Chapter 3-6: '착한 크롤러' 되기 (안전한 크롤링 전략)

Prologue: 투명 망토를 벗고, 정중한 방문객 되기

지금까지 우리 크롤러는 투명 망토를 쓴 침입자와 같았습니다. User-Agent를 설정했지만, 그 외에는 신원을 거의 밝히지 않았죠. 이제 우리는 투명 망토를 벗고, "안녕하세요, 신호용 님의 학습용 프로젝트에서 데이터 수집을 위해 방문했습니다. 서버에 부담이 되지 않도록 천천히 둘러보겠습니다." 라고 말하는 정중한 방문객이 될 것입니다.

이번 챕터에서는 **HTTP 헤더(Headers)**와 **요청 간격(Request Interval)**을 고도화하여, 우리 크롤러를 훨씬 더 안정적이고, 윤리적이며, 프로페셔널하게 만들 것입니다.

1. 학습 목표 (Objectives)

- HTTP 헤더의 역할과 User-Agent의 중요성을 깊이 이해한다.
- `requests.Session` 객체를 사용하여 모든 요청에 일관된 헤더를 적용할 수 있다.
- 고정된 `time.sleep` 대신, `random.uniform`을 사용하여 인간과 유사한 불규칙한 요청 간격을 만들 수 있다.

2. 핵심 개념 (Core Concepts)

2.1 정중한 신원 밝히기: 상세한 헤더(Headers) 구성

- **What:** 헤더는 브라우저가 서버에게 보내는 '자기소개서'입니다. User-Agent 외에도 "저는 한국어를 선호합니다(`Accept-Language`)", "압축된 파일도 받을 수 있습니다(`Accept-Encoding`)" 등 다양한 정보가 담겨있습니다.
- **Why:** 상세한 헤더를 추가하면, 우리 크롤러는 "수상한 스크립트"가 아닌, **"표준적인 웹 브라우저"**처럼 보이게 됩니다. 이는 정교한 봇 차단 시스템에 의해 탐지될 확률을 크게 낮춰줍니다.
- **How:** `requests` 라이브러리를 직접 사용할 때 더욱 효과적입니다. Selenium은 드라이버 레벨에서 헤더를 제어하지만, 우리는 `BaseCrawler`를 `requests` 기반으로 바꾸는 리팩토링을 통해 이 개념을 완벽하게 적용할 수 있습니다. (이번 미션에서는 우선 **Selenium 구조 내에서 할 수 있는 최선**인, 요청 간격 제어에 집중하겠습니다.)

2.2 서버 배려하기: 인간적인 요청 간격 (Randomized Delay)

- **What:** `time.sleep(3)`처럼 항상 똑같은 시간만큼 기다리는 것이 아니라, `random.uniform(2, 5)`처럼 **2초에서 5초 사이의 임의의 시간**만큼 기다리는 것입니다.
- **Why:** 서버 관리자 입장에서, **정확히 3.000초 간격으로 들어오는 요청**은 기계가 보낸다는 100% 확실한 증거입니다. 반면 2.7초, 4.1초, 3.5초 등 불규칙하게 들어오는 요청은 실제 사용자의 행동 패턴과 훨씬 유사하여 탐지를 피하는 데 유리합니다. 또한 서버에 순간적인 부하가 몰리는 것을 방지하는 효과도 있습니다.
- **How:** 파이썬의 `random` 라이브러리를 `import`하고, `time.sleep()` 안에 `random.uniform(최소시간, 최대시간)`을 넣어주기만 하면 됩니다.

[Mission 14] 크롤러에 '배려심' 탑재하기

이 중요한 개념들을 우리 시스템의 심장부인 `BaseCrawler`에 적용하여, 모든 자식 크롤러들이 이 '예절'을 상속받도록 하겠습니다.

Step 1: `base_crawler.py`에 랜덤 슬립 기능 추가

모든 크롤러가 공통으로 사용할 수 있도록, `BaseCrawler`에 랜덤한 시간만큼 대기하는 헬퍼(helper) 메서드를 만듭니다.

```
# crawlers/base_crawler.py (수정)

import time
import random # <- 1. random 라이브러리 임포트
from abc import ABC, abstractmethod
# ... (다른 import 구문) ...

class BaseCrawler(ABC):
    def __init__(self, base_url, delay_range=(2, 5)): # <- 2. delay_range 파라미터 추가
        self.base_url = base_url
        self.delay_range = delay_range # <- 3. 딜레이 범위를 인스턴스 변수로 저장
        self.driver = self._setup_driver()

    def _random_sleep(self):
        """설정된 딜레이 범위 내에서 랜덤한 시간만큼 대기합니다."""
        delay = random.uniform(self.delay_range[0], self.delay_range[1])
        # print(f"  (Polite Delay: {delay:.2f} seconds)") # 디버깅 시 주석 해제
        time.sleep(delay)

    # ... (_setup_driver, crawl, close_driver 메서드는 동일) ...
```

Step 2: 자식 크롤러들이 `_random_sleep`을 사용하도록 수정

이제 각 크롤러의 `time.sleep(3)` 부분을, 우리가 만든 `self._random_sleep()`으로 교체하기만 하면 됩니다.

```
# crawlers/jobkorea_crawler.py (수정 예시)

# ...
    def crawl(self, keyword: str = '백엔드', pages_to_crawl: int = 3):
        # ...
        for page in range(1, pages_to_crawl + 1):
            search_url = f"{self.base_url}/Search/?stext={keyword}&Page_No={page}"
            self.driver.get(search_url)
            self._random_sleep() # <- time.sleep(3) 대신 사용

            print(f"  - {page} 페이지 처리 중...")
            # ...
```

```
# crawlers/wanted_crawler.py (수정 예시)

# ...
    def crawl(self, keyword: str = "백엔드", pages_to_crawl: int = 1):
        # ...
        target_url = f"{self.base_url}/search?query={keyword}&tab=position"
        self.driver.get(target_url)
        self._random_sleep() # <- time.sleep(3) 대신 사용

        # ... (while 루프) ...
        while True:
            body.send_keys(Keys.END)
            self._random_sleep() # <- time.sleep(3) 대신 사용
            # ...
```

최종 결론 및 다음 단계

신호용 님의 제안 덕분에, 우리 프로젝트는 이제 기술적 완성도를 넘어 **운영 안정성과 윤리적 책임감**까지 고려하는 한 단계 더 성숙한 시스템으로 발전했습니다. 이 부분은 면접관에게 어필할 수 있는 매우 강력한 차별화 포인트가 될 것입니다.

다음 행동 계획 (Next Action):

1. ☐ `base_crawler.py`와 모든 자식 크롤러(`wanted_crawler.py`, `jobkorea_crawler.py`, `saramin_crawler.py`)에 위 '안전한 크롤링 전략'을 적용하여 코드를 수정합니다.
2. ☐ 수정한 코드가 정상적으로 동작하는지 `main.py`를 실행하여 최종 테스트합니다.
3. ☐ 이 작업이 완료되면, 우리는 **Phase 3의 모든 목표를 완벽하게 달성**하고, 드디어 **Phase 4: 지능형 기능 및 API 서버 구축**으로 나아갈 준비를 마치게 됩니다.

이 중요한 개선 작업을 마친 후 알려주십시오. 다음 단계가 우리를 기다리고 있습니다.