# Py Review App Final Report

소속 : 정보컴퓨터공학부 인공지능전공
이름 : 최재원 (2025)

---

### Background

최근에 동기들이랑 문제를 풀고 코드 리뷰를 하는 식으로 파이썬 스터디를 진행하였다. 다만 매주 정해진 시간에 리뷰를 진행하다보니 피드백이 지연됨으로써 효율이 떨어진다는 생각이 들었다. 실력을 늘리는데 있어서 중요한 요소가 빠른 피드백이라고 생각하였기 때문에 이러한 문제점을 최근에 발전하고 있는 LLM을 활용하여 해결 할 수 있지 않을까라는 생각이 들었다. 따라서 이러한 코드 리뷰를 자동으로 해주는 웹 앱을 만들고자 한다.

### Problem Specification

입력으로 문제와 파이썬 코드가 주어지고 이를 여러 도구를 활용하여 리뷰 한 후에 그 결과를 출력으로 내보내야 한다.

### Design

- 웹 서버
  - FastAPI를 사용한다.
- 데이터 베이스
  - 유저 정보와 유저가 실행한 코드 리뷰 데이터 로그를 저장한다.
  - 데이터베이스는 매우 단순한 nosql/json 기반의 tinydb를 사용한다.
- 로그인 및 회원가입 기능
  - 세션을 통해 로그인을 유지한다.
- 코드 리뷰 (linter, formatter)
  - ruff 라는 도구는 cli 전용이기 때문에 subprocess를 띄워서 처리한 후 결과를 받는다.
- 코드 리뷰 (LLM)
  - OpenAI의 o1-mini 모델을 활용하여 미리 작성한 문제 리뷰용 프롬프트와 사용자가 입력한 값들을 조합하여 전달한다.
- 프론트 엔드
  - 최대한 간단하게 하기 위해서 tailwindcss와 daisyui 같은 css/ui 라이브러리를 사용한다
  - 각 페이지 당 하나의 html만 가지도록해 최대한 단순하게 설계한다.

### Implementation

먼저 프로젝트 디렉터리 구조는 다음과 같다.

```
.
├── LICENSE
├── README.md
├── app
│   ├── auth_helper.py
│   ├── config_helper.py
│   ├── db_helper.py
│   ├── front_helper.py
│   ├── main.py
│   └── tools.py
├── db.json
├── doc
│   ├── DFD for pyreview-app.drawio
│   ├── DFD for pyreview-app.jpg
│   ├── final_report.typ
│   ├── proposal.pdf
│   └── proposal.typ
├── frontend
│   └── html
│       ├── main.html
│       ├── signin.html
│       └── signup.html
├── pyproject.toml
└── uv.lock
```

유저의 프론트엔드 부분은 모두 `/frontend` 에 넣었고, 실제 서버 로직과 관련된 파이썬 코드들은 모두 `/app` 하위에 두었다. 기본적으로 서버 관련 로직들은 main.py에 있는 router들이 auth_helper, db_helper와 같은 모듈들을 참고하게 함으로써 계층적으로 설계하였다.

실제 구현된 화면은 다음과 같다.

