Universal Script:

1. Lead-capture.js

- Universal lead capture script for customer websites.

Features:

- Normalizes varied form field names into a fixed schema:
- Name, PhoneNumber, EmailAddress, LeadDetails, Status, SubmissionURL, LeadSource, CustomerName, CreatedAt, UpdatedAt
- Uses explicit mappings first, then heuristics (rule of thumbs)
- Works across multiple forms (static or dynamically injected)
- Sends JSON payload to an n8n webhook via POST
- Preserves unmapped fields as extra (to avoid data loss)

// code from below

```
(function () {
  // --- Config — (The configuration for that particular customer)
  const WEBHOOK_URL = (window.LEAD_CAPTURE_WEBHOOK || "").trim();
  const CUSTOMER_ID = (window.LEAD_CAPTURE_CUSTOMER || "unknown").trim();
  const MAPPINGS = window.LEAD_CAPTURE_MAPPINGS || {};

  if (!WEBHOOK_URL) {
    console.warn("[LeadCapture] Missing window.LEAD_CAPTURE_WEBHOOK. Payloads will not be sent.");
  }

  // --- Helpers — for timestamp and field extraction fn
  const nowIso = () => new Date().toISOString();

  function normalizeField(name, value) {
    const lower = name.toLowerCase();

    // 1. Explicit customer mappings - if manual fields wanted and mapping provided
    if (MAPPINGS[name]) {
      return { key: MAPPINGS[name], value };
    }

    // 2. Heuristics for target fields - if manually no mapping fed
    if (lower.includes("name") && !lower.includes("customer")) {
      return { key: "Name", value };
    }
    if (lower.includes("customer")) {
      return { key: "CustomerName", value };
    }
    if (lower.includes("phone") || lower.includes("tel") || lower.includes("mobile")) {
```

```
    return { key: "PhoneNumber", value };
  }
  if (lower.includes("mail") || lower.includes("email")) {
    return { key: "EmailAddress", value };
  }
  if (lower.includes("message") || lower.includes("comment") || lower.includes("detail") ||
lower.includes("lead")) {
    return { key: "LeadDetails", value };
  }
  if (lower.includes("status")) {
    return { key: "Status", value };
  }
  if (lower.includes("url") || lower.includes("link")) {
    return { key: "SubmissionURL", value };
  }
  if (lower.includes("source")) {
    return { key: "LeadSource", value };
  }
  if (lower.includes("created")) {
    return { key: "CreatedAt", value };
  }
  if (lower.includes("updated")) {
    return { key: "UpdatedAt", value };
  }

  // 3. Preserve everything else - every other data fields that the form had and was not
categorized or wanted in the o/p

  return { key: `extra_${name}`, value };
}

function buildPayload(form) {
  const formData = new FormData(form);
  const fields = {};
  for (const [name, value] of formData.entries()) {
    const { key, value: normalizedValue } = normalizeField(name, value);
    fields[key] = normalizedValue;
  }

  return {
    sourceUrl: window.location.href,
    timestamp: nowIso(),
    customerId: CUSTOMER_ID,
    fields
  };
}

async function sendPayload(payload) {
```

```
  if (!WEBHOOK_URL) return;
  try {
    await fetch(WEBHOOK_URL, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(payload),
      keepalive: true
    });
  } catch (err) {
    console.error("[LeadCapture] Error sending payload:", err);
  }
}

// Attach to all forms as asked
function attach() {
  const forms = document.querySelectorAll("form");
  forms.forEach((form) => {
    if (form.dataset.leadCaptureBound === "true") return;
    form.dataset.leadCaptureBound = "true";

    form.addEventListener("submit", (event) => {
      try {
        const payload = buildPayload(form);
        sendPayload(payload);
      } catch (err) {
        console.error("[LeadCapture] Prepare/send error:", err);
      }
    });
  });
}

// Observe dynamically injected forms (SPAs, CMS)
const observer = new MutationObserver(() => attach());

document.addEventListener("DOMContentLoaded", () => {
  attach();
  observer.observe(document.body, { childList: true, subtree: true });
});
})();
```

Important:

Form handling

- ***Multiple forms: The script attaches to all elements. Make sure you only want to capture those forms (e.g., avoid newsletter sign-ups unless intended).***
- ***Dynamic forms (SPAs/CMS): MutationObserver ensures new forms are bound, but needs test on sites that load forms via JavaScript to confirm.***

- ***Duplicate submissions: The script prevents double binding, but if a form auto-submits multiple times, we may need deduplication logic in n8n.***

Required elements on client code:

- Form tags (<form>...</form>):

- The script looks specifically for <form> elements in the DOM.

