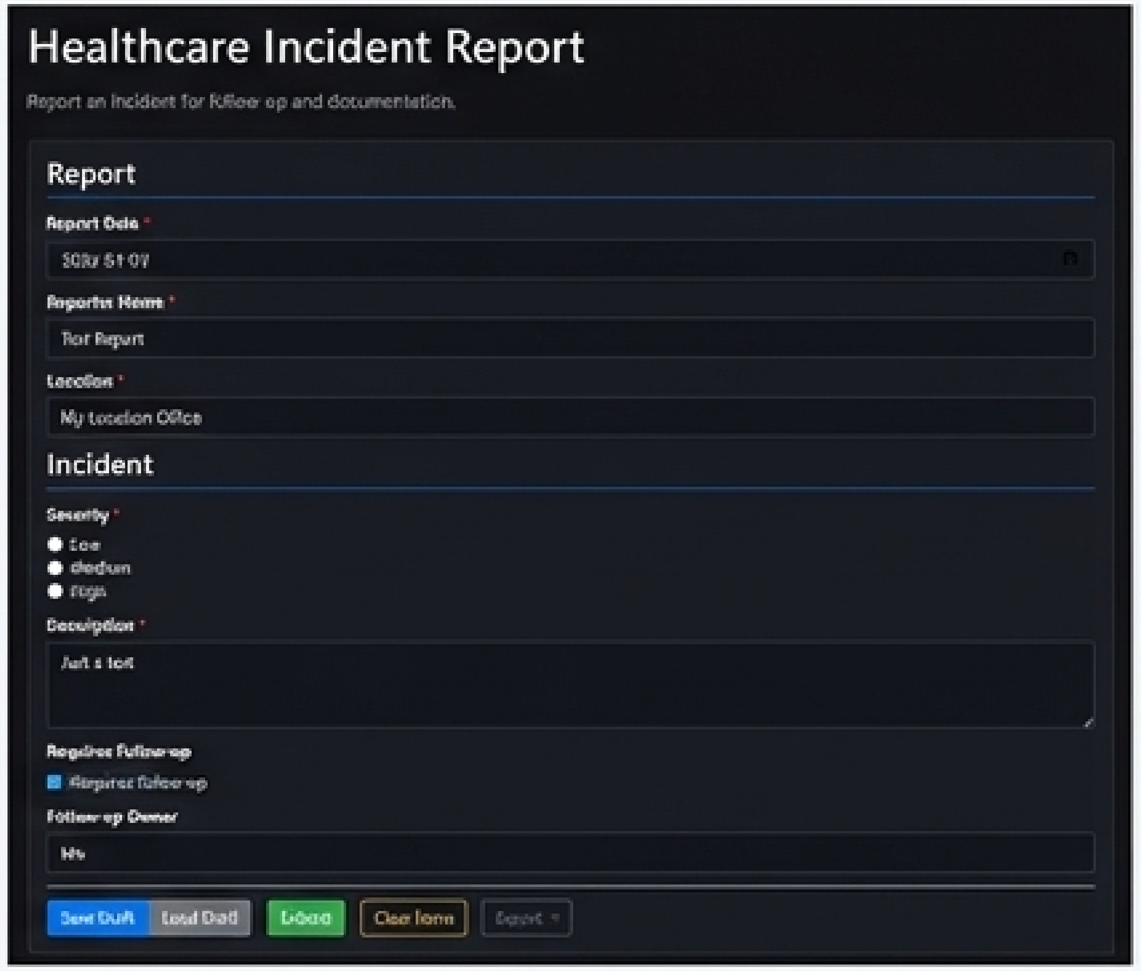
PortalSmith FormGen: From Schema to Live Web Form in 15 Minutes

A step-by-step guide to generating a fully functional data entry portal without writing any front-end code.



Healthcare Incident Report

Report an incident for follow-up and documentation.

Report

Report Date *
3026-01-07

Reporter Name *
Test Report

Location *
My Location Office

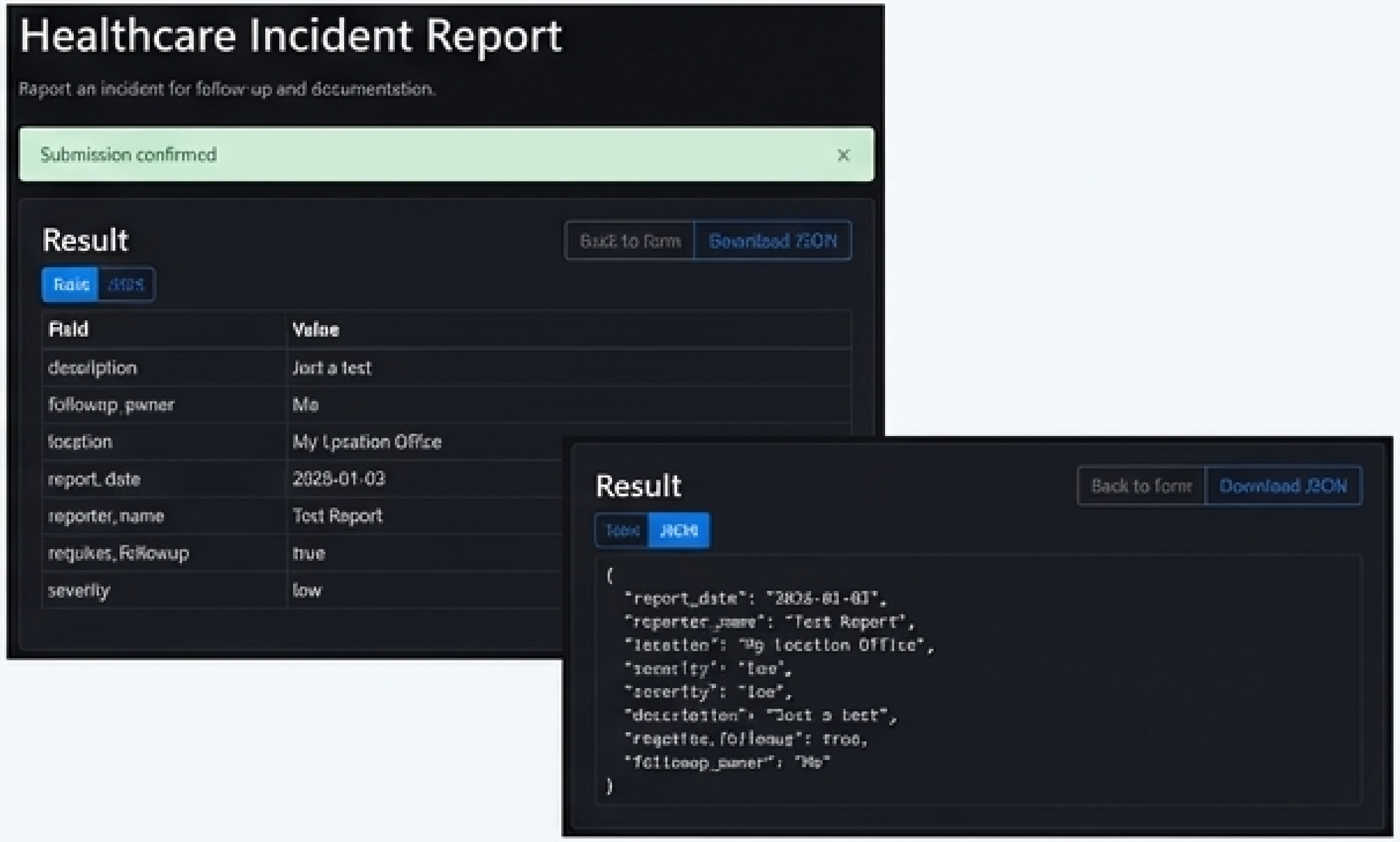
Incident

Severity *
● Low
● Medium
● High

Description *
Just a test

Requires Follow-up
☑ Requires follow-up

Follow-up Owner
Me



Healthcare Incident Report

Report an incident for follow-up and documentation.

Submission confirmed ×

Result

Field	Value
description	Just a test
followup_owner	Me
location	My Location Office
report_date	2026-01-03
reporter_name	Test Report
requires_followup	true
severity	low

Result

```
{
  "report_date": "2026-01-03",
  "reporter_name": "Test Report",
  "location": "My Location Office",
  "security": "low",
  "severity": "low",
  "description": "Just a test",
  "requires_followup": true,
  "followup_owner": "Me"
}
```

Today, we will build this exact web application. We'll start with a JSON definition and end with a live, interactive portal. This entire process is automated and requires no build tools like Webpack or Vite.

- Verify your environment and install PortalSmith FormGen.
- Create a Schema Builder UI to design our form.
- Import an example schema for a Healthcare Incident Report.
- Generate a `uibuilder` form portal with a single click.
- Use the form: save/load drafts, submit data.
- View the submitted results in your browser.

Setting the Stage: Required Environment

PortalSmith FormGen relies on modern Node-RED and `` `uibuilder` `` features. Please ensure your environment meets these minimum versions for a smooth experience.

Minimum Versions

- **Node.js**: 18.x LTS or newer (20.x LTS recommended)
- **Node-RED**: 3.1 or newer (4.x recommended)
- **`node-red-contrib-uibuilder`**: v7.x (latest v7 recommended)
- **Browser**: Modern Chrome, Firefox, or Edge

Critical Technical Notes

⚠ A Word on TLS

Modern browsers will **not** allow a web page to make direct requests to an API with an invalid or self-signed TLS certificate. This is a browser security feature, not a PortalSmith limitation.

