VCola.ai - Cloudflare Worker v2

Form handler - worker.js (updated with Turnstile + Google Sheets)

What this Worker does

- Verifies Cloudflare Turnstile token to block bots

- Rate-limits form submissions to 5 per IP per hour

- Handles CV uploads as base64 and attaches them as PDF to outgoing emails

- Routes emails to the correct address depending on the form type (contact vs. job application vs. newsletter) via Resend

- Logs form submissions (email + date) to a Google Sheet in the background

Environment variables required

Add the following secrets in the Worker settings:

- **RESEND_API_KEY** : API key from your Resend account

- **TURNSTILE_SECRET_KEY** : Secret key from Cloudflare Turnstile dashboard

- **GOOGLE_SERVICE_ACCOUNT_EMAIL** : vcola-forms@vcola-website.iam.gserviceaccount.com

- **GOOGLE_PRIVATE_KEY** : Private key from the service account JSON file (with \n literal characters)

- **GOOGLE_SHEET_ID** : 1DXcdXlfVdnzPffq8Q3LpeeiiTZ5RoxLnEVnG66YwKzc

KV Namespace

Create a KV namespace and bind it to the Worker with the variable name **FORM_RATE_LIMIT**.

Google Sheets

The Google Sheet used is shared with the service account vcola-forms@vcola-website.iam.gserviceaccount.com (Editor access). The Sheet ID and credentials are managed via the Google Cloud project vcola-website.

Virus scanning for CV uploads

To add automatic virus scanning in the future, two things are required:

- Upgrade to the **Cloudflare Workers Paid plan** ($5/month) to support the longer processing time needed for file scanning.

- Integrate the **VirusTotal API**, which is free up to 500 requests/day and handles the actual file analysis.

