Headless Next.js Contact Form (Email Notifications) — Step■by■Step Guide

Make a contact page in your existing Next.js + Vercel (Shopify headless) project that emails submissions to your chosen inbox — without storing anything in Shopify.

Who this is for

Beginners building a headless storefront with Next.js + Vercel, using Shopify only for catalog/checkout. You'll add a simple form that sends an email on submit.

What you'll build

- A Contact page (`/contact`) with Name, Email, Message fields.
- A serverless API endpoint (`/api/contact`) that sends an email via SMTP (e.g., Gmail App Password, Outlook, or any SMTP).
- Environment variables configured in Vercel for safe credentials.

Prerequisites

- An existing Next.js project deployed on Vercel.
- Node.js & npm installed locally.
- An email account that supports SMTP (Gmail with an App Password recommended, or SendGrid/Resend/Outlook SMTP).

Folder Conventions

This guide shows both Next.js routers. Use the section that matches your project:

- Pages Router → files under `pages/` (common in many projects).
- App Router → files under `app/` (Next.js 13+).

A) Pages Router (files in `pages/`)

1) Install dependency

```
npm install nodemailer
```

2) Create the Contact Page

Create `pages/contact.js` with a simple form that posts to `/api/contact`.

```
import { useState } from "react";

export default function Contact() {
  const [formData, setFormData] = useState({ name: "", email: "", message: "" });
  const [status, setStatus] = useState("");

  const handleSubmit = async (e) => {
    e.preventDefault();
    setStatus("Sending...");

    const res = await fetch("/api/contact", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(formData),
    });

    const data = await res.json();
    setStatus(data.success ? "Message sent ■" : `Error ■: ${data.error || "Failed"}`);
  };

  return (
    <main style={{ maxWidth: 560, margin: "40px auto", padding: "0 16px" }}>
      <h1>Contact Us</h1>
      <form onSubmit={handleSubmit} style={{ display: "grid", gap: 12 }}>
        <input
          type="text"
          placeholder="Name"
          required
          onChange={(e) => setFormData({ ...formData, name: e.target.value })}
        />
        <input
          type="email"
          placeholder="Email"
          required
          onChange={(e) => setFormData({ ...formData, email: e.target.value })}
        />
        <textarea
          placeholder="Message"
          rows={6}
          required
          onChange={(e) => setFormData({ ...formData, message: e.target.value })}
        />
        <button type="submit">Send</button>
      </form>
      <p>{status}</p>
    </main>
  );
}
```

3) Create the API Route

Create `pages/api/contact.js`. This uses Nodemailer to send the submission to your inbox via SMTP.

```
import nodemailer from "nodemailer";

export default async function handler(req, res) {
```

```
  if (req.method !== "POST") {
    return res.status(405).json({ error: "Method not allowed" });
  }

  const { name, email, message } = req.body || {};

  if (!name || !email || !message) {
    return res.status(400).json({ success: false, error: "Missing fields" });
  }

  try {
    const transporter = nodemailer.createTransport({
      service: "Gmail", // or configure host/port/secure for custom SMTP
      auth: {
        user: process.env.MAIL_USER,
        pass: process.env.MAIL_PASS,
      },
    });

    await transporter.sendMail({
      from: process.env.MAIL_FROM || process.env.MAIL_USER,
      replyTo: email,
      to: process.env.CONTACT_RECEIVER,
      subject: "New Contact Form Submission",
      text: `Name: ${name}\nEmail: ${email}\nMessage: ${message}`,
    });

    return res.status(200).json({ success: true });
  } catch (err) {
    console.error("[contact] mail error:", err);
    return res.status(500).json({ success: false, error: "Email failed" });
  }
}
```

4) Add Environment Variables

Create `.env.local` in your project root (never commit this):

```
MAIL_USER="your@gmail.com"
MAIL_PASS="your_gmail_app_password"
MAIL_FROM="Your Store <your@gmail.com>"
CONTACT_RECEIVER="support@yourstore.com"
```

Then in Vercel → Project → Settings → Environment Variables, add the same variables for production. Redeploy after adding them.

