# Contact Form

## Purpose

Contact page for **recruiters / collaborators / small missions**.

- Default = **recruitment** (so HR can send fast).
- If subject = **freelance** → show extra fields (company, budget, deadline).
- If subject = **collab** → show extra fields (project link, goal).
- Sends via **Mailjet SMTP** to `conatct@darrybook.fr`.
- Sanitized + Zod validated.
- Same visual shell as other pages.

---

## Routes

- `/[lang]/contact`
- API: `/api/form`

## 1. Dictionaries

**Contact FR** → app/dictionaries/fr/contact.json

```
{
  "title": "Me contacter",
  "subtitle": "Développeur Full-Stack JavaScript / TypeScript basé à Paris.
Recrutement, mission ou collaboration.",
  "meta": {
    "version": 1,
    "api": {
      "endpoint": "/api/form",
      "method": "POST",
      "headers": { "Content-Type": "application/json" }
    },
    "locale": "fr",
    "successRedirect": null,
    "captcha": false
  },
  "fields": [
    {
```

```
      "id": "name",
      "label": "Nom complet",
      "type": "text",
      "placeholder": "Jean Darry",
      "required": true,
      "width": "1/2"
    },
    {
      "id": "email",
      "label": "Email",
      "type": "email",
      "placeholder": "vous@domaine.fr",
      "required": true,
      "width": "1/2"
    },
    {
      "id": "subject",
      "label": "Sujet",
      "type": "select",
      "required": true,
      "width": "full",
      "options": [
        { "value": "recruitment", "label": "Recrutement" },
        { "value": "freelance", "label": "Mission / projet" },
        { "value": "collab", "label": "Collaboration" },
        { "value": "other", "label": "Autre" }
      ]
    },
    {
      "id": "company",
      "label": "Société",
      "type": "text",
      "placeholder": "ACME",
      "required": false,
      "width": "1/3",
      "showWhen": { "subject": ["freelance"] }
    },
    {
      "id": "budget",
      "label": "Budget",
      "type": "select",
      "required": false,
      "width": "1/3",
      "options": [
        { "value": "<1k", "label": "< 1 000 €" },
        { "value": "1k-3k", "label": "1 000 – 3 000 €" },
```

```
        { "value": "3k-5k", "label": "3 000 - 5 000 €" },
        { "value": "5k-10k", "label": "5 000 - 10 000 €" },
        { "value": "10k+", "label": "10 000 € +" }
      ],
      "showWhen": { "subject": ["freelance"] }
    },
    {
      "id": "deadline",
      "label": "Deadline",
      "type": "text",
      "placeholder": "Nov. 2025",
      "required": false,
      "width": "1/3",
      "showWhen": { "subject": ["freelance"] }
    },
    {
      "id": "projectLink",
      "label": "Lien du projet",
      "type": "url",
      "placeholder": "https:// ... ",
      "required": false,
      "width": "full",
      "showWhen": { "subject": ["collab"] }
    },
    {
      "id": "goal",
      "label": "Objectif / besoin",
      "type": "text",
      "placeholder": "Ce que vous voulez faire",
      "required": false,
      "width": "full",
      "showWhen": { "subject": ["collab"] }
    },
    {
      "id": "message",
      "label": "Message",
      "type": "textarea",
      "placeholder": "Décrivez rapidement le contexte.",
      "required": true,
      "rows": 5,
      "width": "full"
    }
  ],
  "privacy": "En envoyant ce formulaire, vous acceptez que vos données soient
utilisées pour vous répondre.",
  "submit": "Envoyer",
```

```
  "success": "Merci, je reviens vers vous rapidement."
}
```

---

## 2.Dependencies (pnpm)

```
pnpm add zod nodemailer sanitize-html
// types
pnpm add -D @types/nodemailer @types/sanitize-html
// test
pnpm add -D vitest @vitest/ui @vitest/coverage-v8
// dom test
pnpm add -D  @testing-library/react @testing-library/jest-dom jsdom
```

These handle:

- **zod** → form validation + schema typing
- **nodemailer** → SMTP transport (Mailjet)
- **sanitize-html** → XSS + HTML cleaning for messages

**Result Check:**
After install, `pnpm dev` should compile without missing packages.
All imports resolve (`zod`, `sanitize-html`, `nodemailer`).
Mail works locally via `.env` and on Vercel once env vars are set

