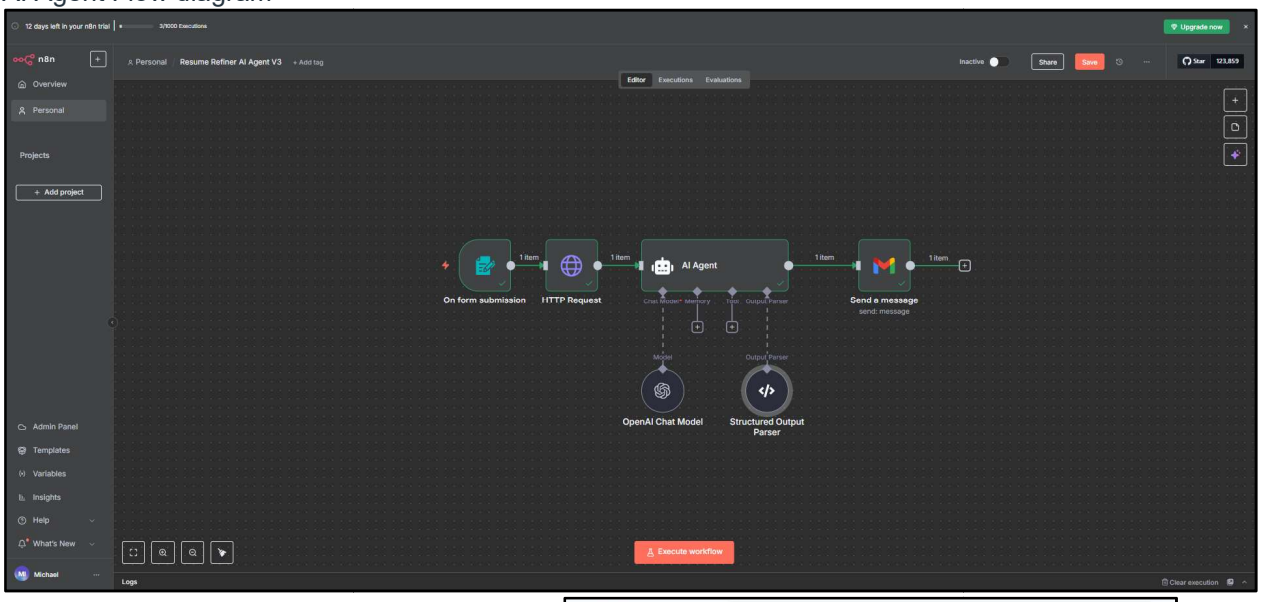
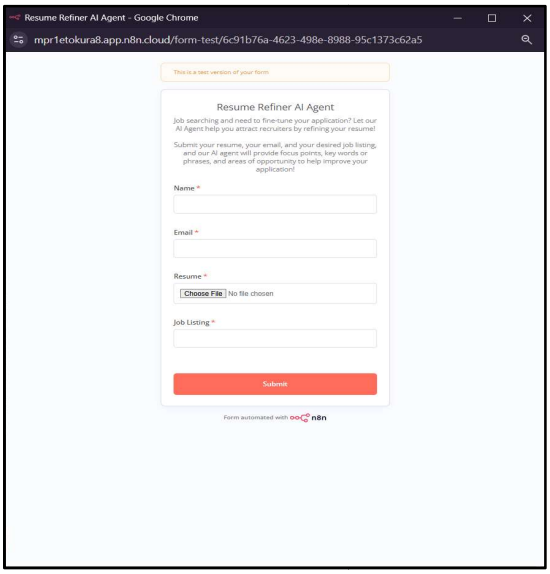
Overview