5) Test Locally

```
npm run dev
# open http://localhost:3000/contact, submit the form, check your inbox
```

B) App Router (files in `app/`)

1) Install dependency

```
npm install nodemailer
```

2) Create the Contact Page

Create `app/contact/page.tsx` (or `page.jsx`) that posts to `/api/contact`.

```
"use client";
import { useState } from "react";

export default function ContactPage() {
  const [formData, setFormData] = useState({ name: "", email: "", message: "" });
  const [status, setStatus] = useState("");

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    setStatus("Sending...");

    const res = await fetch("/api/contact", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(formData),
    });
    const data = await res.json();
    setStatus(data.success ? "Message sent ■" : `Error ■: ${data.error || "Failed"}`);
  };

  return (
    <main style={{ maxWidth: 560, margin: "40px auto", padding: "0 16px" }}>
      <h1>Contact Us</h1>
      <form onSubmit={handleSubmit} style={{ display: "grid", gap: 12 }}>
        <input
          type="text"
          placeholder="Name"
          required
          onChange={(e) => setFormData({ ...formData, name: e.target.value })}
        />
        <input
          type="email"
          placeholder="Email"
          required
          onChange={(e) => setFormData({ ...formData, email: e.target.value })}
        />
        <textarea
          placeholder="Message"
          rows={6}
          required
          onChange={(e) => setFormData({ ...formData, message: e.target.value })}
        />
        <button type="submit">Send</button>
      </form>
      <p>{status}</p>
    </main>
  );
}
```

3) Create the API Route (Server)

Create `app/api/contact/route.ts` (or `route.js`).

```
import nodemailer from "nodemailer";
import { NextResponse } from "next/server";
```

```
export async function POST(request: Request) {
  try {
    const { name, email, message } = await request.json();

    if (!name || !email || !message) {
      return NextResponse.json({ success: false, error: "Missing fields" }, { status: 400 })
    }

    const transporter = nodemailer.createTransport({
      service: "Gmail", // or custom SMTP: host, port, secure
      auth: {
        user: process.env.MAIL_USER,
        pass: process.env.MAIL_PASS,
      },
    });

    await transporter.sendMail({
      from: process.env.MAIL_FROM || process.env.MAIL_USER,
      replyTo: email,
      to: process.env.CONTACT_RECEIVER,
      subject: "New Contact Form Submission",
      text: `Name: ${name}\nEmail: ${email}\nMessage: ${message}`,
    });

    return NextResponse.json({ success: true });
  } catch (err) {
    console.error("[contact] mail error:", err);
    return NextResponse.json({ success: false, error: "Email failed" }, { status: 500 });
  }
}
```

4) Environment Variables

```
MAIL_USER="your@gmail.com"
MAIL_PASS="your_gmail_app_password"
MAIL_FROM="Your Store <your@gmail.com>"
CONTACT_RECEIVER="support@yourstore.com"
```

Add the same variables in Vercel → Project → Settings → Environment Variables and redeploy.

5) Test

```
npm run dev
# open http://localhost:3000/contact
```

Optional: Auto■Reply to the Customer

You can send a confirmation email back to the submitter. Just add a second `sendMail` call after the first one:

```
await transporter.sendMail({
  from: process.env.MAIL_FROM || process.env.MAIL_USER,
  to: email,
  subject: "Thanks for contacting us",
  text: "We received your message and will get back to you shortly."
});
```

Security & Deliverability Tips

- Use a Gmail App Password (Google Account → Security → App passwords) if using Gmail SMTP. Don’t use your normal password.
- Consider a dedicated sender like SendGrid or Resend for higher deliverability.
- Add a simple honeypot field or rate■limit to reduce spam.
- Never expose SMTP credentials in client■side code. Keep them in environment variables.

Troubleshooting

- `Email failed`: Check SMTP credentials and that MAIL_USER/MAIL_PASS/CONTACT_RECEIVER are set in both `.env.local` and Vercel.
- Gmail blocked sign■in: Use App Passwords; ensure 2■Step Verification is enabled.
- No email received but API returns success: Check spam folder, verify sender domain (SPF/DKIM).
- 500 error on production only: Ensure you redeployed after adding env vars in Vercel.

That’s it!

You now have a headless contact form that emails submissions—just like Shopify’s built■in form, but fully under your control in Next.js.