- If inputs are not wrapped in a <form>, the script won't detect submissions.

- Each form should have a proper submit event (either a <button type="submit"> or pressing Enter in a field).

Input fields with name attributes:

- The script uses the name attribute to normalize data.

- Example: <input name="email" /> → becomes EmailAddress.

- If a field has no name, it won't be captured.

Submit button:

- <button type="submit">Send</button> or <input type="submit" value="Submit" />.

- Without a submit trigger, the form won't fire the event listener.

- Custom JavaScript submit handlers that bypass the native submit event (unless they still trigger the event).

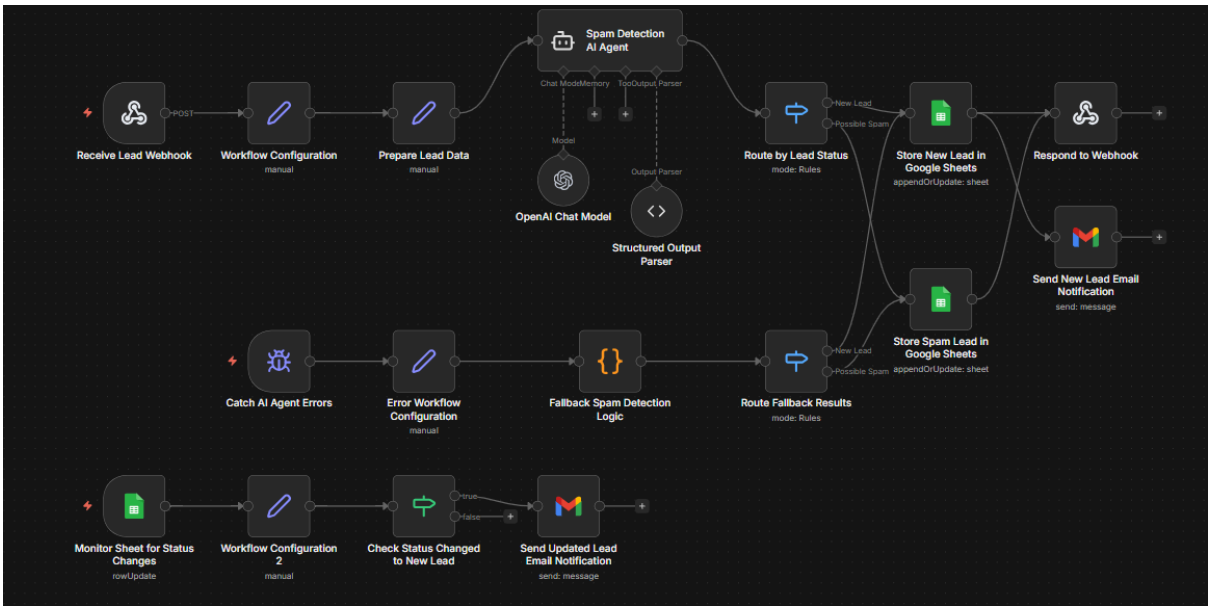
- Forms hidden inside iframes (the script only sees the current page DOM).

2. Configuration script

To be used inside the HTML file

```
<script>
window.LEAD_CAPTURE_CUSTOMER = "abc-industries"; // customer name
 window.LEAD_CAPTURE_WEBHOOK =
"https://taru2106.app.n8n.cloud/webhook/lead-webhook"; //n8n webhook
 window.LEAD_CAPTURE_MAPPINGS = {
   cx_name: "CustomerName", //custom mapping example
   os_name: "LeadSource", //custom mapping example
   fmi_name: "CustomerName" //custom mapping example
 };
</script>
<script src="lead-capture.js"></script>
```

N8N workflow:



Overview

- **Flow 1: Lead Processing**
  Triggered by a webhook. It receives lead data, checks for spam using AI with a fallback system, saves the results in a Google Sheet, and sends emails when appropriate.
- **Flow 2: Status Monitoring**
  Triggered by changes in the sheet. It looks for leads that were marked as spam but later upgraded to valid leads, then sends notifications.