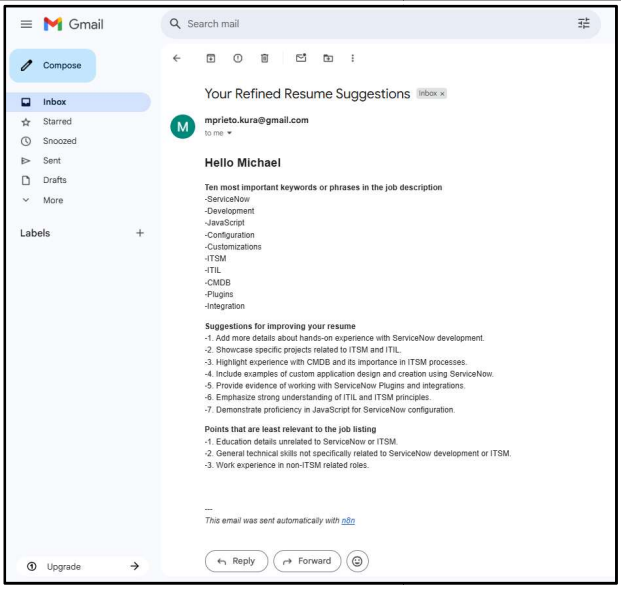
The Resume Refiner AI Agent is a tool that collects a user's resume, as well as an HTTP job listing or description, to provide the user with:

1. A list of the ten most important key words or phrases in the job listing or description
2. A list of seven actionable suggestions that the user can make to their resume to make it more compatible to the job listing or description
3. A list of three least compatible points of data in the resume to improve

I. AI Agent Flow diagram







II. Building the Agent

I. What approach did you take to design your agent?

- I started off by creating the basics first: the initial Form Trigger and the final Send Email action. I created and configured these two nodes with each other first, as a test before getting into the meat and potatoes of the AI Agent. The reason for this was to confirm two absolutely necessary steps: that the form submission did in fact create a trigger form (***input***), and that an email was correctly sent as a response to the form trigger (***output*****)**. A simple test email confirmed initial setup- These had to be reconfigured later.
- For the trigger, I created a form submission node.
  Authentication: None [we want this form to be openly accessible to anyone]
  Form Title: Resume Refiner AI Agent
  Form Description: [display text]
  Form Elements [all required]:
    Name [Text field to be referenced in the email message opener]
    Email [To be referenced in the final send email node]
    Resume [File Submission referenced by the AI agent, accepted formats: .doc,.docx,.pdf,.odt,.txt]
    Job Listing [Text field referenced by AI Agent]
- For the Email, I connected my Gmail account as the source.
    Operation was to send
    Send To: The email on the trigger form. I used the schema to pull the JavaScript data from the Trigger Form to make this field dynamic for anyone accessing it. (Will be changed later)
    Subject: [Text]
    Email type: HTML
    Message: I pulled the JavaScript data for the name field to make a dynamic intro that said “Hello [name]” (will be changed later)
- Once these nodes were created, I began connecting the AI agent:
    I created a HTTP Request node to GET HTTP data from the previous node dynamically with JavaScript.
- This information will be fed to the following node: AI Agent
  The AI agent followed the HTTP Request. This node was connected to OpenAI as the LLM and given a basic prompt:
    “You are a job coach who reads resumes and makes suggestions to improve compatibility with job listings. Your job is to read the job description in {{ $('On form submission').item.json['Job Listing'] }} and return a list of the ten most important keywords or phrases in the job description.