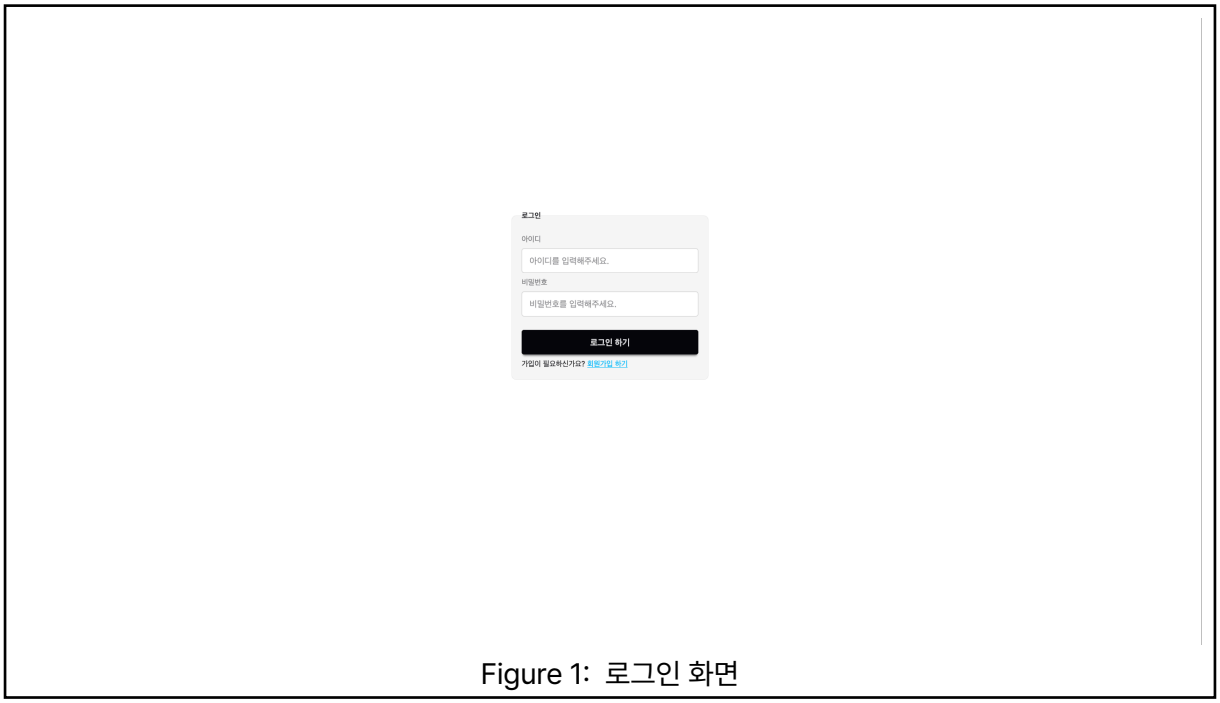


Figure 1: 로그인 화면

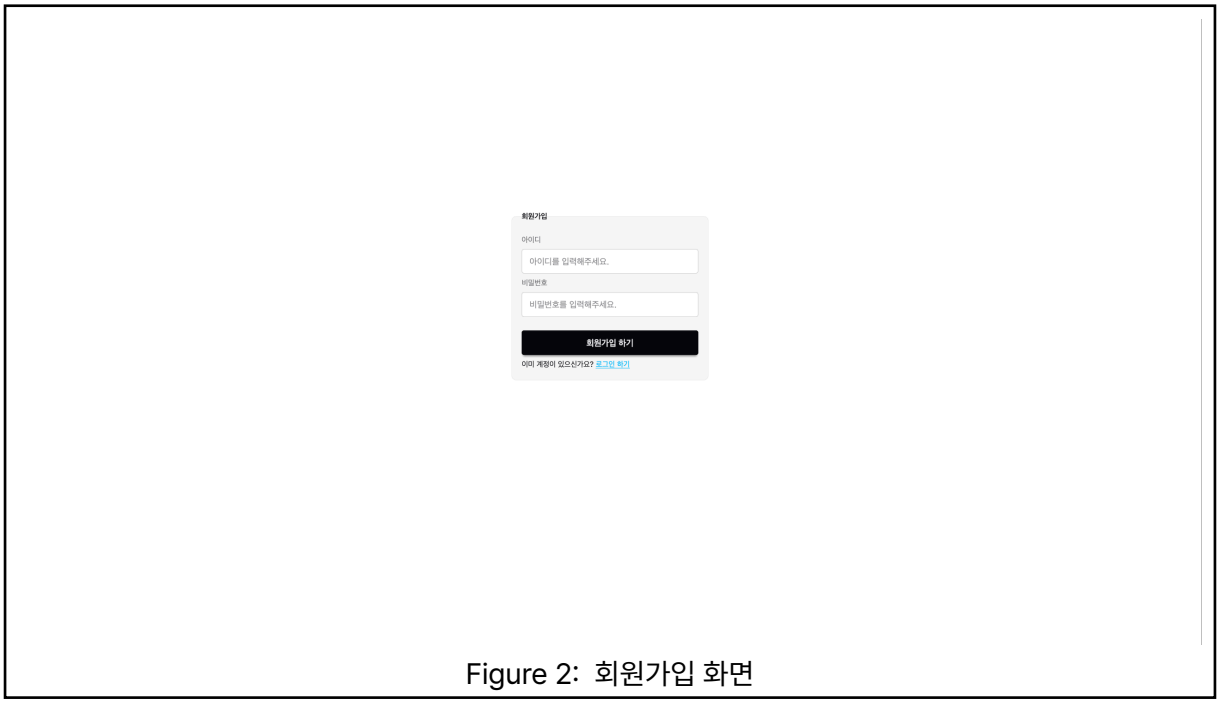