The Solution

If you need to call a self-signed API, use the built-in **Server-side API Proxy** feature in the `uibuilder-formgen` node. This routes the call through your trusted Node-RED server, bypassing the browser's restriction.

Authentication

This quick start focuses on form generation. Authentication and authorization are not built-in and should be handled separately in your Node-RED flows.

NotebookLM

Step 1: Verify and Install `uibuilder` v7

The Why

The generated form portals require the `uibuilder` v7 client API (`uibuilder.iife.min.js`). Older versions (like v4, often included in OEM installs) are incompatible. This check ensures our foundation is correct before we build on it.

The How

1. **Check Version**
   In your Node-RED user directory (e.g., `~/.node-red`), check the installed version via the Palette Manager or command line.
2. **Important Warning**
   If you have an OEM or third-party Node-RED installation, you might have `uibuilder` v4. Upgrading will overwrite it. Proceed only if you don't have existing portals depending on an older version.
3. **Upgrade/Install**
   In your Node-RED user directory, run:

   ```
   npm install node-red-contrib-uibuilder
   ```

4. **Restart Node-RED**
   A restart is mandatory for Node-RED to recognize the updated node.

Common Restart Methods

Installation Method	**Command to Restart**
Official Linux Script	`node-red-restart`
`systemd` Service	`sudo systemctl restart nodered`
Local Terminal	**Ctrl-C**, then `node-red`
Docker Container	`docker restart [container_name]`

Step 2: Install PortalSmith FormGen

The Why

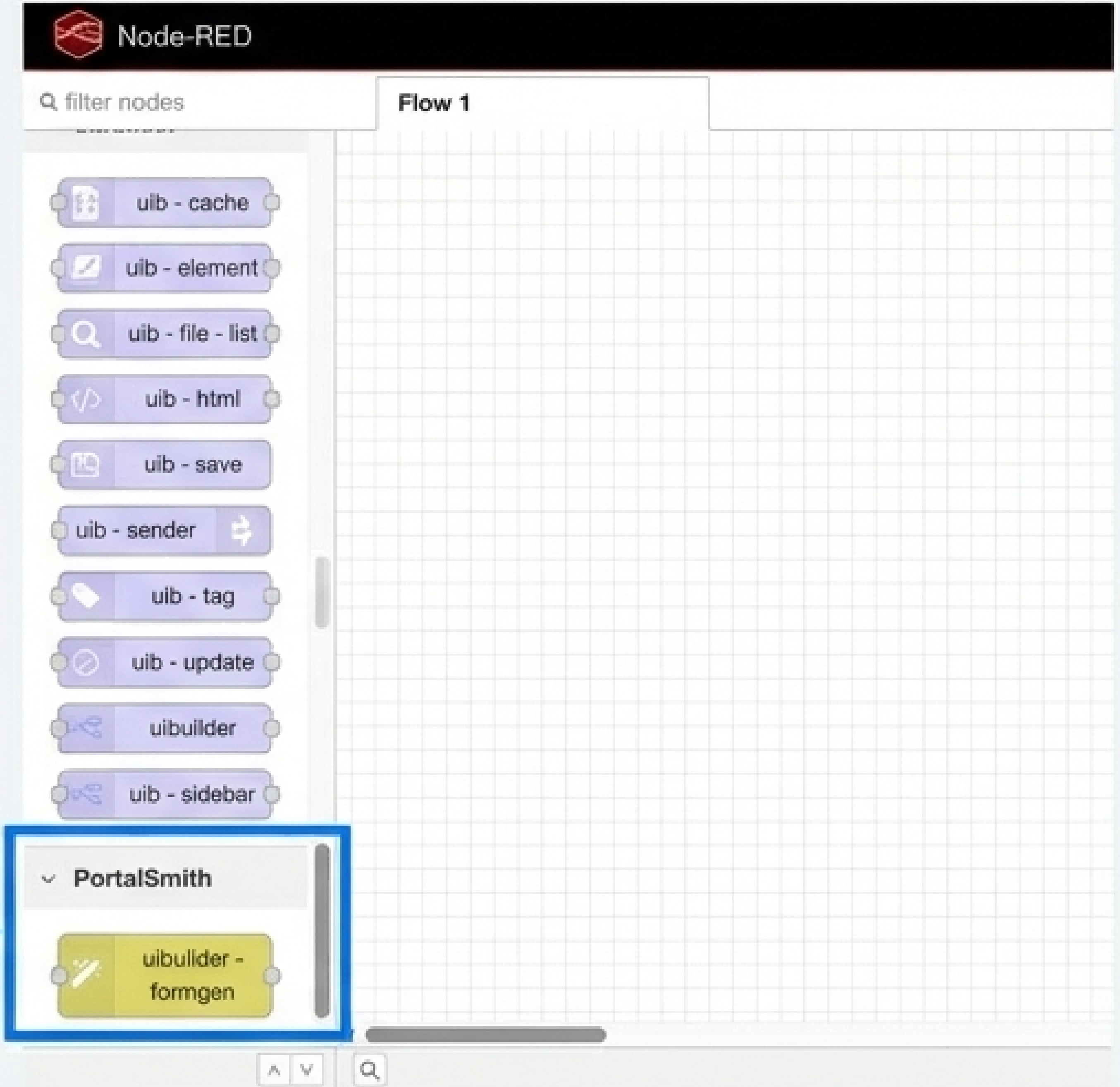
Installing the `uibuilder-formgen` package adds the core node to Node-RED's palette, which does the heavy lifting of converting our JSON schema into HTML and JavaScript files.

The How

1. **Copy Package:** Copy the `node-red-contrib-uibuilder-formgen-x.x.x.tgz` file to your Node-RED home directory (e.g., `~/.node-red`).
2. **Install via npm:** Navigate to your Node-RED home directory in a terminal and run the install command:

   ```
   npm install ./node-red-contrib-uibuilder-formgen-x.x.x.tgz
   ```

   (Note: Replace `x.x.x` with the actual version number.)
3. **Restart Node-RED:** Just like with `uibuilder`, a restart is required. Use the method appropriate for your setup.
4. **Verify Installation:** After restart, open the Node-RED editor. The `uibuilder-formgen` node should now appear in your node palette, under a 'PortalSmith' category.



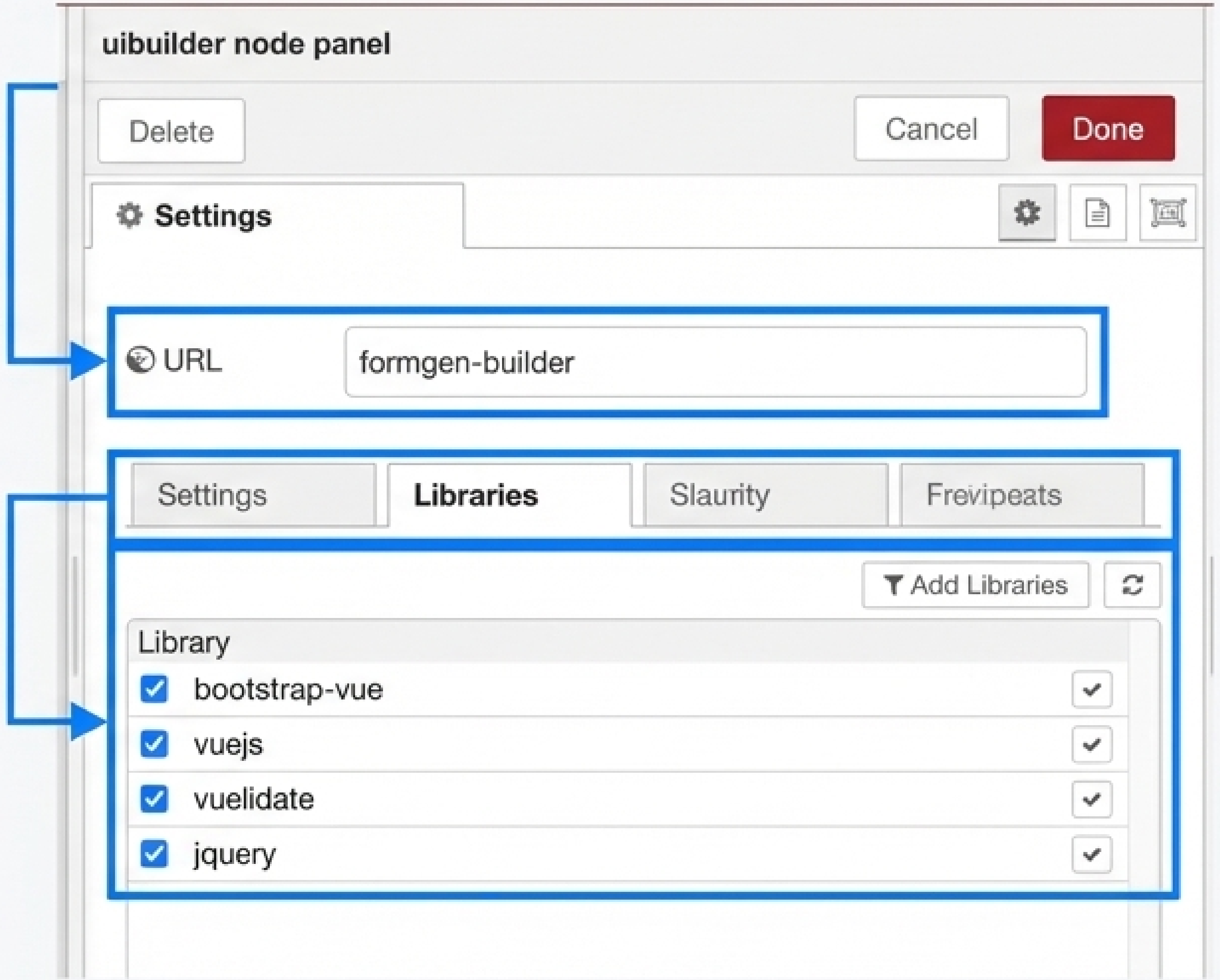
Step 3 & 4: Create and Configure the Schema Builder

The Why

The Schema Builder is a web UI that helps us create and edit form schemas visually. We host this UI using a standard `uibuilder` node. Deploying the node creates the necessary server-side folders for our UI files.

