Chapter 4-3: 보석 세공하기 (Gemini 응답 파싱 및 Notion 저장)

Prologue: 혼돈 속에서 질서를 창조하다

Gemini가 우리에게 보내준 JSON 형식의 텍스트는 보석의 원석과 같습니다. 그 안에는 '요약', '필수 스킬', '우대 스킬'이라는 귀중한 정보가 들어있지만, 아직은 하나의 거대한 텍스트 덩어리일 뿐입니다. 이 원석을 그대로 Notion의 '텍스트' 칸에 넣어두는 것은 너무나 아까운 일입니다.

이번 챕터에서는 이 텍스트 원석을 정밀하게 세공하여, **'AI 요약'은 '텍스트' 속성에, '핵심 기술'은 '멀티 선택' 태그로, '요구 경력'은 '선택' 속성에** 각각 알맞게 담아내는 '데이터 파싱(Parsing)' 기술을 구현할 것입니다.

이 미션을 완료하면, 신호용 님의 Notion 데이터베이스는 단순한 데이터 창고를 넘어, **한눈에 모든 핵심 정보를 파악할 수 있는 강력한 '대시보드'**로 변모하게 될 것입니다.

1. 학습 목표 (Objectives)

- JSON 형식의 문자열을 Python의 딕셔너리(Dictionary) 객체로 변환할 수 있다.
- `json.loads()` 사용 시 발생할 수 있는 예외를 처리하는 안정적인 코드를 작성할 수 있다.
- Notion API의 `multi_select`와 `select` 속성 데이터 형식을 이해하고, 그에 맞게 데이터를 가공하여 저장할 수 있다.

2. 핵심 개념 (Core Concepts)

2.1 JSON 파싱 (Parsing): 문자열을 살아있는 객체로

- **What:** `"{"key": "value"}"` 처럼 생긴, 따옴표로 감싸진 **문자열(string)**을, 파이썬에서 `data['key']` 처럼 자유롭게 다룰 수 있는 **딕셔너리(dictionary)** 객체로 변환하는 과정입니다. 파이썬의 `json` 라이브러리가 이 역할을 합니다.
- **Why:** Gemini가 보내준 텍스트는 아직 단순한 문자열 덩어리입니다. 우리는 이것을 파싱하여 `summary`, `required_skills` 등의 키로 각 부분에 접근해야만, 원하는 정보를 Notion의 각기 다른 속성에 나눠 담을 수 있습니다.
- **How:** `json.loads(json_string)` 함수를 사용합니다.

2.2 Notion의 데이터 타입: `multi_select`와 `select`

- `multi_select` **(멀티 선택):** 여러 개의 태그를 동시에 가질 수 있는 속성입니다. (e.g., 'Python', 'Java', 'Spring')
  - **API 형식:** `[{'name': 'Python'}, {'name': 'Java'}]` 처럼, 각 태그의 이름을 `name` 키에 담은 딕셔너리들의 **리스트** 형태로 보내야 합니다.
- `select` **(선택):** 여러 옵션 중 단 하나의 값만 가질 수 있는 속성입니다. (e.g., '신입', '경력', '무관')
  - **API 형식:** `{'name': '경력'}` 처럼, 선택할 옵션의 이름을 `name` 키에 담은 **단일 딕셔너리** 형태로 보내야 합니다.

[Mission 19] Gemini 분석 결과 파싱 및 Notion 저장

Step 1: Notion 데이터베이스에 '그릇' 최종 준비

1. Notion 데이터베이스에 아래 속성들이 준비되었는지 다시 한번 확인합니다. (이전 미션에서 이미 만드셨을 겁니다.)
   - `AI 요약` (유형: `텍스트`)
   - `핵심 기술` (유형: `멀티 선택`)
   - `요구 경력` (유형: `선택`) - "신입", "경력", "경력 무관" 등의 옵션을 미리 만들어두면 좋습니다.
2. **`main.py`의 Notion 저장 로직을 대대적으로 업그레이드합니다.** `for` 루프 안의 `notion.pages.create` 부분을 아래 코드로 교체해주세요.