## 3. Env

```
SMTP_HOST=in-v3.mailjet.com
SMTP_PORT=587
SMTP_SECURE=false
SMTP_USER=your-mailjet-api-key
SMTP_PASS=your-mailjet-secret-key
CONTACT_TO=contact@darrybook.fr
CONTACT_FROM=info@darrybook.fr
```

---

## Tests

Use **Vitest** for simplicity (works with Next 15).
Install:

Then add a tests/ folder:

```
tests/
 ├── contactSchema.test.ts
 └── formApi.test.ts
```

---

## 1. ContactSchema.test.ts

Validates Zod logic only.

```tsx
// tests/ContactForm.fields.test.tsx
import { render, screen, fireEvent } from "@testing-library/react";
import { describe, it, expect } from "vitest";
import { ContactForm } from "@/components/form/ContactForm";
import { contactDict } from "./__fixtures__/contactDict";

describe("ContactForm field rendering", () => {
  it("renders base fields", () => {
    render(<ContactForm lang="fr" dict={contactDict} />);
    expect(screen.getByLabelText("Nom complet")).toBeInTheDocument();
    expect(screen.getByLabelText("Email")).toBeInTheDocument();
    expect(screen.getByLabelText("Sujet")).toBeInTheDocument();
  });

  it("shows freelance-specific fields when subject is freelance", () => {
    render(<ContactForm lang="fr" dict={contactDict} />);
    const select = screen.getByLabelText("Sujet");
    fireEvent.change(select, { target: { value: "freelance" } });
    expect(screen.getByLabelText("Société")).toBeInTheDocument();
    expect(screen.getByLabelText("Budget")).toBeInTheDocument();
  });
});
```

---

## 2. formApi.test.ts

Mock API call → ensure validation + mail send logic is triggered.

```ts
import { describe, it, expect, vi } from "vitest";
import { POST } from "@/app/api/form/route";
import { sendMail } from "@/lib/mail";

vi.mock("@/lib/mail", () => ({
  sendMail: vi.fn(() => Promise.resolve())
}));
const originalFetch = global.fetch;
describe("/api/form route", () => {
  it("returns 200 for valid recruitment request", async () => {
    const mockReq = {
      json: async () => ({
        name: "Jean",
        email: "test@test.com",
        subject: "recruitment",
        message: "I like your work"
      })
    };

    const res: any = await POST(mockReq as any);
    const body = await res.json();

    expect(body.ok).toBe(true);
    expect(sendMail).toHaveBeenCalledOnce();
  });

  it("returns 422 for invalid payload", async () => {
    const mockReq = {
      json: async () => ({
        name: "Jean",
        subject: "recruitment",
        message: "missing email"
      })
    };

    const res: any = await POST(mockReq as any);
    const body = await res.json();

    expect(body.ok).toBe(false);
    expect(res.status).toBe(422);
  });
  afterEach(() => {

  global.fetch = originalFetch;
```

```
});
});
```

### Mock dictionary for the component

```
// tests/__fixtures__/contactDict.ts
export const contactDict = {
  meta: {
    api: {
      endpoint: "/api/form",
      method: "POST",
      headers: { "Content-Type": "application/json" },
    },
    locale: "fr",
    successRedirect: null,
    captcha: false,
  },
  title: "Me contacter",
  subtitle: "Développeur Full-Stack …",
  privacy: "En envoyant ce formulaire, vous acceptez que vos données soient
utilisées pour vous répondre.",
  submit: "Envoyer",
 "messages": {
    "success": "Merci, je reviens vers vous rapidement.",
    "error": "L'envoi a échoué. Réessayez dans quelques minutes.",
    "validation": "Certains champs sont invalides.",
    "loading": "Envoi …"
  },
  fields: [
    { id: "name", label: "Nom complet", type: "text", required: true, width:
"1/2" },
    { id: "email", label: "Email", type: "email", required: true, width: "1/2"
},
    {
      id: "subject",
      label: "Sujet",
      type: "select",
      required: true,
      width: "full",
      options: [
        { value: "recruitment", label: "Recrutement" },
        { value: "freelance", label: "Mission / projet" },
        { value: "collab", label: "Collaboration" },
        { value: "other", label: "Autre" },
      ],
    },
```

```
    {
      id: "company",
      label: "Société",
      type: "text",
      showWhen: { subject: ["freelance"] },
      width: "1/3",
    },
    {
      id: "budget",
      label: "Budget",
      type: "select",
      options: [
        { value: "<1k", label: "< 1 000 €" },
        { value: "1k-3k", label: "1 000 - 3 000 €" },
      ],
      showWhen: { subject: ["freelance"] },
      width: "1/3",
    },
    {
      id: "deadline",
      label: "Deadline",
      type: "text",
      showWhen: { subject: ["freelance"] },
      width: "1/3",
    },
    {
      id: "projectLink",
      label: "Lien du projet",
      type: "url",
      showWhen: { subject: ["collab"] },
      width: "full",
    },
    {
      id: "goal",
      label: "Objectif / besoin",
      type: "text",
      showWhen: { subject: ["collab"] },
      width: "full",
    },
    {
      id: "message",
      label: "Message",
      type: "textarea",
      required: true,
      rows: 5,
      width: "full",
```

```
    },
  ],
};
```