The How

1. **Create a New Flow**
   In Node-RED, click the ➕ icon in the flow tabs to create a blank canvas.
2. **Add `uibuilder` Node**
   Drag a `uibuilder`` node from the palette onto the canvas.
3. **Name the Instance**
   Double-click the node. In the URL field, enter **formgen-builder**. This exact name is important. Click **Done**.
4. **Deploy**
   Click the main **Deploy** button. This action creates the `~/.node-red/uibuilder/formgen-builder` directory on your server.
5. **Add Libraries**
   Re-open the `formgen-builder` node and go to the **Libraries** tab. Add the following required libraries one by one: `bootstrap-vue, vuejs, vuelidate, jquery.`



uibuilder node panel

URL: formgen-builder

Library	
☑ bootstrap-vue	✓
☑ vuejs	✓
☑ vuelidate	✓
☑ jquery	✓

(These are runtime dependencies loaded by the browser; no build step is needed.)

NotebookLM

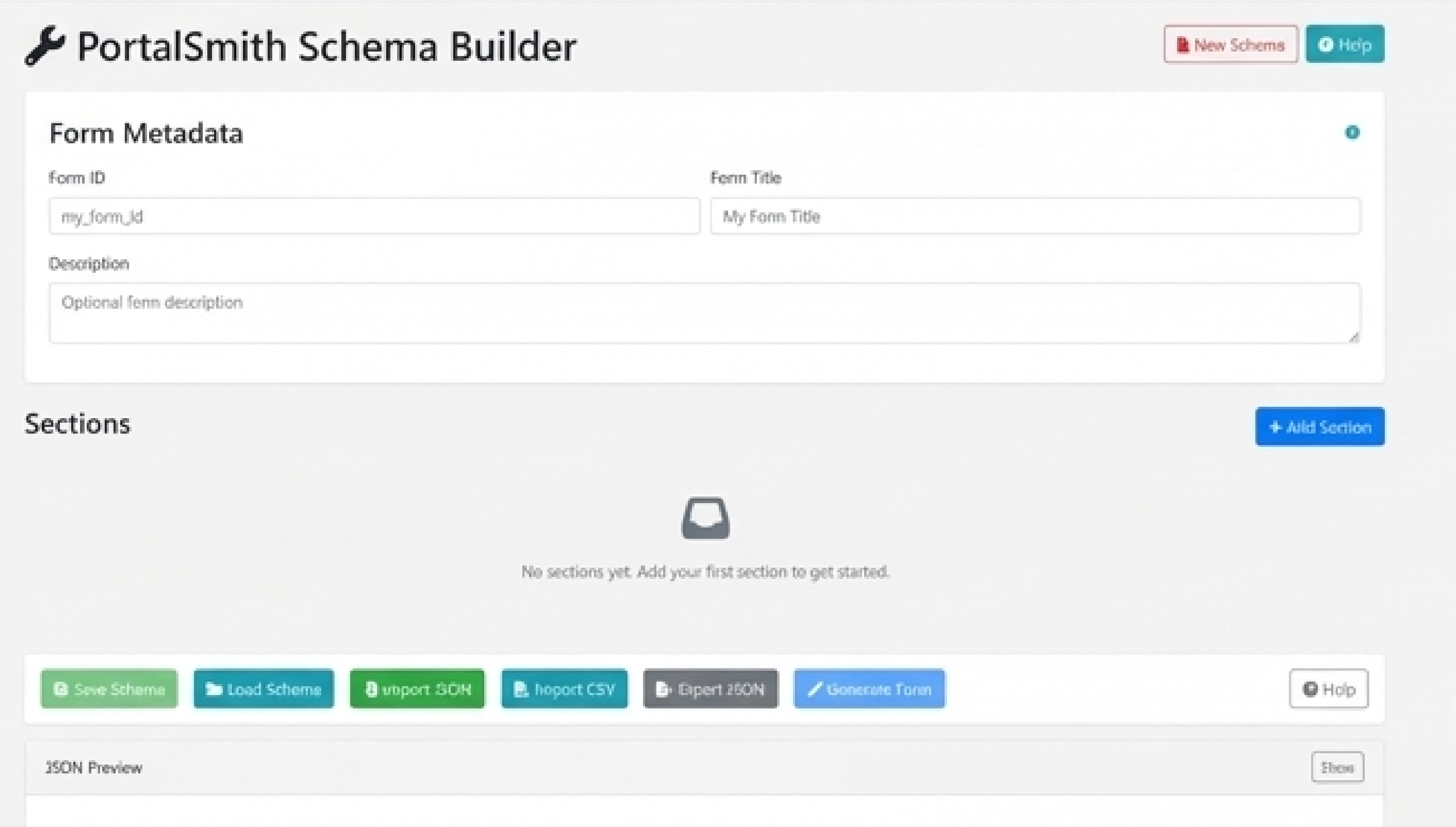
Step 5: Load the Schema Builder Files

The Why

`uibuilder` serves static files from a specific source directory. We need to copy the pre-built HTML and JavaScript files for the Schema Builder application into this directory so they can be accessed via a browser.

The How

1. **Locate Source Directory:** On your Node-RED server's filesystem, navigate to the newly created directory: `.../uibuilder/formgen-builder/src` (The full path depends on your Node-RED home directory.)
2. **Copy Files:** Copy the `index.html` and `index.js` files from the PortalSmith package into this `src` directory.
3. **Open the UI**: You can now access the Schema Builder. Either double-click the `formgen-builder` node in Node-RED and click the 'Open' button, or navigate directly to the URL:
   `http://<your-node-red-ip>:1880/formgen-builder`



PortalSmith Schema Builder

Form Metadata

Form ID: my_form_id

Fenn Title: My Fonn Title

Description: Optional fenn description

Sections

No sections yet. Add your first section to get started.

JSON Preview

NotebookLM

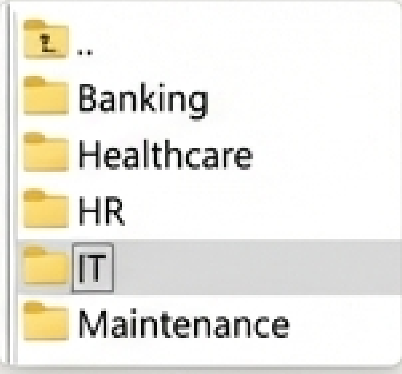
Step 6: Import an Example Schema

The Why

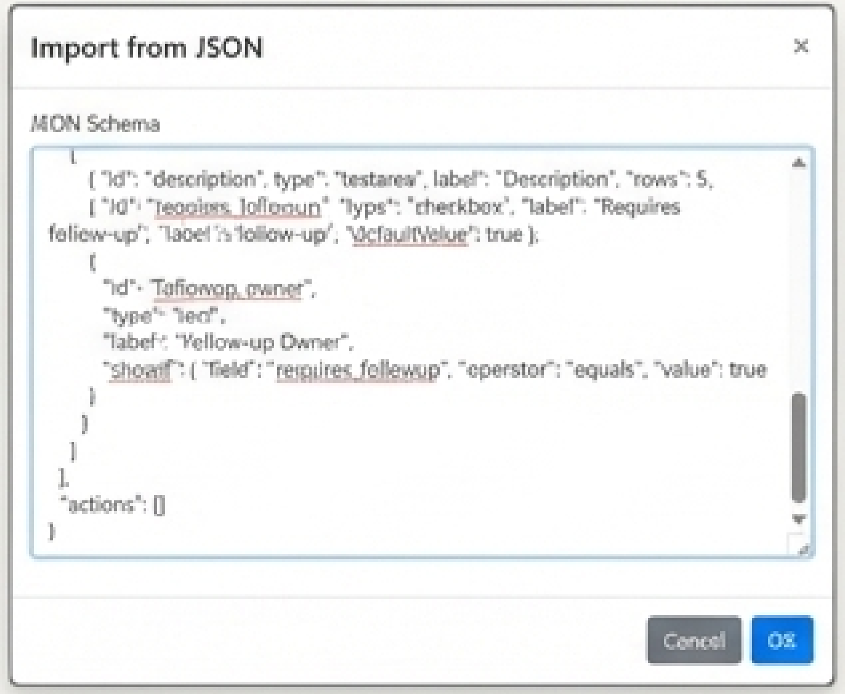
To get started quickly, PortalSmith provides ready-to-use example schemas for common industries. We will import one to see how the builder translates a JSON structure into a configurable form definition.

The How

1. **Locate Schemas:** The example schemas are located in your Node-RED modules directory: `` `../node_modules/node-red-contrib-uibuilder-formgen/examples/schemas/` ``
2. **Choose a Schema:** Inside, you'll find folders for industries like IT, HR, Healthcare, etc. Open a JSON file from one of these folders (e.g., from Healthcare) and copy its entire content.



