2. 다음 프로젝트 교재 (Phase 3)

이제 우리의 시스템을 더욱 견고하고 확장 가능하게 만드는 Phase 3의 첫 번째 미션을 위한 교재입니다.

Chapter 3-1: 확장 가능한 크롤러 아키텍처 설계

Prologue: 레고 블록처럼 조립하는 크롤러

지금까지 우리의 크롤링 로직은 `main.py` 파일 안에 모두 들어있었습니다. 이는 원티드라는 단 하나의 사이트만 상대할 때는 괜찮았지만, 잡코리아, 사람인 등 새로운 사이트를 추가하려면 `main.py`가 점점 더 복잡해지고 유지보수가 어려워지는 구조입니다.

이번 챕터에서는 **객체 지향 프로그래밍(OOP)**의 핵심인 **추상화(Abstraction)**와 **상속(Inheritance)** 개념을 도입하여, 어떤 채용 사이트든 '레고 블록'처럼 쉽게 갈아 끼울 수 있는 유연하고 확장 가능한 크롤러 구조를 설계할 것입니다. 이 경험은 신호용 님의 코드를 한 단계 더 성숙시켜 줄 것입니다.

1. 학습 목표 (Objectives)

- 추상 클래스(Abstract Class)의 개념과 필요성을 이해한다.
- 공통 기능을 가진 부모 클래스(`BaseCrawler`)를 정의하고, 사이트별 특성을 가진 자식 클래스(`WantedCrawler`)가 이를 상속받도록 구현할 수 있다.
- 기존의 절차 지향적 코드를 객체 지향적으로 리팩토링할 수 있다.

2. 핵심 개념 (Core Concepts)

- **추상화 (Abstraction):** 여러 크롤러들의 공통된 기능(e.g., 페이지 접속, HTML 소스 가져오기)은 남겨두고, 각 사이트마다 달라지는 세부 내용(e.g., 채용 공고 태그 찾기)은 비워두는 설계 방식입니다. 우리는 "모든 크롤러는 반드시 '데이터 추출' 기능을 가져야 한다"는 **'규칙' 또는 '설계도'**를 만들 것입니다.
- **상속 (Inheritance):** '설계도'(`BaseCrawler`)에 정의된 공통 기능을 자식 클래스(`WantedCrawler`, `JobKoreaCrawler` 등)가 그대로 물려받아 사용하고, 비워진 세부 내용만 각자 구현하도록 하는 방식입니다. 이를 통해 코드 중복을 획기적으로 줄일 수 있습니다.

[Mission 9] 크롤러 모듈화 및 추상화

아래 단계를 따라 기존 코드를 새로운 구조로 리팩토링해봅시다.

> **Step 1: 파일 구조 변경**
>
> `web-crowler` 폴더 안에 `crawlers` 라는 새 폴더를 만듭니다. 이 폴더는 앞으로 우리가 만들 모든 크롤러 모듈을 담는 공간이 될 것입니다. 그리고 그 안에 `__init__.py` (빈 파일), `base_crawler.py`, `wanted_crawler.py` 파일을 생성합니다.

```
web-crowler/
├── crawlers/
│   ├── __init__.py
│   ├── base_crawler.py  # 크롤러 설계도
```

```
│   └── wanted_crawler.py  # 원티드 전용 크롤러
├── main.py
└── ...
```