**Component test** tests/ContactForm.test.tsx

```tsx
// tests/ContactForm.test.tsx
import { render, screen, fireEvent, waitFor } from "@testing-library/react";
import { describe, it, expect, vi, beforeEach } from "vitest";
import { ContactForm } from "@/components/form/ContactForm";
import { contactDict } from "./__fixtures__/contactDict";

const originalFetch = global.fetch;

describe("ContactForm", () => {
  beforeEach(() => {
    global.fetch = vi.fn(() =>
      Promise.resolve({
        ok: true,
        json: () => Promise.resolve({ ok: true }),
      } as Response)
    );
  });

  // restore if needed
  // afterEach(() => {
  //   global.fetch = originalFetch;
  // });

  it("renders base fields by default (recruitment)", () => {
    render(<ContactForm lang="fr" dict={contactDict} />);

    expect(screen.getByLabelText("Nom complet")).toBeInTheDocument();
    expect(screen.getByLabelText("Email")).toBeInTheDocument();
    expect(screen.getByLabelText("Sujet")).toBeInTheDocument();
    expect(screen.getByLabelText("Message")).toBeInTheDocument();

    // extra freelance fields should NOT be there
    expect(screen.queryByLabelText("Société")).not.toBeInTheDocument();
    expect(screen.queryByLabelText("Budget")).not.toBeInTheDocument();
  });

  it("shows freelance extra fields when subject changes to freelance", () => {
    render(<ContactForm lang="fr" dict={contactDict} />);

    const subjectSelect = screen.getByLabelText("Sujet");
```

```
    fireEvent.change(subjectSelect, { target: { value: "freelance" } });

    expect(screen.getByLabelText("Société")).toBeInTheDocument();
    expect(screen.getByLabelText("Budget")).toBeInTheDocument();
    expect(screen.getByLabelText("Deadline")).toBeInTheDocument();
  });

  it("shows collab extra fields when subject changes to collab", () => {
    render(<ContactForm lang="fr" dict={contactDict} />);

    const subjectSelect = screen.getByLabelText("Sujet");
    fireEvent.change(subjectSelect, { target: { value: "collab" } });

    expect(screen.getByLabelText("Lien du projet")).toBeInTheDocument();
    expect(screen.getByLabelText("Objectif / besoin")).toBeInTheDocument();
    // freelance fields should stay hidden
    expect(screen.queryByLabelText("Société")).not.toBeInTheDocument();
  });

  it("submits to the endpoint defined in dict.meta.api", async () => {
    render(<ContactForm lang="fr" dict={contactDict} />);

    fireEvent.change(screen.getByLabelText("Nom complet"), {
      target: { value: "Jean Darry" },
    });
    fireEvent.change(screen.getByLabelText("Email"), {
      target: { value: "jd@test.com" },
    });
    fireEvent.change(screen.getByLabelText("Message"), {
      target: { value: "Hello recruiter" },
    });

    fireEvent.submit(screen.getByRole("button", { name: /envoyer/i
}).closest("form")!);

    await waitFor(() => {
      expect(global.fetch).toHaveBeenCalledTimes(1);
    });

    const call = (global.fetch as any).mock.calls[0];
    expect(call[0]).toBe("/api/form"); // from dict.meta.api
    expect(call[1].method).toBe("POST");
  });

  it("shows success text when API returns ok", async () => {
    render(<ContactForm lang="fr" dict={contactDict} />);
```

```
    fireEvent.change(screen.getByLabelText("Nom complet"), {
      target: { value: "Jean Darry" },
    });
    fireEvent.change(screen.getByLabelText("Email"), {
      target: { value: "jd@test.com" },
    });
    fireEvent.change(screen.getByLabelText("Message"), {
      target: { value: "Hello recruiter" },
    });

    fireEvent.submit(screen.getByRole("button", { name: /envoyer/i
}).closest("form")!);

    await waitFor(() => {
      expect(screen.getByText(contactDict.success)).toBeInTheDocument();
    });
  });

  it("keeps recruitment as default subject", () => {
    render(<ContactForm lang="fr" dict={contactDict} />);
    const subjectSelect = screen.getByLabelText("Sujet") as HTMLSelectElement;
    expect(subjectSelect.value).toBe("recruitment");
  });
});
```