Flow 1: Lead Processing

Step 1: Receive Lead Webhook
A webhook listens for incoming POST requests at a defined URL. It accepts any JSON structure from forms, captures extra fields automatically, and returns an error for GET requests.

Step 2: Workflow Configuration
A Set node stores reusable values like the customer's email and name. This makes it easy to update recipient details in one place.

Step 3: Prepare Lead Data
Incoming data is normalized into a consistent format. Standard fields are extracted, missing ones default to "Not provided," and extra fields are captured as metadata.

Step 4: AI Spam Detection
Lead data is sent to an AI model, which checks for spam indicators such as invalid emails, fake names, suspicious phone numbers, or poor message quality. It returns a structured JSON with a status ("New Lead" or "Possible Spam") and a reason. If the AI fails, the workflow moves to the fallback system.

Step 5: Error Trigger
If the AI node fails due to issues like API errors or timeouts, the Error Trigger node captures the problem. It preserves the original lead data and logs the error details.

Step 6: Fallback Rule-Based Spam Detection
When the AI fails, a JavaScript node runs simple rules to detect spam. It checks for test emails, fake names, invalid phone numbers, or suspicious messages. If two or more indicators are found, the lead is marked as spam; otherwise, it's marked as valid. The reason field notes that fallback rules were used.

Step 7: Set Fallback Status
The fallback output is formatted to match the AI's output structure, ensuring consistency for the rest of the workflow.

Step 8: Merge Paths
Both the AI path and the fallback path are merged back together. This guarantees the workflow continues smoothly regardless of which path was taken.

Step 9: Route by Lead Status
A Switch node checks the spam detection result. If status is "New Lead," the workflow stores the lead and sends an email. If status is "Possible Spam," the workflow stores the lead but skips email notifications.

Step 10A: Store New Lead
A Google Sheets node appends a new row with all lead details and marks the status as "New Lead." It also saves timestamps and any extra metadata.

Step 10B: Store Spam Lead
Spam leads are also stored in the sheet but marked as “Possible Spam.” This creates an audit trail and allows manual review.

Step 11A: Send Email for New Lead
A Gmail node sends an email to the customer contact when a new lead is received. The email includes all lead details and any extra metadata.

Step 11B: No Email for Spam
Spam leads are only logged in the sheet. No email is sent to avoid unnecessary alerts.

Step 12: Respond to Webhook
The workflow sends a response back to the form or script confirming receipt. The response includes success status, the lead classification, and a timestamp. This prevents timeouts and gives immediate feedback.

Flow 2: Status Monitoring

Step 13: Monitor Sheet
A Google Sheets trigger checks for updates every few minutes. If any row changes, the workflow runs again.

Step 14: Workflow Configuration 2
A Set node defines the customer email and name for this monitoring flow. This can be different from the main flow.

Step 15: Check Status Change
An IF node looks specifically for leads that change from “Possible Spam” to “New Lead.” Only this transition triggers the next step, preventing duplicate or irrelevant emails.

Step 16: Send Updated Lead Email
A Gmail node sends an email when a spam lead is upgraded to a valid lead. The email explains that the lead was reviewed and reclassified.

Edge Cases Covered

- Flexible JSON input with missing fields handled.
- AI failures trigger fallback detection.
- Spam leads logged for review.
- Emails only sent for valid leads.
- Monitoring flow prevents duplicate notifications.
- Immediate webhook responses ensure smooth user experience.

Configuration Checklist

- Set up a Google Sheet with the required columns.
- Replace placeholder Sheet IDs and customer emails in the workflow.
- Connect credentials for OpenAI, Gmail, and Google Sheets.
- Activate the workflow so the monitoring trigger runs.