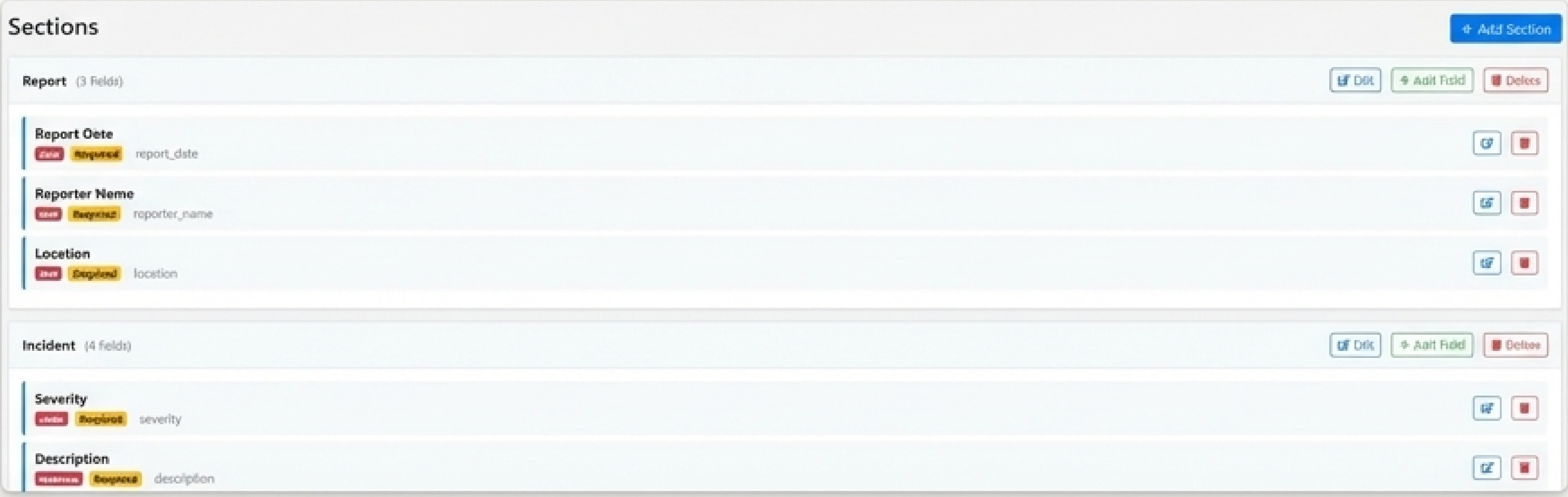
3. **Import:** In the Schema Builder UI, click the **Import JSON** button.
4. **Paste and Confirm:** Paste the copied JSON into the dialog box and click **OK**.
5. **Verify:** The builder will now be populated with sections and fields from the schema, showing labels, types, and required flags.



Import from JSON

MON Schema

```
    {
      { "id": "description", type": "testarea", label": "Description", "rows": 5,
      { "Id": "teopires_lolloaun" "lyps": "checkbox", "label": "Requires
fellew-up", "laoel ':s follow-up', 'VcfaultVelue": true };
      {
        "id"- Tafiowop_owner",
        "type"- "leo",
        "label": "Yellow-up Dwner",
        "showif": ( "field": "rerpiires_follewup", "cperstor": "equals", "value": true
      }
    }
  ]
 ],
 "actions": []
}
```



Sections

Report (3 Fields)

Report Oete
report_date

Reporter Neme
reporter_name

Location
location

Incident (4 fields)

Severity
severity

Description
description

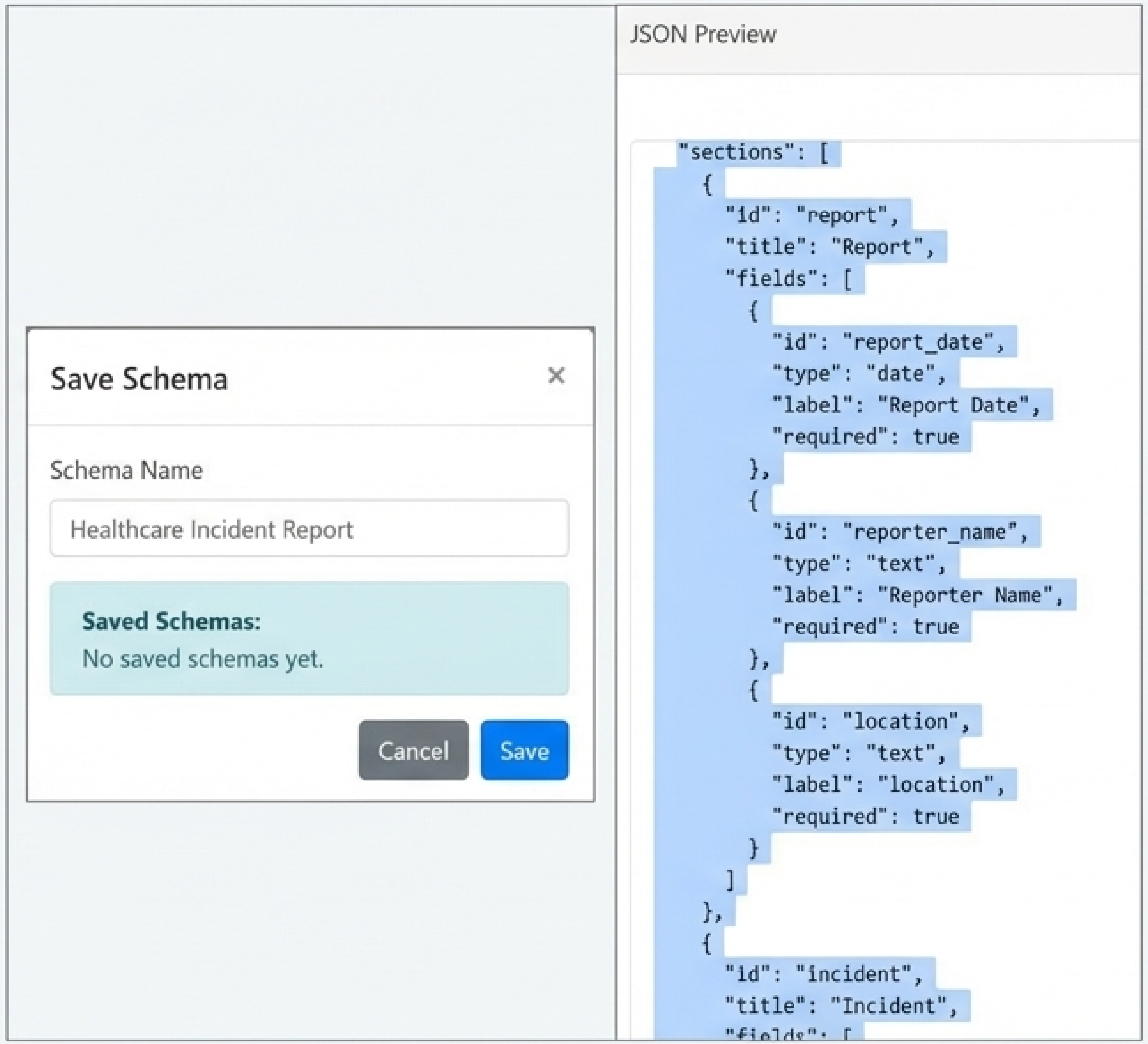
Step 7: Save and Export the Schema

The Why

The JSON schema is the complete definition—the 'contract'—for our form. We now copy this schema from the builder so we can pass it to the `uibuilder-formgen` node in Node-RED to generate the final portal.

The How

1. **Save (Optional but good practice)**
   Click **Save Schema** and give it a name (e.g., 'Healthcare Incident Report'). This saves it in your browser's local storage for later use.
2. **Export**
   Click **Export JSON**. This will either download the schema as a file or, more conveniently, display it in the **JSON Preview** box at the bottom of the page.
3. **Copy to Clipboard**
   Select and copy the **entire contents** of the JSON Preview text area.



Save Schema

Schema Name: Healthcare Incident Report

Saved Schemas:
No saved schemas yet.

```
"sections": [
  {
    "id": "report",
    "title": "Report",
    "fields": [
      {
        "id": "report_date",
        "type": "date",
        "label": "Report Date",
        "required": true
      },
      {
        "id": "reporter_name",
        "type": "text",
        "label": "Reporter Name",
        "required": true
      },
      {
        "id": "location",
        "type": "text",
        "label": "location",
        "required": true
      }
    ]
  },
  {
    "id": "incident",
    "title": "Incident",
    "fields": [
```

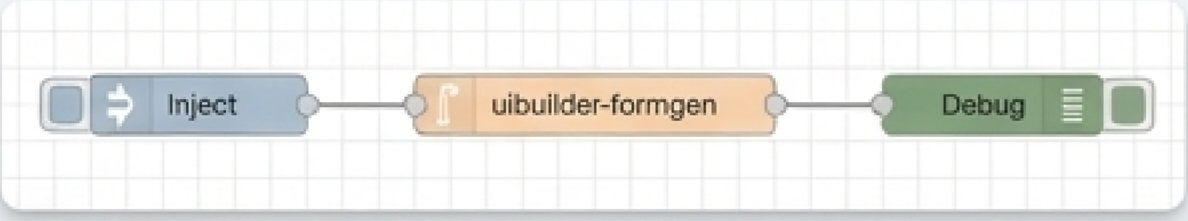
Step 8: Configure the Form Generation Flow