---

## Vitest config

Add in package.json:

```
"scripts": {
  "test": "vitest"
}
```

Optional config file vitest.config.ts:

```
// vitest.config.ts
import { defineConfig } from "vitest/config";
import path from "node:path";

export default defineConfig({
  test: {
```

```
    globals: true,
    environment: "jsdom",              // default → React / components
    setupFiles: ["./tests/setup.ts"],
    include: ["tests/**/*.test.ts"],
    // folders that should run in node (no DOM)
    environmentMatchGlobs: [
      ["tests/api/**/*.test.ts", "node"],
      ["tests/server/**/*.test.ts", "node"],
    ],
  },
  resolve: {
    alias: {
      "@": path.resolve(__dirname, "."), // @ = project root
    },
  },
});
```

```
// tests/setup.ts
import "@testing-library/jest-dom";
```

---

## Run

```
pnpm test
```

✅ Tests cover schema validation and API response without hitting Mailjet.

# 4. Shared shell

components/form/FormShell.tsx

```
export function FormShell({
  title,
  subtitle,
  children,
}: {
  title: string;
  subtitle?: string;
  children: React.ReactNode;
}) {
  return (
```

```
    <main className="min-h-screen bg-black text-white">
      <section className="mx-auto max-w-5xl px-6 py-10">
        <div className="rounded-[48px] bg-[#d3d3d3] p-10 text-black">
          <header className="mb-6 space-y-2">
            <h1 className="text-3xl font-black tracking-tight uppercase">
{title}</h1>
            {subtitle ? (
              <p className="max-w-2xl text-sm text-black/70">{subtitle}</p>
            ) : null}
          </header>
          <div className="rounded-[40px] bg-black p-8 text-white">
            {children}
          </div>
        </div>
      </section>
    </main>
  );
}
```

---

## 5. Pages

### app/[lang]/contact/page.tsx

```
import { getDictionary } from "@/lib/getDictionary";
import { FormShell } from "@/components/form/FormShell";
import { ContactForm } from "@/components/form/ContactForm";

export default async function ContactPage({ params }: { params: { lang: string
} }) {
  const dict = await getDictionary(params.lang, "contact");
  return (
    <FormShell title={dict.title} subtitle={dict.subtitle}>
      <ContactForm lang={params.lang} dict={dict} />
    </FormShell>
  );
}
```

---

## 5. form fields (client)

```tsx
// components/form/FormFields.tsx
"use client";

type FieldDef = {
  id: string;
  label: string;
  type: string;
  placeholder?: string;
  required?: boolean;
  width?: "full" | "1/2" | "1/3";
  options?: { value: string; label: string }[];
  showWhen?: { subject?: string[] };
  rows?: number;
};

type FormFieldsProps = {
  fields: FieldDef[];
  lang?: string;
  subject?: string;
  onSubjectChange?: (next: string) => void;
};

export function FormFields({
  fields,
  lang = "fr",
  subject,
  onSubjectChange,
}: FormFieldsProps) {
  const visible = fields.filter((field) => {
    const cond = field.showWhen?.subject;
    if (!cond) return true;
    // if we don't have subject in parent, just hide conditional fields
    if (!subject) return false;
    return cond.includes(subject);
  });

  return (
    <div className="grid gap-4 md:grid-cols-2">
      {visible.map((field) => {
        const widthClass =
          field.width === "full"
            ? "col-span-2"
            : field.width === "1/3"
              ? "md:col-span-1"
              : "md:col-span-1";
```

```
        // subject field = special
        if (field.id === "subject") {
          return (
            <label key={field.id} className={widthClass + " flex flex-col gap-
1 text-sm"}>
              {field.label}
              <select
                name="subject"
                value={subject}
                onChange={(e) => onSubjectChange?.(e.target.value)}
                required={field.required}
                className="input"
              >
                {field.options?.map((opt) => (
                  <option key={opt.value} value={opt.value}>
                    {opt.label}
                  </option>
                ))}
              </select>
            </label>
          );
        }

        // textarea
        if (field.type === "textarea") {
          return (
            <label key={field.id} className="col-span-2 flex flex-col gap-1
text-sm">
              {field.label}
              <textarea
                name={field.id}
                rows={field.rows ?? 5}
                required={field.required}
                placeholder={field.placeholder}
                className="input resize-none"
              />
            </label>
          );
        }

        // select
        if (field.type === "select") {
          return (
            <label key={field.id} className={widthClass + " flex flex-col gap-
1 text-sm"}>
              {field.label}
```

```
              <select name={field.id} required={field.required}
className="input">
                <option value="">{lang === "fr" ? "Sélectionner" : "Select"}
</option>
                {field.options?.map((opt) => (
                  <option key={opt.value} value={opt.value}>
                    {opt.label}
                  </option>
                ))}
              </select>
            </label>
          );
        }

        // default input
        return (
          <label key={field.id} className={widthClass + " flex flex-col gap-1
text-sm"}>
            {field.label}
            <input
              name={field.id}
              type={field.type || "text"}
              required={field.required}
              placeholder={field.placeholder}
              className="input"
            />
          </label>
        );
      })}
    </div>
  );
}
```