Step 2: `base_crawler.py` **작성 (설계도 만들기)**

모든 크롤러의 공통 부모가 될 추상 클래스를 정의합니다.

```python
# crawlers/base_crawler.py

from abc import ABC, abstractmethod
from selenium import webdriver
from selenium.webdriver.chrome.service import Service as ChromeService
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.chrome.options import Options

class BaseCrawler(ABC):
    """모든 크롤러가 상속받아야 하는 추상 기본 클래스"""

    def __init__(self, base_url):
        self.base_url = base_url
        self.driver = self._setup_driver()

    def _setup_driver(self):
        """Selenium WebDriver를 설정하고 반환합니다."""
        chrome_options = Options()
        chrome_options.add_argument("--headless")
        user_agent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36"
        chrome_options.add_argument(f"user-agent={user_agent}")
        chrome_options.add_argument("--no-sandbox")
        chrome_options.add_argument("--disable-dev-shm-usage")
        service =
ChromeService(executable_path=ChromeDriverManager().install())
        driver = webdriver.Chrome(service=service, options=chrome_options)
        return driver

    @abstractmethod
    def crawl(self):
        """
        크롤링 프로세스를 시작하는 메인 메소드.
        이 메소드는 자식 클래스에서 반드시 구현해야 합니다.
        """
        pass

    def close_driver(self):
        """드라이버를 안전하게 종료합니다."""
        if self.driver:
            self.driver.quit()
```

Step 3: `wanted_crawler.py` **작성 (원티드 크롤러 구현)**

`BaseCrawler`를 상속받아 원티드에 특화된 로직을 구현합니다.

```
# crawlers/wanted_crawler.py

import time
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from bs4 import BeautifulSoup
from .base_crawler import BaseCrawler

class WantedCrawler(BaseCrawler):
    """원티드 사이트 크롤러"""

    def __init__(self):
        super().__init__("https://www.wanted.co.kr/")

    def crawl(self, keyword='백엔드'):
        """원티드 채용 정보를 크롤링하여 리스트로 반환합니다."""
        print("원티드 크롤링을 시작합니다...")
        self.driver.get(self.base_url)
        time.sleep(2)

        # 기존 main.py에 있던 원티드 크롤링 로직을 여기에 넣습니다.
        # 검색 버튼 클릭 -> 키워드 입력 -> 스크롤 -> 데이터 추출
        # (자세한 코드는 기존 main.py를 참고하여 채워주세요)

        # ... (생략) ...

        # 데이터 추출 후, 가공하여 list of dict 형태로 반환
        job_data = []
        html = self.driver.page_source
        soup = BeautifulSoup(html, 'lxml')
        job_cards = soup.select("div[role='listitem'] a")

        for card in job_cards:
            title = card.select_one("strong[class*='JobCard_title']").text
            company_name =
card.select_one("span[class*='CompanyName']").text
            link = "https://www.wanted.co.kr" + card['href']
            job_data.append({"title": title, "company": company_name,
"link": link, "source": "Wanted"})

        print(f"원티드에서 총 {len(job_data)}개의 공고를 찾았습니다.")
        return job_data
```

Step 4: `main.py` **리팩토링 (간결해진 메인)**

이제 `main.py`는 각 크롤러를 '호출'하고, 그 결과를 받아 Notion에 저장하는 '조율자(Orchestrator)' 역할만 수행하게 되어 훨씬 깔끔해집니다.

```python
# main.py (수정 후 예시)

import os
from dotenv import load_dotenv
import notion_client
from crawlers.wanted_crawler import WantedCrawler # WantedCrawler를 임포트

load_dotenv()

# 1. Notion API 설정 (기존과 동일)
# ...

# 2. 크롤러 실행
wanted_crawler = WantedCrawler()
all_jobs = []
try:
    wanted_jobs = wanted_crawler.crawl(keyword='백엔드')
    all_jobs.extend(wanted_jobs)
finally:
    # 크롤링이 성공하든 실패하든 드라이버는 항상 종료되도록 finally 구문 사용
    wanted_crawler.close_driver()

# 여기에 나중에 JobKoreaCrawler, SaraminCrawler 등을 추가할 수 있습니다.
# jobkorea_crawler = JobKoreaCrawler()
# all_jobs.extend(jobkorea_crawler.crawl())


# 3. Notion에 저장 (기존 로직과 유사)
print(f"총 {len(all_jobs)}개의 채용 공고를 찾았습니다. Notion에 저장을 시작합니다...")
for job in all_jobs:
    # Notion DB에 페이지 생성 로직
    # ...

print("모든 작업이 완료되었습니다.")
```