Figure 2: 회원가입 화면

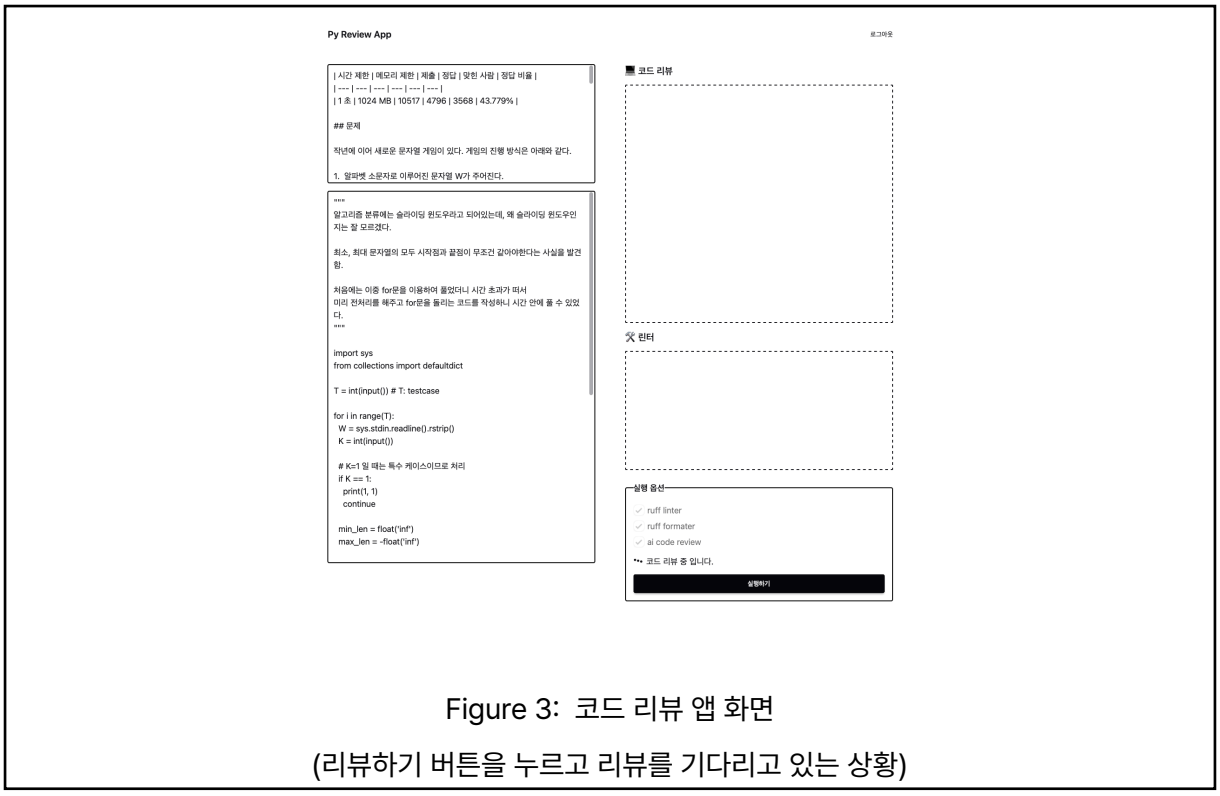


Figure 3: 코드 리뷰 앱 화면

(리뷰하기 버튼을 누르고 리뷰를 기다리고 있는 상황)

