CoralinStudio — Formulario básico con Next.js (lista para crecer)

Esta guía te deja un formulario funcional (una sola página) con **Next.js (App Router)**, estilado con **Tailwind**, validado con **Zod + React Hook Form**, y un endpoint **/api/submit** que hoy guarda los envíos en un JSON local (`/data/submissions.json`). Más adelante podés cambiar el almacenamiento a Postgres/SQL Server/Supabase sin tocar el front.

---

1) Crear el proyecto (TypeScript + Tailwind)

```
# En E:\Programacion\REPOS
npx create-next-app@latest coralin-studio --ts --eslint --tailwind --app --
src-dir=false --import-alias "@/*"
cd coralin-studio
npm run dev
```

Abrí http://localhost:3000 para verificar.

> Si querés usar tu repo existente `CoralinStudio`, podés crear el proyecto dentro o copiar los archivos a tu carpeta actual.

---

2) Instalar dependencias de formularios

```
npm i react-hook-form zod @hookform/resolvers
```

---

3) Página con el formulario (`app/page.tsx`)

Reemplazá el contenido del archivo `app/page.tsx` por esto:

```
"use client";
import { useForm } from "react-hook-form";
import { z } from "zod";
import { zodResolver } from "@hookform/resolvers/zod";
import { useState } from "react";

const FormSchema = z.object({
  firstName: z.string().min(1, "Requerido"),
  lastName: z.string().min(1, "Requerido"),
  email: z.string().email("Email inválido"),
  phone: z.string().min(7, "Teléfono inválido").optional().or(z.literal("")),
```

```
  birthDate: z.string().optional(),
  howDidYouHear: z
    .enum(["instagram", "google", "amigo", "flyer", "otro"])
    .default("instagram"),
  medicalHistory: z.string().max(2000, "Máx. 2000 caracteres").optional(),
  consent: z.literal(true, { errorMap: () => ({ message: "Aceptá los
términos" }) }),
});

type FormValues = z.infer<typeof FormSchema>;

export default function Page() {
  const [status, setStatus] = useState<"idle" | "ok" | "error">("idle");

  const {
    register,
    handleSubmit,
    formState: { errors, isSubmitting },
    reset,
  } = useForm<FormValues>({ resolver: zodResolver(FormSchema) });

  const onSubmit = async (data: FormValues) => {
    setStatus("idle");
    try {
      const res = await fetch("/api/submit", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(data),
      });
      if (!res.ok) throw new Error("Error al enviar");
      setStatus("ok");
      reset();
    } catch (e) {
      setStatus("error");
    }
  };

  return (
    <main className="min-h-dvh bg-gray-50 py-10">
      <div className="mx-auto max-w-2xl rounded-2xl bg-white p-8 shadow">
        <h1 className="text-2xl font-semibold">Ficha de cliente — Estudio</
h1>
        <p className="mt-1 text-sm text-gray-600">
          Completá tus datos para que podamos atenderte mejor.
        </p>

        <form onSubmit={handleSubmit(onSubmit)} className="mt-6 space-y-5">
          <div className="grid grid-cols-1 gap-4 sm:grid-cols-2">
            <div>
              <label className="block text-sm font-medium">Nombre</label>
              <input {...register("firstName")} className="mt-1 w-full
```

```
rounded-lg border p-2" />
              {errors.firstName && (
                <p className="mt-1 text-sm text-
red-600">{errors.firstName.message}</p>
              )}
            </div>
            <div>
              <label className="block text-sm font-medium">Apellido</label>
              <input {...register("lastName")} className="mt-1 w-full
rounded-lg border p-2" />
              {errors.lastName && (
                <p className="mt-1 text-sm text-
red-600">{errors.lastName.message}</p>
              )}
            </div>
          </div>

          <div className="grid grid-cols-1 gap-4 sm:grid-cols-2">
            <div>
              <label className="block text-sm font-medium">Email</label>
              <input {...register("email")} type="email" className="mt-1 w-
full rounded-lg border p-2" />
              {errors.email && (
                <p className="mt-1 text-sm text-
red-600">{errors.email.message}</p>
              )}
            </div>
            <div>
              <label className="block text-sm font-medium">Teléfono</label>
              <input {...register("phone")}
className="mt-1 w-full rounded-lg border p-2" />
              {errors.phone && (
                <p className="mt-1 text-sm text-
red-600">{errors.phone.message}</p>
              )}
            </div>
          </div>

          <div className="grid grid-cols-1 gap-4 sm:grid-cols-2">
            <div>
              <label className="block text-sm font-medium">Fecha de
nacimiento</label>
              <input {...register("birthDate")} type="date" className="mt-1
w-full rounded-lg border p-2" />
            </div>
            <div>
              <label className="block text-sm font-medium">¿Cómo se enteró?</
label>
              <select {...register("howDidYouHear")} className="mt-1 w-full
rounded-lg border p-2">
                <option value="instagram">Instagram</option>
```

```
                <option value="google">Google</option>
                <option value="amigo">Recomendación de un amigo</option>
                <option value="flyer">Flyer/Cartelería</option>
                <option value="otro">Otro</option>
              </select>
            </div>
          </div>

          <div>
            <label className="block text-sm font-medium">Historia médica /
notas</label>
            <textarea {...register("medicalHistory")} rows={5}
className="mt-1 w-full rounded-lg border p-2" />
            {errors.medicalHistory && (
              <p className="mt-1 text-sm text-
red-600">{errors.medicalHistory.message}</p>
            )}
          </div>

          <div className="flex items-start gap-2">
            <input id="consent" type="checkbox" {...register("consent")}
className="mt-1" />
            <label htmlFor="consent" className="text-sm text-gray-700">
              Acepto el uso de mis datos con fines de atención y contacto.
            </label>
          </div>
          {errors.consent && (
            <p className="-mt-3 text-sm text-
red-600">{errors.consent.message}</p>
          )}

          <button
            type="submit"
            disabled={isSubmitting}
            className="inline-flex items-center justify-center rounded-xl
border bg-black px-4 py-2 text-white disabled:opacity-60"
          >
            {isSubmitting ? "Enviando…" : "Enviar"}
          </button>

          {status === "ok" && (
            <p className="text-green-700">¡Enviado! Gracias por completar
tus datos.</p>
          )}
          {status === "error" && (
            <p className="text-red-700">Ocurrió un error. Intentá de nuevo.</
p>
          )}
        </form>
      </div>
    </main>
```

```
  );
}
```