Worker code - worker.js

```
export default {
```

```
async fetch(request, env, ctx) {

  const corsHeaders = {
    "Access-Control-Allow-Origin": "*",
    "Access-Control-Allow-Methods": "POST, OPTIONS",
    "Access-Control-Allow-Headers": "Content-Type",
  };

  if (request.method === "OPTIONS") {
    return new Response(null, { headers: corsHeaders });
  }

  if (request.method !== "POST") {
    return new Response("Method not allowed", { status: 405 });
  }

  // --- FORM DATA ---
  const formData = await request.json();
  const { formType, cv, cvName, ...fields } = formData;

  // --- TURNSTILE VERIFICATION ---
  const turnstileToken = formData['cf-turnstile-response'];
  const verifyRes = await fetch('https://challenges.cloudflare.com/turnstile/v0/siteverify', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      secret: env.TURNSTILE_SECRET_KEY,
      response: turnstileToken,
    }),
  });
  const verifyData = await verifyRes.json();
  if (!verifyData.success) {
    return new Response(
      JSON.stringify({ error: "Bot detected. Submission blocked." }),
      { status: 403, headers: { ...corsHeaders, "Content-Type": "application/json" } }
    );
  }

  // --- RATE LIMITING ---
  const ip = request.headers.get("CF-Connecting-IP") || "unknown";
  const key = `rate:${ip}`;
  const LIMIT = 5;
  const WINDOW = 3600;

  const current = await env.FORM_RATE_LIMIT.get(key);
  const count = current ? parseInt(current) : 0;

  if (count >= LIMIT) {
    return new Response(
      JSON.stringify({ error: "Too many submissions. Please try again later." }),
      { status: 429, headers: { ...corsHeaders, "Content-Type": "application/json" } }
    );
  }

  await env.FORM_RATE_LIMIT.put(key, String(count + 1), {
    expirationTtl: WINDOW,
  });

  const subject = formType === 'apply'
```

```
  ? 'Role Application - VCola'
  : formType === 'newsletter'
  ? 'Newsletter Signup - VCola'
  : 'Early Adopter - VCola';

const to = 'hello@vcola.ai';

const emailPayload = {
  from: "form@vcola.ai",
  to: to,
  subject: subject,
  html: `<pre>${JSON.stringify(fields, null, 2)}</pre>`,
};

if (cv && cvName) {
  emailPayload.attachments = [{ filename: cvName, content: cv }];
}

// --- RESEND ---
const resendResponse = await fetch("https://api.resend.com/emails", {
  method: "POST",
  headers: {
    "Authorization": `Bearer ${env.RESEND_API_KEY}`,
    "Content-Type": "application/json",
  },
  body: JSON.stringify(emailPayload),
});
const resendResult = await resendResponse.json();
console.log('Resend status:', resendResponse.status);
console.log('Resend result:', JSON.stringify(resendResult));

// --- GOOGLE SHEETS (background) ---
const sheetsPromise = (async () => {
  try {
    const token = await getGoogleAuthToken(env);
    if (!token) return;

    const sheetsResponse = await fetch(
      `https://sheets.googleapis.com/v4/spreadsheets/${env.GOOGLE_SHEET_ID}/values/Sheet1!A1:append?valueIn
      {
        method: 'POST',
        headers: {
          'Authorization': `Bearer ${token}`,
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          values: [[new Date().toISOString(), fields['email'] || '']],
        }),
      }
    );
    const sheetsText = await sheetsResponse.text();
    console.log('Sheets status:', sheetsResponse.status);
    console.log('Sheets raw:', sheetsText.substring(0, 500));
  } catch (err) {
    console.log('Google Sheets error:', err.message);
  }
})();

ctx.waitUntil(sheetsPromise);
```

```
    return new Response(
      JSON.stringify({ success: true, message: "Form received!" }),
      { status: 200, headers: { ...corsHeaders, "Content-Type": "application/json" } }
    );
  },
};

async function getGoogleAuthToken(env) {
  try {
    const header = btoa(JSON.stringify({ alg: 'RS256', typ: 'JWT' }));
    const now = Math.floor(Date.now() / 1000);
    const claim = btoa(JSON.stringify({
      iss: env.GOOGLE_SERVICE_ACCOUNT_EMAIL,
      scope: 'https://www.googleapis.com/auth/spreadsheets',
      aud: 'https://oauth2.googleapis.com/token',
      exp: now + 3600,
      iat: now,
    }));

    const key = await crypto.subtle.importKey(
      'pkcs8',
      pemToArrayBuffer(env.GOOGLE_PRIVATE_KEY),
      { name: 'RSASSA-PKCS1-v1_5', hash: 'SHA-256' },
      false,
      ['sign']
    );

    const signature = await crypto.subtle.sign(
      'RSASSA-PKCS1-v1_5',
      key,
      new TextEncoder().encode(`${header}.${claim}`)
    );

    const signatureArray = new Uint8Array(signature);
    let binary = '';
    for (let i = 0; i < signatureArray.length; i++) {
      binary += String.fromCharCode(signatureArray[i]);
    }
    const jwtSignature = btoa(binary)
      .replace(/\+/g, '-').replace(/\//g, '_').replace(/=/g, '');
    const jwt = `${header}.${claim}.${jwtSignature}`;

    const tokenRes = await fetch('https://oauth2.googleapis.com/token', {
      method: 'POST',
      headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
      body: `grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer&assertion=${jwt}`,
    });

    const tokenData = await tokenRes.json();
    return tokenData.access_token;
  } catch (err) {
    console.log('getGoogleAuthToken error:', err.message);
    return null;
  }
}

function pemToArrayBuffer(pem) {
  const normalized = pem.replace(/\\n/g, '\n');
  const b64 = normalized
```

```
    .replace(/-----BEGIN PRIVATE KEY-----/, '')
    .replace(/-----END PRIVATE KEY-----/, '')
    .replace(/\s+/g, '');
  const binary = atob(b64);
  const buffer = new ArrayBuffer(binary.length);
  const view = new Uint8Array(buffer);
  for (let i = 0; i < binary.length; i++) view[i] = binary.charCodeAt(i);
  return buffer;
}
```