    You will also read {{ $('On form submission').item.json.Resume.filename }} and suggest actionable improvements for seven of the most relevant points that resonate with the most important keywords or phrases.
    You will also suggest three points in the resume that are the least relevant to the job listing.
    Provide JSON output.”
- The data provided by the AI agent was compiled by pulling from the form submission through JavaScript. The resulting data was very spot on and was provided in JSON format.
- In order to integrate the response into the email node in HTML, a structured Output Parser was used and given a basic sample code:
    {
    "Ten most important keywords or phrases in the job description" : ["1,2,3,4,5,6,7,8,9,10"],
    "Suggestions for improving MPrieto_ResumeTemplate.pdf" : ["1,2,3,4,5,6,7"],
    "Points that are least relevant to the job listing" : ["1,2,3"]
    }
- Once converted into separate data lists, I reconfigured the email node to display the “Hello [name]” header, followed by the HTML output as follows:
    <h2>Hello {{ $('On form submission').item.json.Name }}</h2>
    <p>Ten most important keywords or phrases in the job description
-{{ $json.output['Ten most important keywords or phrases in the job description'][0] }}
-{{ $json.output['Ten most important keywords or phrases in the job description'][1] }}
-{{

```
$json.output['Ten most important keywords or phrases in the job description'][2] }}<br>-{{
$json.output['Ten most important keywords or phrases in the job description'][3] }}<br>-{{
$json.output['Ten most important keywords or phrases in the job description'][4] }}<br>-{{
$json.output['Ten most important keywords or phrases in the job description'][5] }}<br>-{{
$json.output['Ten most important keywords or phrases in the job description'][6] }}<br>-{{
$json.output['Ten most important keywords or phrases in the job description'][7] }}<br>-{{
$json.output['Ten most important keywords or phrases in the job description'][8] }}<br>-{{
$json.output['Ten most important keywords or phrases in the job description'][9] }}</p>

<p><strong>Suggestions for improving your resume</strong><br>-{{ $json.output['Suggestions
for improving MPrieto_ResumeTemplate.pdf'][0] }}<br>-{{ $json.output['Suggestions for
improving MPrieto_ResumeTemplate.pdf'][1] }}<br>-{{ $json.output['Suggestions for improving
MPrieto_ResumeTemplate.pdf'][2] }}<br>-{{ $json.output['Suggestions for improving
MPrieto_ResumeTemplate.pdf'][3] }}<br>-{{ $json.output['Suggestions for improving
MPrieto_ResumeTemplate.pdf'][4] }}<br>-{{ $json.output['Suggestions for improving
MPrieto_ResumeTemplate.pdf'][5] }}<br>-{{ $json.output['Suggestions for improving
MPrieto_ResumeTemplate.pdf'][6] }}</p>

<p><strong>Points that are least relevant to the job listing</strong><br>-{{ $json.output['Points
that are least relevant to the job listing'][0] }}<br>-{{ $json.output['Points that are least relevant
to the job listing'][1] }}<br>-{{ $json.output['Points that are least relevant to the job listing'][2]
}}</p>
```

- Once connected, I executed the workflow and received the appropriate email result.

I. What challenges did you face in parsing, formatting, or integrating?

Connecting nodes and using schemas to reference and populate dynamic JavaScript fields was fairly easy. My biggest challenge, however, was in parsing the AI Agent's JSON output into HTML for the Email Sender to compile properly. I didn't have a very strong coding background going into the project, and wasn't able to create sample scripts for the Output Parser or for the HTML of the body. Instead, found myself learning to code these on the fly by trial and error. I understand general syntax, and the parser did a very good job of highlighting line errors that I could decode, but it took the majority of the time I spent on the project. Originally, the first version of my project used an AI Transform node that was given the prompt to transform JSON output to HTML, but I scrapped the idea; I didn't want to give up on coding this part because the AI Agent's JSON output was practically perfect in format, with headers and line breaks. Ultimately, I managed to solve the issue, and learned far more coding than I knew prior to the project :)

II. How did you ensure that the AI returned JSON reliably?

I wish I had worked a bit more on this part. I was lucky to have my AI Agent return well formatted JSON results consistently, and I honestly thought it was a normal thing to happen. I didn't have any instances where I couldn't decode or understand the JSON results. On the contrary, the AI agent organized these into ordered lists far better than I expected. Perhaps my prompt set it up for reliable JSON outputs. In the end, I only asked it to return the restuls in JSON format, which it was already doing in the first place.

II. Troubleshooting

I. What issues did you encounter and how did you resolve them?

My biggest issue after solving the JSON/HTML issue, was receiving consistent results from the AI Agent. There would be times where the Agent returned very strong results that pulled data from the prompts and did its job perfectly. Other times, it would wander and hallucinate points that did not exist in the

provided resume or the html source. After several iterations of prompts, it gave more consistently strong answers, albeit in lesser words (i.e. less explanations of why it chose certain options compared to before). In favor of consistency, I left the shorter answers in, but I would definitely have liked to revisit the issue.

III. Optimization

I. What might you improve or add in future iterations?

I think that, in future iterations of the project, I'd address the AI Agent's consistency issue by perhaps adding a link to the end of the sent email that allows the user to retrigger the AI Agent and receive another email, for however many times the user wanted/needed. It wouldn't solve the issue of consistency, but perhaps it could help the user clarify and narrow down on the AI Agent's responses. Of course, this would just be a bandaid to the consistency issue, so perhaps I could give it better and stronger guard rails to provide stronger answers.