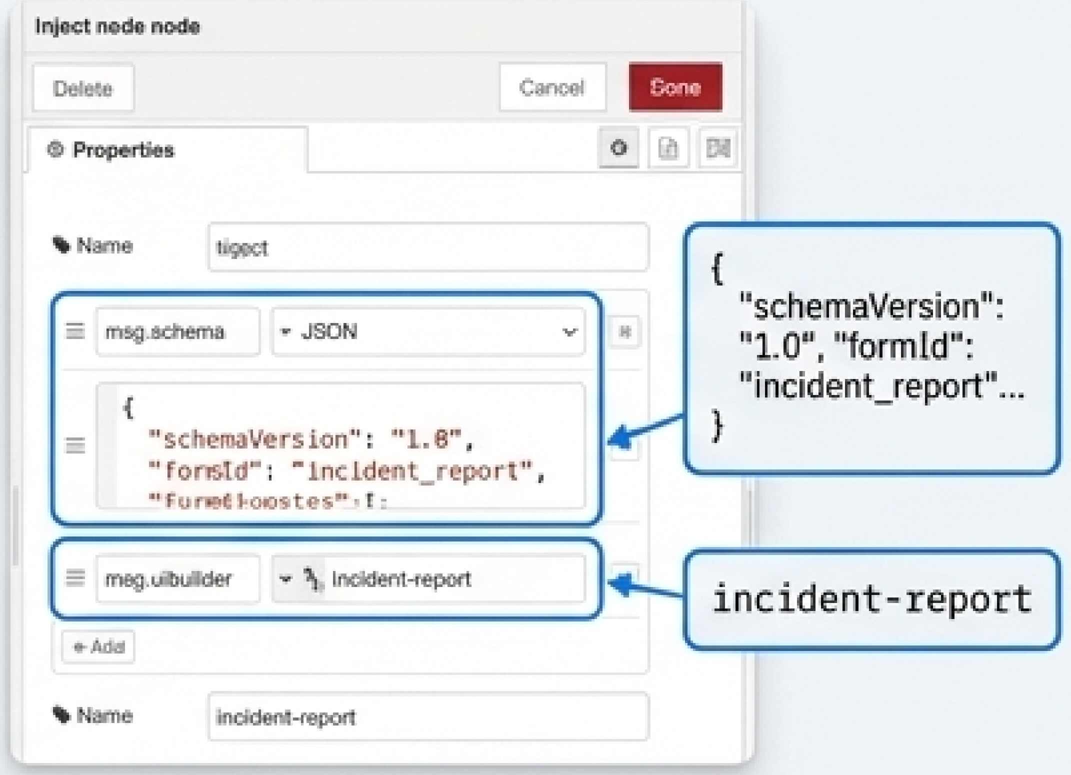
The Why

The `uibuilder-formgen` node expects two key inputs: the form schema itself (`msg.schema`) and the desired URL for the new portal (`msg.uibuilder`). We'll use an `Inject` node to provide this information.

The How

1. **Add Nodes**: On your Node-RED canvas, drag in an `Inject` node, a `uibuilder-formgen` node, and a `Debug` node.
2. **Wire Them**: Connect the output of the `Inject` node to the input of `uibuilder-formgen`, and its output to the `Debug` node's input.
3. **Configure Inject Node**: Double-click the `Inject` node and set two properties:
   – Set `msg.schema` to type **JSON** ({}), and paste the full schema you copied from the builder into the text area.
   – Add a new property. Set `msg.uibuilder` to type **String** (**a–z**), and enter the name for your new portal, e.g., **incident-report**.
4. **Configure Debug Node**: Double-click the `Debug` node and change its output to show the **'complete msg object'**.
5. **Deploy** your changes.





Name: tiigect

msg.schema — JSON

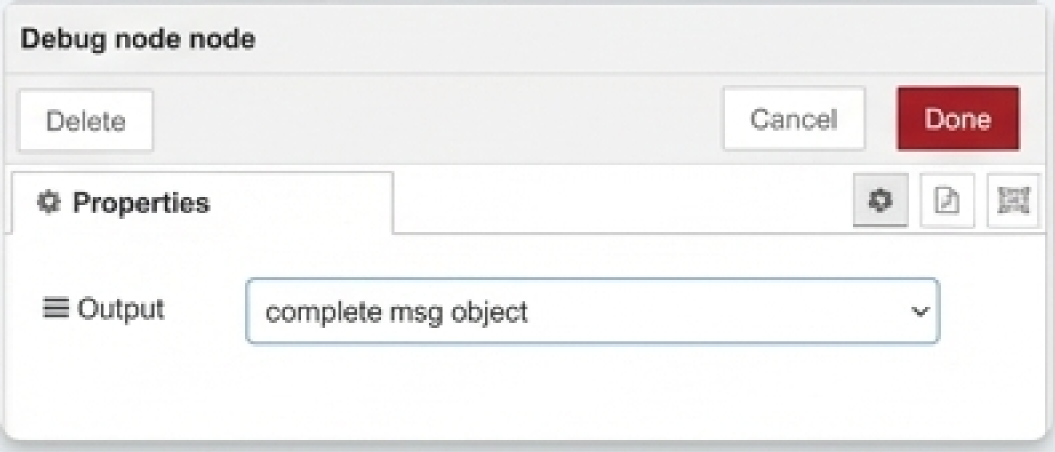
```
{
  "schemaVersion": "1.0",
  "formId": "incident_report",
  "furm€Lkoostes":[:
```

msg.uibuilder — incident-report

Name: incident-report

```
{
  "schemaVersion":
  "1.0", "formId":
  "incident_report"...
}
```

`incident-report`



Output: complete msg object

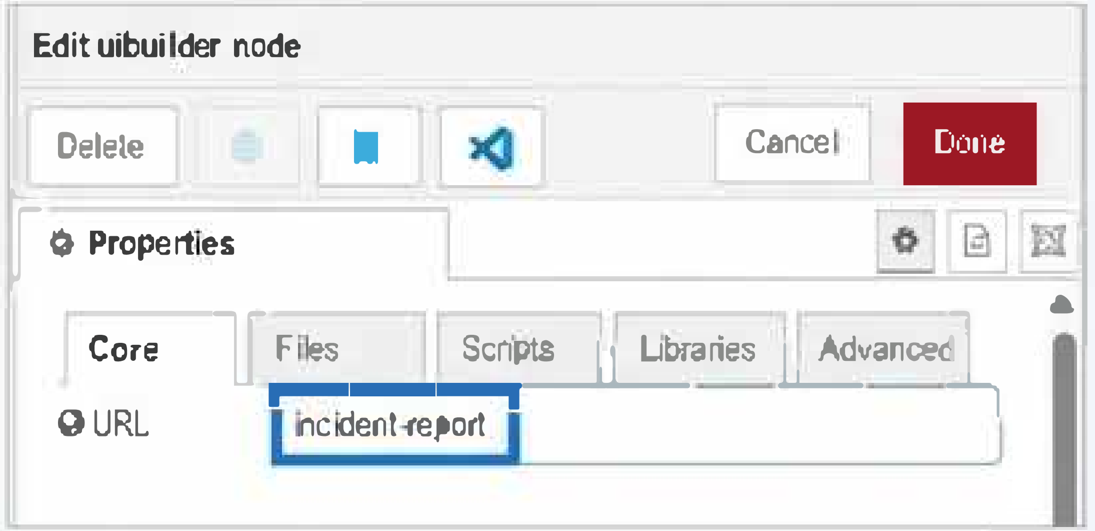
NotebookLM

Step 9: Create the Runtime Portal and Generate

The Why

The `uibuilder-formgen` node generates files but doesn't create the web server endpoint to serve them.

We need to create a second `uibuilder` **instance** that acts as the host for our generated form. The instance name **must** match the name we provided in msg.uibuilder.