## 6. Contact form (client)

components/form/Form.tsx

```
// components/form/ContactForm.tsx
"use client";

import { useState } from "react";
import { useRouter } from "next/navigation";
import { FormFields } from "./FormFields";

export function ContactForm({ lang = "fr", dict }: { lang?: string; dict: any
```

```
}) {
  const [subject, setSubject] = useState("recruitment");
  const [loading, setLoading] = useState(false);
  const [done, setDone] = useState(false);
  const [error, setError] = useState<string | null>(null);
  const router = useRouter();

  const endpoint = dict?.meta?.api?.endpoint ?? "/api/form";
  const method = dict?.meta?.api?.method ?? "POST";
  const headers = dict?.meta?.api?.headers ?? { "Content-Type":
"application/json" };
  const redirect = dict?.meta?.successRedirect ?? null;

  async function onSubmit(e: React.FormEvent<HTMLFormElement>) {
    e.preventDefault();
    const form = e.currentTarget;
    const data = Object.fromEntries(new FormData(form).entries());

    setLoading(true);
    setError(null);

    const res = await fetch(endpoint, {
      method,
      headers,
      body: JSON.stringify({ locale: lang, ...data }),
    });

    setLoading(false);

    if (res.ok) {
      if (redirect) {
        router.push(redirect);
        return;
      }
      setDone(true);
      form.reset();
      setSubject("recruitment");
      return;
    }

    let msg = dict?.messages?.error ?? "Une erreur est survenue.";
    try {
      const body = await res.json();
      if (body?.error === "VALIDATION_ERROR") {
        msg = dict?.messages?.validation ?? "Champs invalides.";
      }
```

```
    } catch {
      // ignore parse fail
    }
    setError(msg);
  }

  if (done) {
    return <p className="text-sm text-white/80">{dict.messages.success}</p>;
  }

  return (
    <form onSubmit={onSubmit} className="space-y-6">
      {error ? (
        <p className="rounded-xl bg-red-500/10 border border-red-500/40 px-4
py-2 text-xs text-red-100">
          {error}
        </p>
      ) : null}

      <FormFields
        fields={dict.fields ?? []}
        lang={lang}
        subject={subject}
        onSubjectChange={setSubject}
      />

      <div className="flex items-center justify-between gap-4">
        <p className="text-[10px] uppercase tracking-[0.2em] text-white/40
max-w-md">
          {dict.privacy}
        </p>
        <button
          type="submit"
          disabled={loading}
          className="rounded-full bg-white px-6 py-2 text-sm font-medium text-
black hover:bg-white/90 transition disabled:opacity-60"
        >
          {loading ? dict.messages.loading : dict.submit}
        </button>
      </div>
    </form>
  );
}
```