---

4) Endpoint para recibir y guardar (`app/api/submit/route.ts`)

Creá el archivo `app/api/submit/route.ts`:

```ts
import { NextResponse } from "next/server";
import { z } from "zod";
import { writeFile, mkdir, readFile } from "fs/promises";
import { existsSync } from "fs";
import path from "path";

const Payload = z.object({
  firstName: z.string().min(1),
  lastName: z.string().min(1),
  email: z.string().email(),
  phone: z.string().optional().nullable(),
  birthDate: z.string().optional().nullable(),
  howDidYouHear: z.enum(["instagram", "google", "amigo", "flyer", "otro"]),
  medicalHistory: z.string().optional().nullable(),
  consent: z.literal(true),
});

export async function POST(req: Request) {
  try {
    const json = await req.json();
    const data = Payload.parse(json);

    const record = {
      id: crypto.randomUUID(),
      receivedAt: new Date().toISOString(),
      ...data,
    };

    const dataDir = path.join(process.cwd(), "data");
    const dbFile = path.join(dataDir, "submissions.json");

    if (!existsSync(dataDir)) {
      await mkdir(dataDir, { recursive: true });
    }

    let arr: unknown[] = [];
    if (existsSync(dbFile)) {
      const current = await readFile(dbFile, "utf-8");
      if (current.trim().length) {
        arr = JSON.parse(current);
      }
```

```
    }

    arr.push(record);
    await writeFile(dbFile, JSON.stringify(arr, null, 2), "utf-8");

    return NextResponse.json({ ok: true });
  } catch (err) {
    console.error("/api/submit error", err);
    return NextResponse.json({ ok: false }, { status: 400 });
  }
}
```

Ahora cada envío se agrega a `data/submissions.json`. Para un MVP es suficiente. Después cambiamos esto por una DB real (Postgres/SQL Server/Supabase) sin tocar el front.

---

5) Tailwind (si el template no lo trajo)

Si creaste el proyecto sin Tailwind, seguí los pasos oficiales y asegurate de incluir en `globals.css` las directivas `@tailwind base; @tailwind components; @tailwind utilities;`.

---

6) Probar

1. `npm run dev`
2. Completar el formulario y enviar.
3. Ver `data/submissions.json` creciendo con cada envío.

---

7) Próximos pasos (cuando tu esposa quiera ver los datos)

- **Página de administración** (`/admin`) protegida por contraseña simple (middleware) para listar `data/submissions.json` (o la DB real).
- **Exportar a CSV/Excel**.
- Migrar de JSON a DB:
- **Prisma + SQLite** (sencillo local) → luego cambiar a **Postgres/SQL Server**.
- O usar **Supabase/Neon** (Postgres administrado) o **Azure SQL**.
- **Email de confirmación** tras submit (Resend/SendGrid) y **captcha**.
- **Política de privacidad / Términos** y casilla de consentimiento.

---

8) ¿Por qué Next.js sobre Vite para este caso?

- Te da **API routes** y **server actions** en la misma app (no necesitás levantar un backend aparte).
- Fácil de crecer a login, dashboards, exportaciones, etc.
- SEO y performance si después publicás más secciones del estudio.

Si preferís Vite + una API minimal (Express), también se puede. Pero para un formulario con persistencia y futuro crecimiento, **Next.js** simplifica.