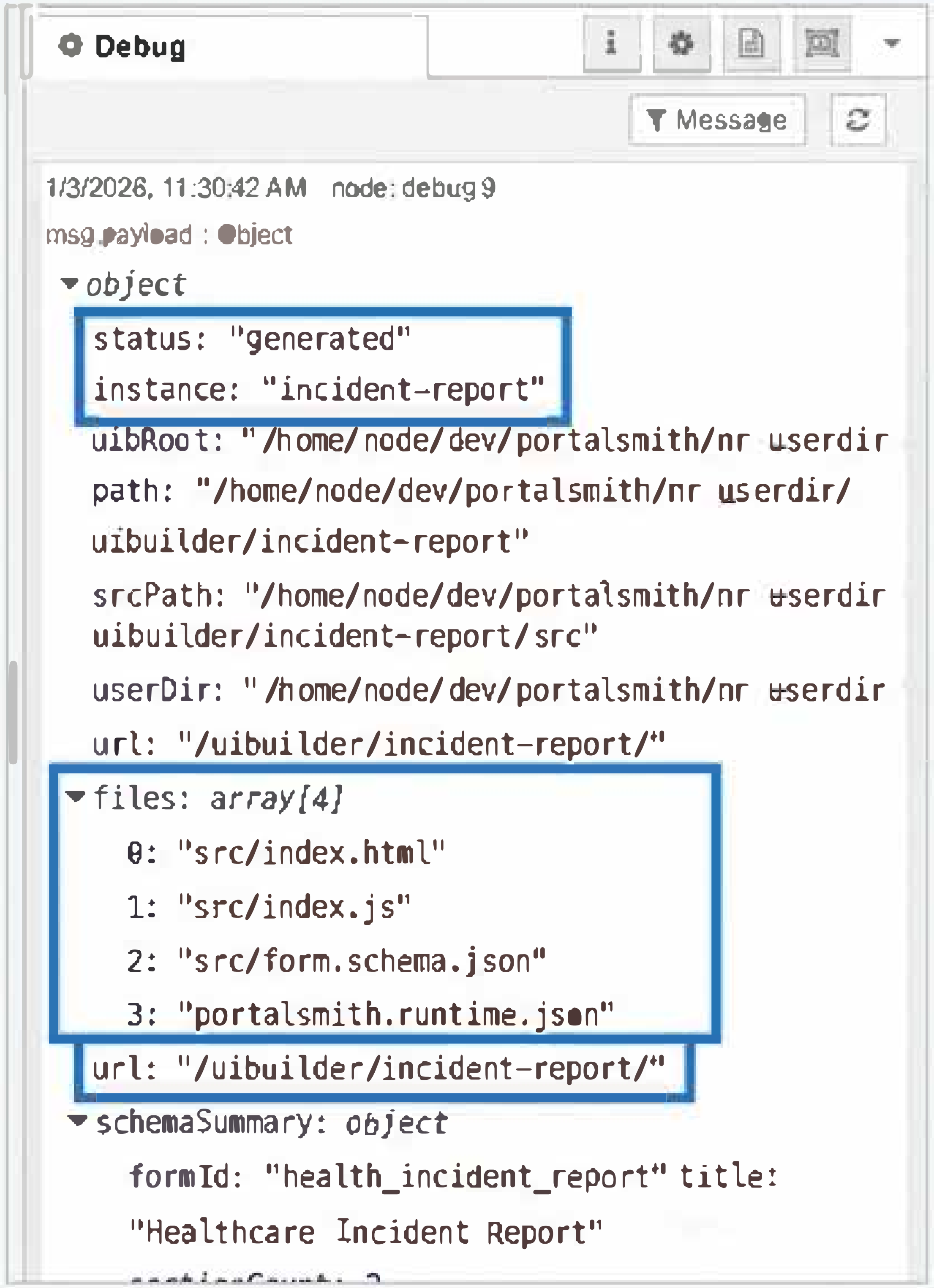
URL: incident-report

The How

1. **Add `uibuilder` Node**
   Drag a new `uibuilder` node onto the canvas.
2. **Set Exact URL**
   Double-click the node. In the URL field, enter the **exact same name** you used in the Inject node: **`incident-report`**.
3. **Deploy**
   Click **Done** and then click the main **Deploy** button. This creates the .../uibuilcident-report/ directory structure.
4. **Generate the Files**
   Now, click the button on the `Inject` node. This triggers the flow. The `uibuilder-formgen` node writes the generated `index.html` and `index.js` files into the `incident-report/src` folder.

Verification

Check the **Debug sidebar**. You will see a message from the `uibuilder-formgen` node confirming success, listing the generated files and the final URL for the portal. This is your proof that it worked.



```
1/3/2026, 11:30:42 AM   node: debug 9
msg.payload : Object
▾object
  status: "generated"
  instance: "incident-report"
  uibRoot: "/home/node/dev/portalsmith/nr userdir
  path: "/home/node/dev/portalsmith/nr userdir/
  uibuilder/incident-report"
  srcPath: "/home/node/dev/portalsmith/nr userdir
  uibuilder/incident-report/src"
  userDir: "/home/node/dev/portalsmith/nr userdir
  url: "/uibuilder/incident-report/"
 ▾files: array[4]
    0: "src/index.html"
    1: "src/index.js"
    2: "src/form.schema.json"
    3: "portalsmith.runtime.json"
  url: "/uibuilder/incident-report/"
 ▾schemaSummary: object
    formId: "health_incident_report" title:
    "Healthcare Incident Report"
    sectionCount: 2
```

Step 10: Use Your Generated Form

The Why

The portal is now live. Let's open it and test the interactive features that were automatically generated from our schema, including validation, conditional conditional logic, and draft management.

The How

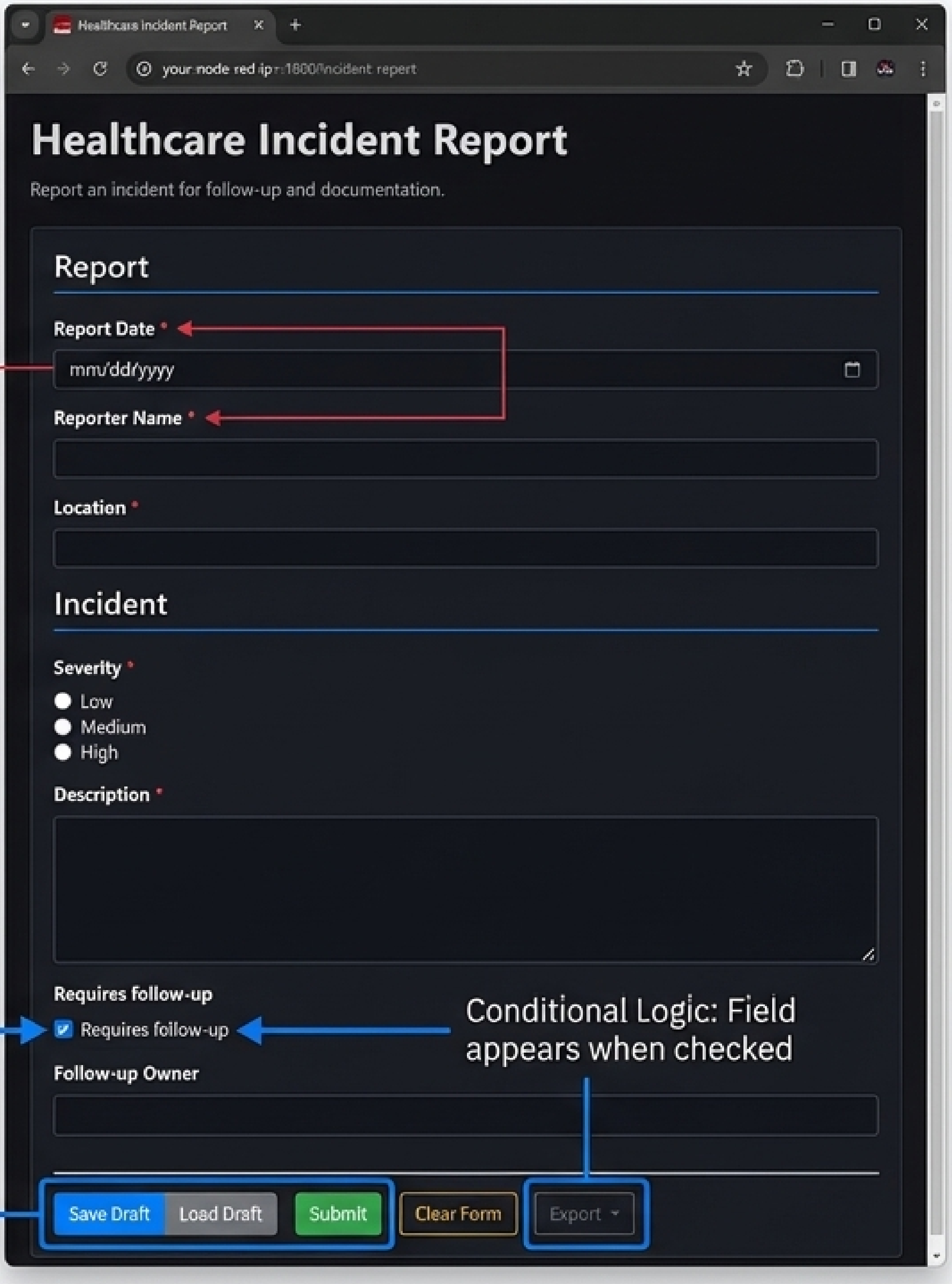
1. **Open Portal:** Double-click the **incident-report uiebuilder** node and click its 'Open' button, or navigate directly to http://<your-node-red-ip>:1880/incident-report.
2. **Interact with the Form:**

   Validation: Notice the red asterisks (*) on required fields. Try to submit without filling them to see the validation messages.

   Conditional Loglc: Check the 'Requires follow-up' box. Notice how the 'Follow-up Owner' field appears dynamically, just as defined in the schema's showIf condition.

   Drafts: Enter some data and click Save Draft. The form data is saved as a JSON file to your computer. Click Clear Form, then Load Draft to restore it.

   Export: Use the Export dropdown to download the current form data as JSON, CSV, or HTML.


Healthcare Incident Report
your node red ip:1800/incident report

Healthcare Incident Report

Report an incident for follow-up and documentation.