## 8. Zod schemas

lib/schemas/contact.ts

```
import { z } from "zod";

export const contactSchema = z.discriminatedUnion("subject", [
  z.object({
    name: z.string().trim().min(2),
    email: z.string().email(),
    subject: z.literal("recruitment"),
    message: z.string().trim().min(10),
    locale: z.enum(["fr","en"]).default("fr"),
    source: z.string().default("portfolio-contact")
  }),
  z.object({
    name: z.string().trim().min(2),
    email: z.string().email(),
    subject: z.literal("other"),
    message: z.string().trim().min(10),
    locale: z.enum(["fr","en"]).default("fr"),
    source: z.string().default("portfolio-contact")
  }),
  z.object({
    name: z.string().trim().min(2),
    email: z.string().email(),
    subject: z.literal("freelance"),
    message: z.string().trim().min(10),
    company: z.string().optional(),
    budget: z.enum(["<1k","1k-3k","3k-5k","5k-10k","10k+"]).optional(),
    deadline: z.string().optional(),
    locale: z.enum(["fr","en"]).default("fr"),
    source: z.string().default("portfolio-contact")
  }),
  z.object({
    name: z.string().trim().min(2),
    email: z.string().email(),
    subject: z.literal("collab"),
    message: z.string().trim().min(10),
    projectLink: z.string().url().optional(),
    goal: z.string().optional(),
    locale: z.enum(["fr","en"]).default("fr"),
    source: z.string().default("portfolio-contact")
  })
]);
```

## 9. Mail (SMTP via Mailjet)

lib/mail.ts

```
import nodemailer from "nodemailer";
import sanitizeHtml from "sanitize-html";

export const mailer = nodemailer.createTransport({
  host: process.env.SMTP_HOST,
  port: Number(process.env.SMTP_PORT || 587),
  secure: process.env.SMTP_SECURE === "true",
  auth: {
    user: process.env.SMTP_USER,
    pass: process.env.SMTP_PASS,
  },
});

export async function sendMail({
  to,
  subject,
  fromName,
  replyTo,
  html,
  text,
}: {
  to: string;
  subject: string;
  fromName: string;
  replyTo: string;
  html: string;
  text: string;
}) {
  const safeHtml = sanitizeHtml(html, {
    allowedTags:
["b","i","strong","em","a","p","br","ul","ol","li","h1","h2","h3"],
    allowedAttributes: { a: ["href"] },
  });
  const safeText = sanitizeHtml(text, { allowedTags: [], allowedAttributes: {}
});

  await mailer.sendMail({
    from: process.env.CONTACT_FROM,
    to,
    replyTo,
```

```
    subject,
    text: safeText,
    html: safeHtml,
  });
}
```

---

## 10. API routes

app/api/form/route.ts

```
import { NextRequest, NextResponse } from "next/server";
import { contactSchema } from "@/lib/schemas/contact";
import { sendMail } from "@/lib/mail";

export const dynamic = "force-dynamic";

export async function POST(req: NextRequest) {
  const json = await req.json();
  const parsed = contactSchema.safeParse(json);
  const headers = { "Cache-Control": "no-store" };
  if (!parsed.success) {
    return NextResponse.json({ ok: false, error: "VALIDATION_ERROR", issues:
parsed.error.flatten() }, { status: 422 });
  }

  const data = parsed.data;

  const subjectLabel =
    data.subject === "recruitment" ? "Recruitment" :
    data.subject === "freelance"   ? "Freelance Project" :
    data.subject === "collab"      ? "Collaboration" :
                                     "Other";

  const html = `
    <h2>New contact via portfolio</h2>
    <p><strong>Name:</strong> ${data.name}</p>
    <p><strong>Email:</strong> ${data.email}</p>
    <p><strong>Subject:</strong> ${subjectLabel}</p>
    ${data.company ? `<p><strong>Company:</strong> ${data.company}</p>` : ""}
    ${data.budget ? `<p><strong>Budget:</strong> ${data.budget}</p>` : ""}
    ${data.deadline ? `<p><strong>Deadline:</strong> ${data.deadline}</p>` :
""}
    ${data.projectLink ? `<p><strong>Project link:</strong>
```

```
${data.projectLink}</p>` : ""}
    ${data.goal ? `<p><strong>Goal:</strong> ${data.goal}</p>` : ""}
    <hr/>
    <p>${data.message.replace(/\n/g, "<br/>")}</p>
  `;

  try {
    await sendMail({
      to: process.env.CONTACT_TO!,
      subject: `📩 Contact via darrybook.fr (${subjectLabel})`,
      fromName: data.name,
      replyTo: data.email,
      html,
      text: data.message,
    });
    return NextResponse.json({ ok: true }, { status: 200, headers });
  } catch (err) {
    console.error("CONTACT_SMTP_ERROR", err);
    return NextResponse.json({ ok: false, error: "MAIL_FAILED" }, { status:
500 });
  }
}
```