Testing Scenarios

1. **Legitimate lead:** AI marks as New Lead → stored, email sent, webhook confirms.
2. **Spam lead:** AI marks as Possible Spam → stored, no email, webhook confirms.
3. **Fallback detection:** Disconnect AI → fallback rules classify lead, workflow continues.
4. **Status change monitoring:** Update a spam lead to New Lead in the sheet → trigger detects change, email sent.

This is the complete rephrased explanation of your workflow, presented in one clear narrative.

Rollout to Three Customers

1. Small Startups / MSMEs (new websites)

Why target them:
They often rely on a single contact form for all inbound leads. Every genuine inquiry matters, but they don't have the bandwidth to sift through spam.

Rollout approach:

- Provide a simple copy-paste script for their contact page.
- Store leads in Google Sheets (easy, free, familiar).
- Send email notifications directly to the founder or small team.
- Use fallback spam detection to guarantee reliability even if the AI fails.

Conversion impact:
Clean leads and instant alerts help them respond quickly, improving their chance of winning early customers.

2. Professional Services Firms (agencies, consultancies, law firms)

Why target them:
They depend on inbound consultations and project inquiries. Spam wastes time and can cause missed opportunities.

Rollout approach:

- Embed the script into "Contact Us" and "Book a Consultation" forms.
- Normalize fields like service type, preferred date, and client details.
- Store leads in Google Sheets or Airtable for easy review.
- Send structured email notifications to the managing partner or account manager.

Conversion impact:
Ensures only qualified leads reach the team, reducing admin overhead and increasing timely follow-ups.

3. Real Estate / High-Value Local Businesses

Why target them:
They rely on property inquiries, quotes, or bookings. Spam leads can overwhelm small sales teams.

Rollout approach:

- Add the script to property inquiry or booking forms.
- Normalize fields like property ID, location, and budget.
- Store leads in Sheets or CRM with SMS/email notifications to agents.
- Use fallback rules to catch obvious spam so agents only see genuine inquiries.

Conversion impact:
Agents spend time only on real prospects, improving response speed and increasing deal closure rates.

Scenarios Accounted For

- **Multiple forms:** Script attaches to all tags without duplication.
- **Dynamic content:** MutationObserver captures forms added later by SPAs or CMS.
- **Flexible field names:** Explicit mappings and heuristics normalize data.
- **Partial submissions:** Missing fields default to "Not provided."
- **Extra fields:** Captured as JSON metadata for auditability.
- **Spam detection:** AI classification with fallback rule-based checks.
- **Error handling:** AI failures (timeouts, API errors) trigger fallback logic.
- **Notifications:** Emails only sent for valid leads, avoiding spam fatigue.
- **Audit trail:** Spam leads stored for manual review and possible upgrades.
- **Immediate feedback:** Webhook responds with status to prevent form timeouts.

Next Steps with Two More Weeks

- **Duplicate detection:** Add lookup or hashing logic to prevent duplicate entries.
- **Customer dashboard:** Web UI for configuring mappings, spam keywords, and recipient emails.
- **Advanced spam scoring:** Weighted scoring system with thresholds for more nuanced classification.
- **Observability:** Error alerts to Slack/Whatsapp and daily summary reports.
- **Extras viewer:** Parse extra_* fields into readable columns for easier customer review.
- **Cooldown logic:** Prevent multiple notifications if status changes rapidly.

Troubleshooting Leads Not Being Captured

1. **Check form structure**
    - Ensure forms use tags and inputs have name attributes.
    - Verify a submit button exists.
2. **Verify configuration**
    - Confirm window.LEAD_CAPTURE_WEBHOOK and window.LEAD_CAPTURE_CUSTOMER are set correctly.
    - Check mappings for unusual field names.

3. **Inspect browser console**
    - Look for errors logged by the script during submission.
4. **Check webhook and workflow**
    - Ensure the n8n webhook URL is active and accessible.
    - Review execution logs for errors.
5. **Validate credentials**
    - Confirm Google Sheets, Gmail, and OpenAI credentials are connected.
    - Ensure the workflow is set to “Active.”
6. **Test with sample data**

- Send a test POST request with known values.
- Verify the lead appears in the sheet and notifications are sent.