400.

AL TERMINAR

Checklist

- [ ] Documentos en fragmentos de una idea, con título y fuente guardados junto a cada vector.
- [ ] `minScore` calibrado con preguntas con y sin respuesta.
- [ ] `system` con las reglas y `NO_INFO` sin llamar al modelo cuando no hay fragmentos.
- [ ] Documentos reindexados al editarse; índice filtrado por permisos del cliente.