Report

Report Date *

mm/dd/yyyy

Reporter Name *

Location *

Incident

Severity *

○ Low
○ Medium
○ High

Description *

Requires follow-up

☑ Requires follow-up

Follow-up Owner

Save Draft
Load Draft
Submit
Clear Form
Export ▾

Required field validation

Conditional Logic: Field appears when checked

Draft Management & Submission Actions

Step 11: Handle Submission and Send Confirmation

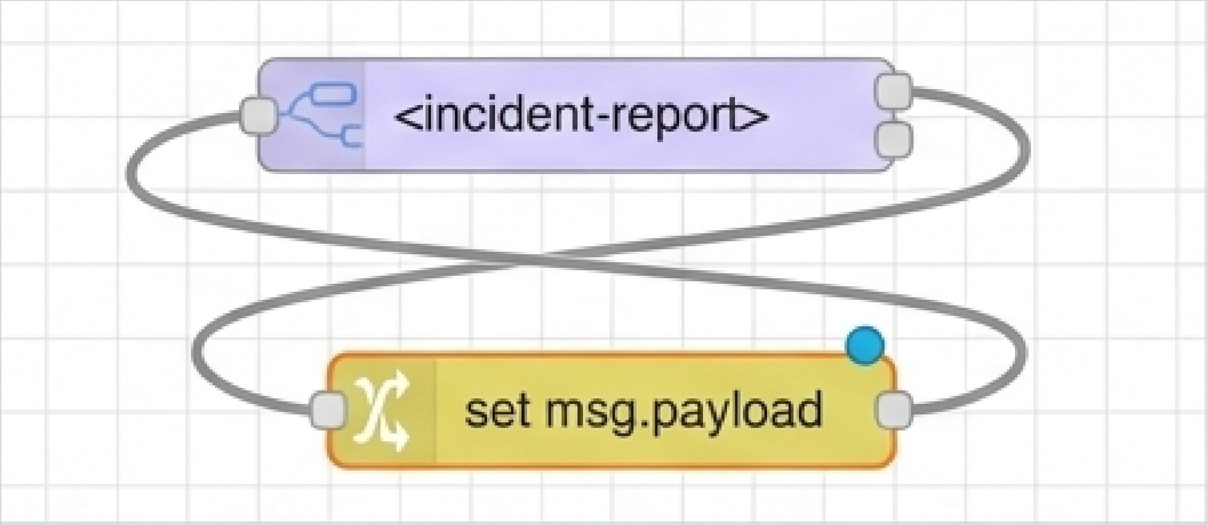
The Why

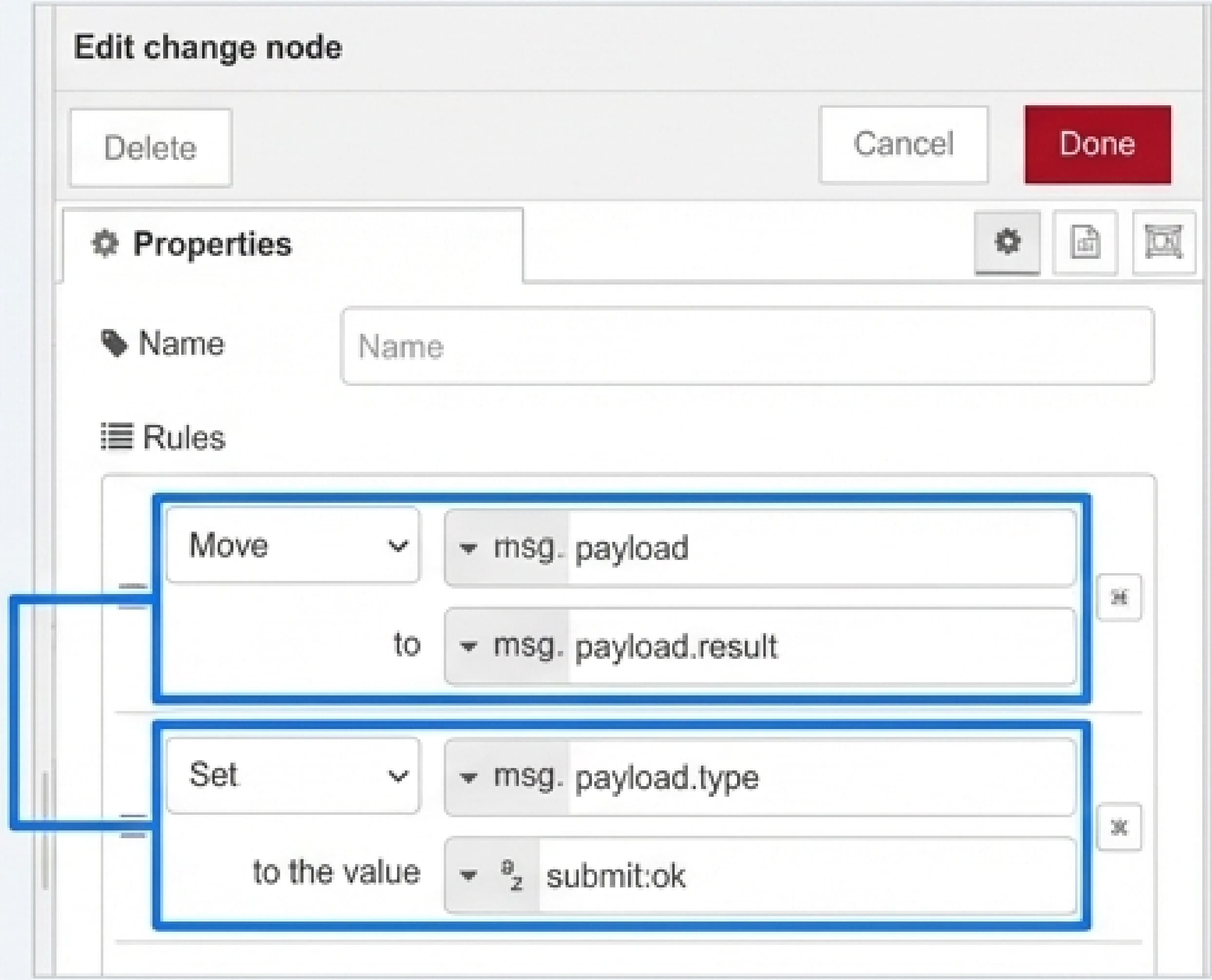
When a user clicks "Submit", the form data is sent from the browser to Node-RED via the `uibuilder` node's output. To show a results page, our flow must process this data and send a specific confirmation message (`submit:ok*k`) back to the UI.

The How

1. **Add a `Change` Node**
   Drag a `Change` node onto the canvas.
2. **Wire the Loop**
   Connect the top output of the <incident-report> uibuilder node to the input of the `Change` node. Then, connect the output of the `Change` node back to the input of the <incident-report> node.
3. **Configure the `Change` Node**
   Set up two rules to structure the confirmation message:
   - **Rule 1**: **Move** msg.payload **to msg.payload.result** (This preserves the original form data).
   - **Rule 2**: **Set** msg.payload.type **to** the string value submit:ok (This is the trigger word for the UI).
4. **Deploy**.

Action: Go back to your form in the browser, fill it out, and click **Submit**. The data will now flow through Node-RED and the confirmation will be sent back.





Name: Name

Rules

Move msg. payload
to msg. payload.result

Set msg. payload.type
to the value submit:ok

NotebookLM

Step 12: View the Results

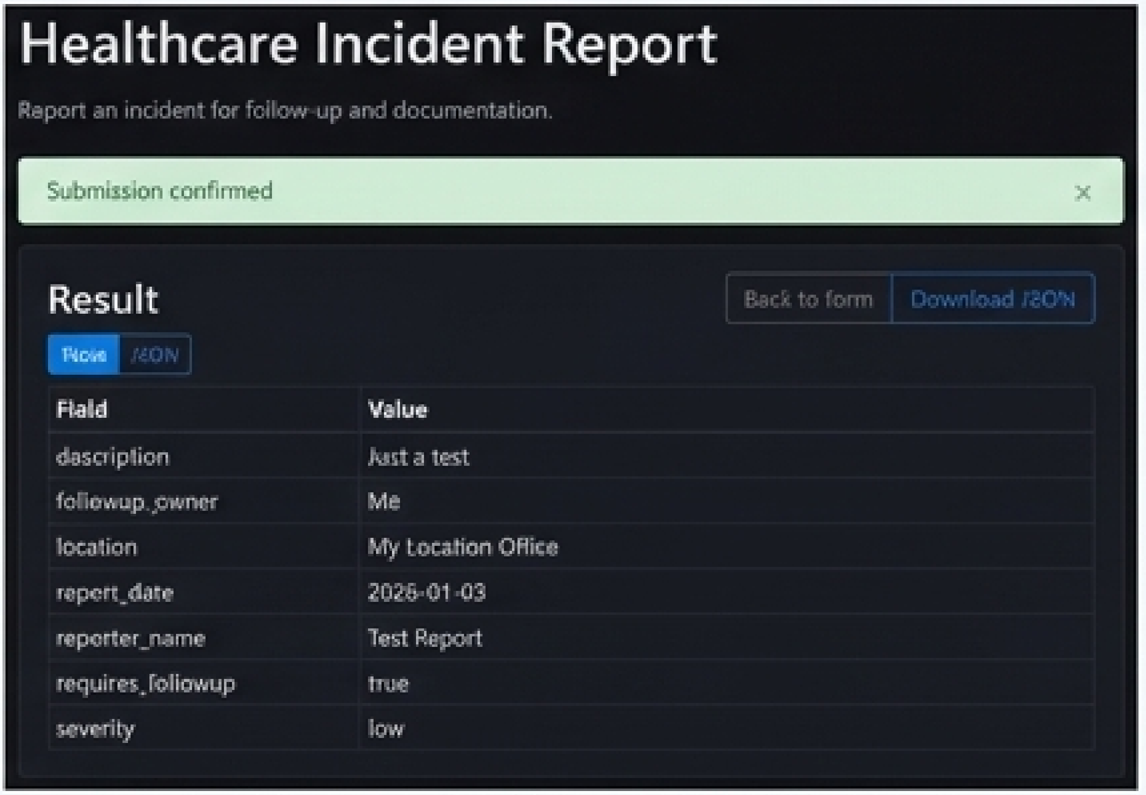
The Why

After receiving the `submit:ok` message, the client-side application automatically hides the form and renders a results page. This provides immediate feedback to the user that their submission was successful and allows them to review the data they sent.