Step 2: `gemini_analyzer.py`의 응답 형식 정리

Gemini가 보내주는 텍스트에 포함된 ```` ```json ... ``` ```` 와 같은 마크다운 형식을 제거하여, 순수한 JSON 문자열만 남도록 함수를 수정합니다.

```
# analysis/gemini_analyzer.py (수정)
import json # json 라이브러리 임포트

# ...
def analyze_job_posting(job_description: str):
    # ... (prompt 내용은 동일)
    try:
        response = model.generate_content(prompt)
        # --- Gemini 응답 텍스트 정제 ---
        cleaned_text = response.text.strip().replace("```json",
"").replace("```", "")
        return json.loads(cleaned_text) # 텍스트가 아닌, 파싱된 dict 객체를 반
환!
    except (json.JSONDecodeError, Exception) as e:
        print(f"  [Gemini 오류] 응답 파싱 중 문제 발생: {e}")
        print(f"  원본 응답: {response.text}")
        return None # 파싱 실패 시 None 반환
```

Step 3: `main.py`의 최종 플레이팅 (Notion 저장)

```
# main.py (Notion 저장 루프 수정)

import json # main.py에도 json 임포트

# ...
for i, job in enumerate(all_jobs):
    # ... (중복 확인 로직) ...

    # --- '신규' 공고일 경우 ---
    # ...
    job_description = job.get('description', "")
    analysis_result = None # 분석 결과를 담을 변수 (dict 또는 None)
```

```
    if job_description:
        analysis_result = analyze_job_posting(job_description)
        if analysis_result:
            print("--- Gemini 분석 완료 ---")
        else:
            print("--- Gemini 분석 실패 ---")
    else:
        print("  - 본문 내용이 없어 Gemini 분석을 건너뜁니다.")

    # --- 최종 Notion 저장 ---
    company = job['company']
    source = job['source']

    # properties 딕셔너리를 동적으로 구성
    properties_to_save = {
        '직무': {'title': [{'text': {'content': title}}]},
        '회사명': {'rich_text': [{'text': {'content': company}}]},
        '링크': {'url': link},
        '출처': {'rich_text': [{'text': {'content': source}}]},
        '수집일': {'date': {'start': collection_date}}
    }

    # Gemini 분석 결과가 있을 경우에만, 해당 속성들을 추가
    if analysis_result:
        summary = analysis_result.get('summary', '요약 정보 없음')
        required_skills = analysis_result.get('required_skills', [])
        # (나중에 프롬프트를 수정하여 'career' 필드도 추가할 수 있습니다)
        # career = analysis_result.get('career', '정보 없음')

        properties_to_save['AI 요약'] = {'rich_text': [{'text': {'content':
summary}}]}

        # multi_select 형식에 맞게 데이터 가공
        # Notion은 100개가 넘는 태그는 저장할 수 없으므로, 최대 100개로 제한
        if required_skills:
            properties_to_save['핵심 기술'] = {'multi_select': [{'name':
skill} for skill in required_skills[:100]]}

        # (나중에 '요구 경력'도 추가)
        # if career:
        #     properties_to_save['요구 경력'] = {'select': {'name': career}}

    try:
        notion.pages.create(
            parent={"database_id": NOTION_DATABASE_ID},
            properties=properties_to_save # 동적으로 구성된 properties 전달
        )
        success_count += 1

    except Exception as e:
        print(f"  🔥 [오류 발생] '{title}' 저장 실패! 원인: {e}")
# ...
```

다음 행동 계획 (Next Action)

1. ☐ Notion DB에 `AI 요약(텍스트)`, `핵심 기술(멀티 선택)` 속성을 추가합니다.
2. ☐ `gemini_analyzer.py`와 `main.py`를 위 최종 코드로 업데이트합니다.
3. ☐ DB를 비우고(선택), `main.py`를 실행합니다.
4. ☐ **최종 결과 확인:** Notion 데이터베이스에 새로운 공고가 추가되고, **'AI 요약' 칸에는 Gemini가 생성한 요약문이, '핵심 기술' 칸에는 여러 개의 태그가 아름답게 생성되는지** 확인합니다.