

Chapter 4-2 (업그레이드): 동적 상세 페이지 정복

Prologue: 숨겨진 정보를 펼치는 열쇠

단순히 페이지에 접속하는 것을 넘어, 이제 우리는 페이지와 '상호작용'하는 방법을 배웁니다. 사용자가 '더보기' 버튼을 클릭하는 행동을 Selenium으로 흉내 내어, 숨겨진 모든 정보를 드러나게 만드는 것은 동적 웹사이트 크롤링의 핵심 기술입니다. 이번 챕터에서 우리는 그 '열쇠'를 손에 쥐게 될 것입니다.

1. 학습 목표 (Objectives)

- Selenium의 `click()` 메서드를 사용하여 특정 버튼을 클릭하는 동작을 자동화할 수 있다.
- '더보기' 버튼이 없을 수도 있는 예외 상황에 대처하는 안정적인 코드를 작성할 수 있다.
- 복잡한 HTML 구조 속에서 원하는 정보를 모두 포함하는 최상위 컨테이너를 정확히 찾아낼 수 있다.

[Mission 17 업그레이드] WantedCrawler 상세 정보 수집 기능 완성

Step 1: wanted_crawler.py의 get_job_description 메서드 교체

기존의 `get_job_description` 메서드를 아래의 새로운 코드로 완전히 교체해주세요. 신호용 님의 분석이 코드에 어떻게 반영되었는지 집중해서 봐주세요.

```
# crawlers/wanted_crawler.py (get_job_description 메서드 업그레이드)

# ... (import 구문 및 다른 메서드는 동일) ...

def get_job_description(self, url: str) -> str:
```

```
"""
원티드 상세 페이지에 방문하여, '상세 정보 더 보기' 버튼을 클릭하고,
전체 본문 내용을 수집합니다.
"""
try:
    self.driver.get(url)
    self._random_sleep()

    # --- 1. '상세 정보 더 보기' 버튼 클릭 로직 (수정) ---
    try:
        # 신호용 님이 분석한 정확한 구조: '상세 정보 더 보기' 텍스트를 가진 span을
        # 포함한 button
        more_button_xpath = "//button[span[text()='상세 정보 더 보기']]"
        more_button = self.driver.find_element(By.XPATH, more_button_xpath)

        # JavaScript 클릭을 사용하여 다른 요소에 가려져 있어도 클릭되도록 함
        self.driver.execute_script("arguments[0].click();", more_button)
```

```

        print("    -> '상세 정보 더 보기' 버튼을 클릭했습니다.")
        self._random_sleep()
    except Exception:
        print("    -> '상세 정보 더 보기' 버튼이 없거나 클릭할 수 없습니다. 그대로
진행합니다.")
        pass
    # -----

    soup = BeautifulSoup(self.driver.page_source, 'lxml')

    # --- 2. 전체 본문을 감싸는 article 태그 찾기 ---
    content_article =
soup.select_one('article[class*="JobDescription_JobDescription"]')

    if content_article:
        return content_article.text.strip()
    else:
        # ... (2차 방어 코드 - 이전과 동일) ...
        return ""

except Exception as e:
    print(f" 🚨 [상세 정보 수집 오류] {url} 처리 중 문제 발생: {e}")
    return ""


```

코드 심층 분석 (Why & How)

- **more_button.click()**: 신호용 님이 발견하신 대로, 숨겨진 내용을 펼치기 위해 Selenium의 **click** 기능을 사용했습니다.
- **By.XPATH, "//*[@contains(text(), '더보기')]"**: '더보기' 버튼의 클래스 이름이 불안정하기 때문에, 훨씬 더 강력하고 안정적인 **XPath 선택자**를 사용했습니다. 이 코드의 의미는 "HTML 문서 전체(//)에서, 어떤 태그든(*) 상관없이, 텍스트 내용에 '더보기'라는 단어를 포함(**contains(text(), '더보기')**)하는 첫 번째 요소를 찾아라" 입니다. 이 방법은 버튼의 디자인이 아무리 바뀌어도 텍스트만 그대로라면 계속 동작합니다.
- **try-except ... pass**: '더보기' 버튼을 감싼 **try-except**는 매우 중요합니다. 만약 공고 내용이 짧아서 '더보기' 버튼이 아예 없는 경우, **find_element**는 오류를 발생시킵니다. 이때 **except** 블록이 오류를 '잡아서' 아무것도 하지 않고(**pass**) 그냥 넘어가도록 하여, 프로그램이 멈추는 것을 방지합니다.
- **select_one('article[class*="JobDescription_JobDescription"]')**: 신호용 님이 찾아낸 **JobDescription** 클래스를 '부분 일치' 방식으로 사용하여, 뒤에 붙는 해시값이 바뀌더라도 계속 동작하도록 만들었습니다.

다음 행동 계획 (Next Action)

1. ☐ **wanted_crawler.py**의 **get_job_description** 메서드를 위 최종 업그레이드 코드로 교체합니다.
2. ☐ **main.py**의 테스트 범위 제한 로직(**jobs_to_process = crawled_jobs_full_list[:3]**)이 잘 설정되어 있는지 다시 한번 확인합니다.

3.  `main.py`를 실행하여, **Gemini 분석 결과**에 이전과 달리 '주요업무', '자격요건' 등의 내용이 포함된, 훨씬 더 풍부하고 정확한 요약이 출력되는지 확인합니다.