The Result

- A 'Submission confirmed' banner appears at the top.
- The submitted data is displayed in two convenient views, rendered entirely in the browser:
  - **Table View**: A clean, readable key-value table, perfect for quick review.
  - **JSON View**: A formatted block of JSON, ideal for developers or for copying the raw data.
- The user can also **Download JSON** or go **Back to form.**

Submission confirmed



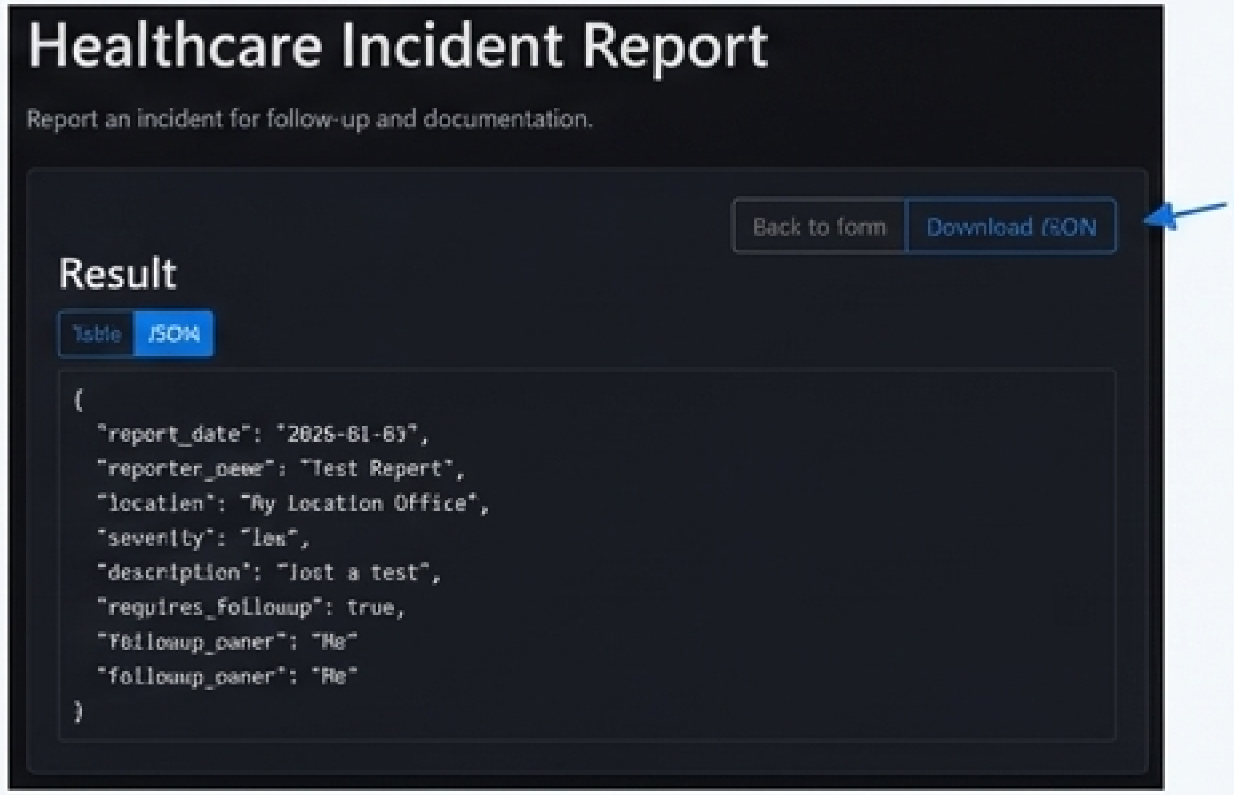
Healthcare Incident Report

Report an incident for follow-up and documentation.

Submission confirmed

Result

Flald	Value
dascription	Just a test
foliowup_owner	Me
location	My Location Office
repert_date	2026-01-03
reporter_name	Test Report
requires_foliowup	true
severity	low



Healthcare Incident Report

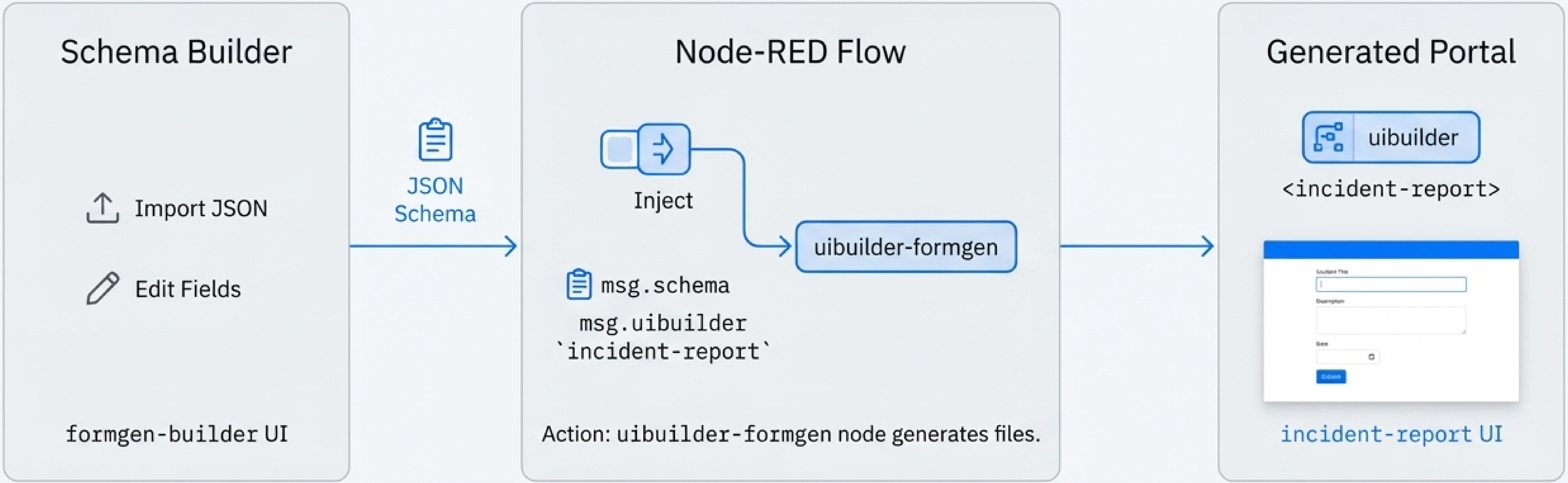
Report an incident for follow-up and documentation.

Result

```
{
  "report_date": "2026-01-03",
  "reporter_oeee": "Test Repert",
  "locatlen": "My Location Office",
  "severlty": "lew",
  "description": "lost a test",
  "requires_followup": true,
  "F0llowup_oaner": "Me"
  "followup_oaner": "Me"
}
```

JSON View

The Complete Workflow at a Glance



Schema Builder

Import JSON

Edit Fields

formgen-builder UI

JSON Schema

Node-RED Flow

Inject

uibuilder-formgen

msg.schema

msg.uibuilder `incident-report`

Action: uibuilder-formgen node generates files.

Generated Portal

uibuilder

<incident-report>

incident-report UI

You design the ‘what’ (the schema) in the Schema Builder.
PortalSmith generates the ‘how’ (the live application) in Node-RED.

Quick Start Recap & Troubleshooting Guide

Key Takeaways

- **Schema is the Contract**
  The JSON schema is the single source of truth for your form's structure, validation, and behavior.
- **Two uibuilder Instances**
  You use one instance (`formgen-builder`) to host the builder UI, and a separate instance for each generated form portal.
- **Names Must Match**
  The portal's `uibuilder` node name must exactly match the value you set in `msg.uibuilder`.
- **"`submit:ok`" is the Trigger**
  To show the results page after a submission, your Node-RED flow must send back a message with \``msg.payload.type` set to \``submit:ok`\`.

Common Troubleshooting Steps

"My generated portal is blank or broken."

Check the Client
Open your browser's developer tools (F12). In the Network tab, check if `/uibuilder/uibuilder.iife.min.js` loaded successfully. If it fails (404), your uibuilder v7 installation has a problem.

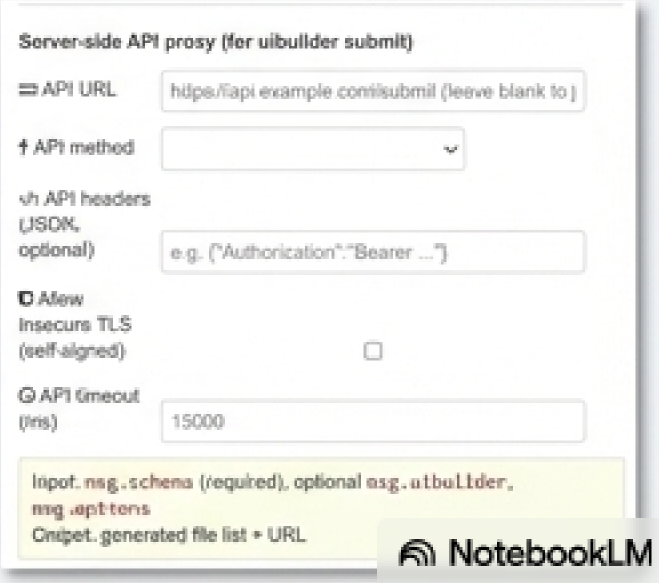
"I submitted the form, but nothing happened."

Check the Wires
Ensure the *top* output of your portal's uibuilder node is wired to your processing flow.

Check the \``submit:ok`\` Message
Use a Debug node to verify the message you are sending back to the portal has the correct `msg.payload.type = 'submit:ok'`\` structure.

"I need to call an API with a self-signed certificate."

Use the Proxy
Do **not** call it from the browser. Configure the **Server-side API proxy** in the uibuilder-formgen node's properties.



Server-side API prosy (fer uibullder submit)

⇄ API URL: hdps//api example conl/submil (leave blank to |

↟ API method: ⌄

‹/› API headers (JSOK, optional): e.g. {"Authorication":"Bearer ..."}

◐ Allow Insecure TLS (self-signed): □

◷ API timeout (ms): 15000

Input: `msg.schema` (required), optional `msg.uibuilder`, `msg.options`
Output: generated file list + URL