EcomLabs

GUÍA PASO A PASO

Validación con zod en TypeScript: el ejemplo completo

Un esquema de zod para un pedido, la función que revisa el formulario, el componente, la ruta de API de Next.js que responde 400 y un script que lo prueba todo. El mismo código que corrimos, listo para copiar.

PARA	Proyectos en TypeScript con formularios y rutas de API, en Next.js (App Router) o cualquier servidor con Request y Response
PROBADO CON	zod 4.6.7 · Node 22.22.3 · tsc 5.7.3 --strict · tsx
ACTUALIZADA	19 de noviembre de 2026
DOCUMENTACIÓN	zod.dev
ARTÍCULO	ecomlabs.dev/g/zod
AUTOR	Luis Morales · @luismoralesdev

Índice

Paso 01	El esquema
Paso 02	El formulario
Paso 03	La ruta de API
Paso 04	Probarlo
Al terminar	Checklist

PASO 01

El esquema

Instala zod 4 con `npm install zod` (probamos la 4.6.7); pide TypeScript 5.5 o superior con `"strict": true`. Los imports usan el alias `@/*` que Next.js crea por defecto.

```
// lib/order-schema.ts
import * as z from "zod";

export const OrderInput = z.object({
  name: z.string().trim().min(2, "Escribe tu nombre completo"),
  email: z.string().trim().pipe(z.email("Revisa tu correo")),
  postalCode: z.string().trim().regex(/^\d{5}$/, "El código postal tiene 5 dígitos"),
  quantity: z.coerce.number("Escribe un número").int("Solo unidades enteras").min(1, "Pide al menos
1").max(10, "Máximo 10 por pedido"),
  notes: z.string().trim().max(500, "Notas: máximo 500 caracteres").optional(),
});

export type OrderInput = z.infer<typeof OrderInput>;
```

- El correo se recorta antes de revisar el formato. `z.email().trim()` rechaza `" ana@example.com "`, porque revisa primero y recorta después.
- `z.coerce.number()` convierte el texto del formulario. Ojo: un campo vacío se vuelve 0 y cae en `.min(1)`.

PASO 02

El formulario

Una función sin React, para poder probarla en Node:

```
// lib/check-order.ts
import * as z from "zod";
import { OrderInput } from "@/lib/order-schema";

// Runs in the browser before anything is sent: same schema as the API.
// Returns the clean order, or the messages for each field.
export function checkOrder(data: FormData) {
  const result = OrderInput.safeParse(Object.fromEntries(data));
  if (!result.success) {
    return { ok: false as const, errors: z.flattenError(result.error).fieldErrors };
  }
  return { ok: true as const, order: result.data };
}
```

`safeParse` no lanza: devuelve `success` con `data` o con `error`. `z.flattenError` agrupa los mensajes por campo, como `{ email: ["Revisa tu correo"] }`. El componente la usa así:

```tsx
// components/OrderForm.tsx
"use client";
import { useState } from "react";
import { checkOrder } from "@/lib/check-order";

export function OrderForm() {
  const [errors, setErrors] = useState<Record<string, string[] | undefined>>({});

  async function onSubmit(event: React.FormEvent<HTMLFormElement>) {
    event.preventDefault();
    const checked = checkOrder(new FormData(event.currentTarget));
    if (!checked.ok) return setErrors(checked.errors);
    const res = await fetch("/api/orders", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(checked.order),
    });
    setErrors(res.status === 400 ? (await res.json()).fieldErrors : {});
  }

  return (
    <form onSubmit={onSubmit} noValidate>
      <label htmlFor="email">Correo</label>
      <input id="email" name="email" type="email" autoComplete="email"
        aria-invalid={Boolean(errors.email)} aria-describedby="email-error" />
      <p id="email-error">{errors.email?.[0]}</p>
      {/* name, postalCode, quantity and notes repeat the same pattern */}
      <button type="submit">Hacer pedido</button>
    </form>
  );
}
```

Repite etiqueta, campo y mensaje para cada clave del esquema; el `name` de cada campo debe coincidir con ella.

PASO 03

La ruta de API

Responde 400 con los errores por campo, también si el cuerpo no es JSON. Precio y stock nunca salen del cuerpo.

```tsx
// app/api/orders/route.ts
import * as z from "zod";
import { OrderInput } from "@/lib/order-schema";

export async function POST(request: Request) {
  const body: unknown = await request.json().catch(() => null);
  const result = OrderInput.safeParse(body);
  if (!result.success) {
    return Response.json(z.flattenError(result.error), { status: 400 });
```

```
  }
  const order = result.data; // OrderInput: trimmed, typed, unknown keys dropped
  // Save the order here. Price and stock come from your database, not from the body.
  return Response.json({ ok: true, quantity: order.quantity }, { status: 201 });
}
```

PASO 04

Probarlo

El script llama a `checkOrder` con un `FormData` y a `POST` con objetos `Request`, sin levantar el servidor; necesita `npm install -D tsx`. La extensión `.mts` permite el `await` de nivel superior aunque tu proyecto sea CommonJS.

```
// test-order.mts: run with npx tsx test-order.mts
import assert from "node:assert/strict";
import { checkOrder } from "@/lib/check-order";
import { POST } from "@/app/api/orders/route";

const form = new FormData();
const fields = { name: " Ana ", email: " ana@example.com ", postalCode: "06700", quantity: "2" };
for (const [k, v] of Object.entries(fields)) form.append(k, v);
const checked = checkOrder(form);
assert.ok(checked.ok);
assert.deepEqual(checked.order, { name: "Ana", email: "ana@example.com", postalCode: "06700", quantity: 2 });

const post = (body: string) => POST(new Request("http://localhost/api/orders", { method: "POST", body }));
const bad = await post(JSON.stringify({ name: "Ana", email: "x", postalCode: "06700", quantity: 50 }));
assert.equal(bad.status, 400);
console.log(await bad.json());
assert.equal((await post("{not json")).status, 400);
const good = await post(JSON.stringify({ ...fields, quantity: 2, price: 0 }));
assert.equal(good.status, 201);
console.log("OK");
```

Imprime los `fieldErrors` del pedido malo y `OK`. Agrega un caso por cada regla que cambies.

AL TERMINAR

Checklist

- [ ] El esquema vive en un archivo que importan el formulario y la ruta.
- [ ] La ruta usa `safeParse` y responde 400 con `z.flattenError`, también con JSON malformado.
- [ ] Los tipos salen de `z.infer`; no hay `interface` a mano ni `as`.
- [ ] Precio, stock y totales salen del servidor.