CÓDIGO COMPLETO Y CHECKLIST

JSON válido de la IA, siempre

Structured outputs con un esquema, validación con zod y un reintento que le devuelve el error al modelo: la función en TypeScript, sus pruebas con un cliente falso y la checklist.

PARA	Proyectos en TypeScript que le piden datos estructurados a Claude
PROBADO CON	Node 22.22.3 · zod 4.6.7 y 3.25.76 · @anthropic-ai/sdk 1.4.0 · TypeScript 5.7.3 · cliente falso
ACTUALIZADA	3 de noviembre de 2026
DOCUMENTACIÓN	platform.claude.com/docs/en/build-with-claude/structured-outputs
ARTÍCULO	ecomlabs.dev/g/json
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Qué vas a montar
Paso 01	El pedido con structured outputs
Paso 02	extract.ts
Paso 03	Pruébala sin llamar a la API
Al terminar	Checklist

ANTES DE EMPEZAR

Qué vas a montar

Una función, `extractJson`, que le pide a Claude un objeto con structured outputs, lo valida con zod y, si algo falla, reintenta mandándole al modelo el error exacto, con un límite de intentos. Una negativa (`refusal`) se detiene; un corte (`max_tokens`) se repite con el doble de tokens.

- Instala en tu proyecto: `npm install @anthropic-ai/sdk zod`. Probado con `@anthropic-ai/sdk` 1.4.0 y zod 4.6.7.
- Si tu proyecto usa zod 3.25.x, cambia `from "zod"` por `from "zod/v4"`. Con `"zod"`, el helper del SDK falla.
- Node 22.22.3 ejecuta estos `.ts` sin opciones extra.

> **CUIDADO**
> La forma del pedido sigue la documentación de Anthropic consultada el 2 de noviembre de 2026. No la ejecutamos contra la API: revisa la página de structured outputs antes de llevarlo a producción.

PASO 01

El pedido con structured outputs

El esquema va en `output_config.format`, con `type: "json_schema"`. Es una función estable: no necesita encabezado beta. El parámetro anterior, `output_format`, está obsoleto.

```
{
  "model": "claude-sonnet-5-5",
  "max_tokens": 1024,
  "messages": [{ "role": "user", "content": "Extract the product from this supplier sheet: …" }],
  "output_config": {
    "format": {
      "type": "json_schema",
      "schema": {
        "type": "object",
        "properties": {
          "title": { "type": "string" },
          "price": { "type": "number" },
          "currency": { "type": "string", "enum": ["MXN", "USD"] },
          "sizes": { "type": "array", "items": { "type": "string" } },
          "inStock": { "type": "boolean" }
        },
        "required": ["title", "price", "currency", "sizes", "inStock"],
        "additionalProperties": false
      }
    }
  }
}
```

Lo que structured outputs no garantiza, según la documentación:

CASO	QUÉ PASA	QUÉ HACE EL CÓDIGO
Negativa	`stop_reason: "refusal"`, estado 200, se cobra; la salida puede no cumplir el esquema	Lanza un error, sin reintentar
Corte	`stop_reason: "max_tokens"`, salida incompleta	Repite con el doble de `max_tokens`
Mayúsculas de un `enum`	No garantizadas, sin error	Zod lo rechaza y reintenta
Restricciones no admitidas	`minimum`, `maxLength`… dan 400; el SDK las pasa a la descripción	Zod las hace cumplir

