🛠️ Problem Explanation

Job hunting is time-consuming and repetitive. Candidates need to:

- Search job portals daily
- Parse long job descriptions
- Check if the role matches their skills
- Write customized cover letters
- Send applications with proper contact details

This workflow automates the **entire process** – from **fetching jobs → extracting details → scoring → generating cover letters → sending emails → storing results in Google Sheets**.

🔄 Workflow Breakdown (Node by Node)

1. **Schedule Trigger**
    - Runs daily at **10:00 AM** to fetch new jobs automatically.
2. **RSS Read Node**
    - Reads **LinkedIn jobs RSS feed** for the given keyword/region.
3. **Loop Over Items Node**
    - Iterates over each job posting in the RSS feed.
4. **HTTP Request Node**
    - Fetches the **full LinkedIn job page** for details.
5. **Google Gemini Model 1 – Job Extractor**

- **System prompt:** “You are an intelligent bot capable of pulling out data from a job listing site.”

User prompt: Extracts job info into JSON with fields:

```
{
  "Company Name": "",
  "Job role": "",
  "Benefits": "",
  "Job Description": "",
  "Location": "",
  "Contact": "",
  "Company domain": "",
  "Contact Email": ""
}
```

-

6. **JavaScript Code Node**
    - Removes unnecessary **nesting** from LLM JSON output.
7. **IF Node**
    - Filters out jobs **without contact person details**.
8. **Google Gemini Model 2 – Job Match Scoring**
    - **System prompt:** “You are an intelligent bot, rating how closely a job listing is to a candidate skill set.”
    - **Scoring Rules:**
        - +3 points: Skills match
        - +1 point: Mostly matching skills
        - +1 point: Right experience level
        - +1 point: Remote job role
        - +1 point: Past experience matches

        - +1 point: Resume skills match
        - +3 points: Candidate meets qualifications

Output JSON:

```
{ "Score": "" }
```

    9. **JavaScript Code Node**
        - Flattens Gemini scoring output.
    10. **Google Gemini Model 3 – Cover Letter Generator**
        - **System prompt:** “You are perfect at creating cover letters for a job.”
        - **User prompt:** Uses candidate resume + job details.

Output JSON:

```
{ "Cover letter": "" }
```

    11. **JavaScript Code Node**
        - Cleans cover letter output for structured saving.
    12. **Google Sheets Node**
        - Appends final job details + score + cover letter into a **Google Sheet tracker**.
    13. **Gmail Node**
        - Sends **customized application email** with cover letter to the recruiter contact.

🔑 API & Prompt Usage

- **RSS Feed API** – Fetches job listings from LinkedIn.
- **HTTP Request** – Scrapes LinkedIn job page details.
- **Google Gemini API (LLM)**
    - *Model 1*: Job extractor – converts raw job descriptions into structured JSON.
    - *Model 2*: Scoring – matches candidate resume with job description.
    - *Model 3*: Cover letter – generates professional, customized cover letters.
- **Google Sheets API** – Stores structured job data, score, and cover letter.
- **Gmail API** – Automates sending job application emails.

⚡ Challenges & Solutions

1. API Rate Limits

- **Problem:** RSS feeds & Gemini API limit number of requests.
- **Solution:** Used **Loop + Split In Batches** nodes to process jobs gradually and avoid throttling.

2. Text Parsing & Nesting Issues

- **Problem:** Gemini often returns **nested JSON** inside strings.
- **Solution:** Added **JavaScript Code nodes** to clean & flatten outputs.

3. Missing Contact Information

- **Problem:** Many job postings don’t list recruiter names.
- **Solution:** Added **IF Node filter** to only continue with jobs that have a contact person.

4. Email Accuracy

- **Problem:** LLM sometimes generated invalid emails.
- **Solution:** Generated emails only when both **contact name + company domain** were present.

📚 Summary of Learnings

- **n8n** is powerful for chaining APIs, AI models, and workflows.
- **LLMs like Gemini** are excellent for extracting structured data, but need **post-processing (JS nodes)**.
- **Automating job search** saves time and ensures consistency in applications.
- **Rate limits & data quality** are real bottlenecks; batching + validation filters are crucial.
- Building this pipeline taught how to integrate **data fetching** → **LLM parsing** → **filtering** → **scoring** → **document generation** → **storage** → **notification** into one cohesive workflow.