Mastering Contact Forms and Email Delivery (Next.js + Resend)

A practical, production-focused guide to implement, harden, and monitor email sending from web contact forms using Next.js App Router and Resend. Includes deliverability best practices, environment setup, and debugging playbooks.

Table of Contents

1) Architecture Overview

Use a server-side API endpoint to send emails to avoid exposing secrets. The browser posts JSON to `/api/contact`, which validates input and calls Resend.

```
Browser (form) → POST /api/contact → Resend → Inboxes (you + autorespond
```

2) Resend Setup

1. Create an account and an API key. Limit scope if possible.
2. For immediate testing, use `onboarding@resend.dev` as the sender.
3. For production, add and verify your domain. Publish DNS for DKIM/SPF/DMARC.

3) Next.js API Route (App Router)

Key points: validate input, guard env vars, await sends, and handle unknown errors safely.

```
import { NextRequest, NextResponse } from 'next/server'
import { Resend } from 'resend'

export async function POST(req: NextRequest) {
  try {
    const { name, email, message } = await req.json()
    if (!name || !email || !message) {
      return NextResponse.json({ error: 'Missing fields' }, { status: 40
    }

    const resend = new Resend(process.env.RESEND_API_KEY)
    const from = 'Brand <onboarding@resend.dev>'

    const result = await resend.emails.send({
      from,
      to: 'owner@example.com',
      reply_to: email,
      subject: `New message from ${name}`,
      html: `

${message}

`
    })

    return NextResponse.json({ ok: true, id: result?.data?.id })
  } catch (err) {
    const msg = err instanceof Error ? err.message : String(err)
    return NextResponse.json({ error: 'Failed', details: msg }, { status
  }
}
```

4) Front-end Form UX

- Validate email client-side and server-side.
- Disable the submit button while sending; show success/error notices.
- Prefer accessible colors and focus rings.

5) Environment Variables

```
# .env.local (dev)
RESEND_API_KEY=...your_key...
RESEND_DOMAIN=yourdomain.com

# Vercel → Project Settings → Environment Variables
```

```
# Add the same keys for Production and Preview
```

6) Deliverability (SPF/DKIM/DMARC)

- **SPF**: Authorizes Resend to send on behalf of your domain.
- **DKIM**: Cryptographic signature proving authenticity.
- **DMARC**: Policy for how receivers treat failed SPF/DKIM.

Configure these in your DNS as instructed by Resend after adding your domain.

7) Troubleshooting Checklist

- Confirm API key exists in the environment (dev and prod).
- For production, use your verified domain as the sender.
- Check Resend dashboard logs for each request (status, errors).
- Inspect Vercel function logs for runtime exceptions.
- Try sending to a single recipient and plain HTML to isolate issues.
- Check spam/Promotions folders; add a recognizable From name.

8) Operational Playbooks

Blue/Green Sender

Keep `onboarding@resend.dev` as a fallback sender during outages of your custom domain.

Rate Limiting

Throttle requests server-side to prevent abuse. Implement CAPTCHA for public forms.

9) Security and Abuse Prevention

- Never expose API keys to the client.
- Sanitize inputs; use length limits and content checks.
- Add CAPTCHA or turnstile on public endpoints.

10) Reusable Snippets

Minimal Email

```
await resend.emails.send({
  from: 'Brand <onboarding@resend.dev>',
  to: 'owner@example.com',
```

```
  subject: 'Test',
  html: '

Test

'
})
```

Auto-reply

```
await resend.emails.send({
  from: 'Brand <onboarding@resend.dev>',
  to: userEmail,
  subject: 'We received your message',
  html: '

Thanks! We will reply shortly.

'
})
```