PASO 02

extract.ts

La función completa. Recibe el cliente como argumento: en producción, `new Anthropic()`; en pruebas, uno falso.

```
import type Anthropic from "@anthropic-ai/sdk";
import { zodOutputFormat } from "@anthropic-ai/sdk/helpers/zod";
import * as z from "zod";

export const Product = z.object({
  title: z.string(),
  price: z.number().positive(),
  currency: z.enum(["MXN", "USD"]),
  sizes: z.array(z.string()),
  inStock: z.boolean(),
});

type Client = {
  messages: { create(body: Anthropic.MessageCreateParamsNonStreaming): Promise<Anthropic.Message> };
};

export async function extractJson<S extends z.ZodType>(
  client: Client,
  schema: S,
  prompt: string,
  { model = "claude-sonnet-5-5", maxTokens = 1024, maxAttempts = 3 } = {},
): Promise<z.infer<S>> {
  const messages: Anthropic.MessageParam[] = [{ role: "user", content: prompt }];
  let lastError = "";

  for (let attempt = 1; attempt <= maxAttempts; attempt++) {
    const res = await client.messages.create({
      model,
      max_tokens: maxTokens,
      messages,
      output_config: { format: zodOutputFormat(schema) },
```

```
    });

    if (res.stop_reason === "refusal") {
      throw new Error("The model refused the request; retrying will not change that.");
    }
    if (res.stop_reason === "max_tokens") {
      lastError = `output cut off at max_tokens=${maxTokens}`;
      maxTokens *= 2;
      continue;
    }

    const text = res.content.flatMap((block) => (block.type === "text" ? [block.text] : [])).join("");
    try {
      const result = schema.safeParse(JSON.parse(text));
      if (result.success) return result.data;
      lastError = z.prettifyError(result.error);
    } catch (err) {
      lastError = (err as Error).message;
    }

    messages.push(
      { role: "assistant", content: text || "(empty)" },
      {
        role: "user",
        content: `Your reply does not match the JSON schema:\n${lastError}\nReply again with only the
corrected JSON object.`,
      },
    );
  }

  throw new Error(`No valid JSON after ${maxAttempts} attempts. Last error: ${lastError}`);
}
```

Así se usa con el SDK (lee la clave de `ANTHROPIC_API_KEY`; este archivo pasó el chequeo de tipos, no se ejecutó):

```
import Anthropic from "@anthropic-ai/sdk";
import { extractJson, Product } from "./extract.ts";

const client = new Anthropic(); // reads ANTHROPIC_API_KEY
const sheet = "Sudadera Volcán, algodón 100 %, tallas S a L, $899 MXN, disponible.";
const product = await extractJson(client, Product, `Extract the product from this supplier sheet:\n${sheet}`);
console.log(product.price, product.inStock);
```

PASO 03

Pruébala sin llamar a la API

Pásale a `extractJson` un objeto con `messages.create` que guarde cada pedido (`JSON.parse(JSON.stringify(body))`, como saldría por la red) y devuelva un mensaje con `content: [{ type:`

`"text", text }]` y el `stop_reason` que quieras probar. Estas son las respuestas rotas que usamos, en orden:

```
export const FAILURES: Reply[] = [
  { text: "```json\n" + VALID + "\n```" },
  { text: "Claro, aquí está el producto: " + VALID },
  { text: '{"title":"Sudadera Volcán","price":"899","currency":"mxn","sizes":["S","M","L"],"inStock":"yes"}'
},
  { text: '{"title":"Sudadera Volcán","price":899,"curr', stop_reason: "max_tokens" },
  { text: VALID },
];
```

Con `node:test` comprobamos cuatro cosas:

- Con `maxAttempts: 5`, pasa por las cinco respuestas y devuelve el objeto válido; cada reintento lleva su error y el quinto pedido sube `max_tokens` a 2048 sin la respuesta cortada.
- El pedido lleva `output_config.format` con `type: "json_schema"` y `additionalProperties: false`.
- Una negativa termina en error tras un solo pedido.
- Tres respuestas inválidas terminan en `No valid JSON after 3 attempts`.

Extracto de la salida de `node --test --test-reporter=spec extract.test.ts` (4 pruebas, 4 pasaron):

```
✓ retries each failure with its error and returns the valid object (9.5142ms)
✓ sends the schema in output_config.format (0.5039ms)
✓ stops on a refusal without retrying (0.6876ms)
✓ gives up after maxAttempts (1.6116ms)
```

AL TERMINAR

Checklist

- [ ] El esquema va en `output_config.format` con `type: "json_schema"`, no solo en el prompt.
- [ ] Cada objeto tiene `additionalProperties: false` y los campos que necesitas están en `required`.
- [ ] Tu código valida cada respuesta con zod, también con structured outputs activo.
- [ ] `refusal` detiene el proceso; `max_tokens` repite con más tokens, sin reenviar lo cortado.
- [ ] Un fallo de `JSON.parse` o de zod vuelve al modelo con el error exacto, como turno `user`.
- [ ] Hay un límite de intentos y un error claro al llegar a él.
- [ ] Con zod 3.25.x importas desde `"zod/v4"`.
- [ ] Sin datos de salud protegidos (PHI) en nombres de campo, `enum`, `const` ni `pattern`.