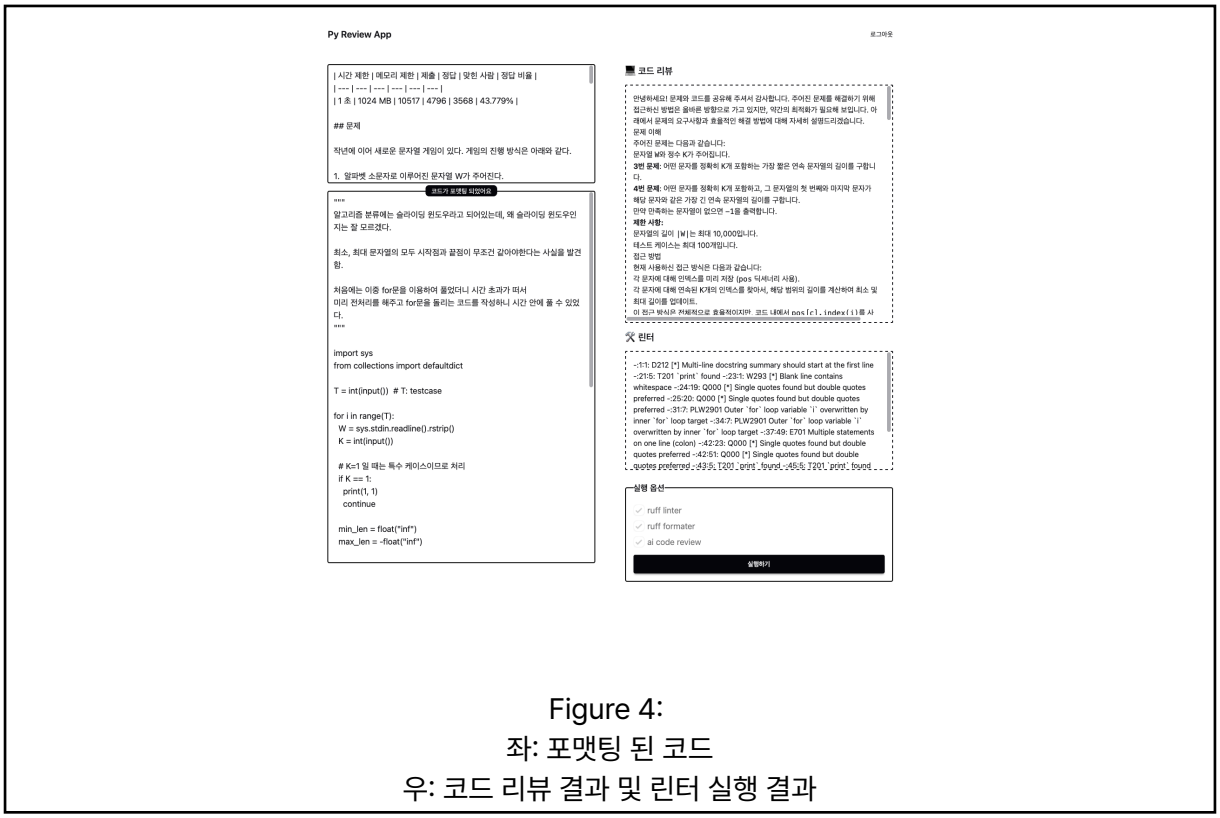


Figure 4:
좌: 포맷팅 된 코드
우: 코드 리뷰 결과 및 린터 실행 결과

**Analysis & Discussion**

코드를 작성하면서 생각했던 점들에 대해 설명해보겠다.

- 데이터베이스와 ORM
  - 처음에는 ORM을 사용할려고 했었다. 다만 여러 ORM을 공식 문서를 보고 내 코드에 적용해보면서 느낀 점은 ORM을 도입함으로써 얻는 이득보다 비용이 더 많이 든다는 사실을 알게되었다.

    - 따라서 이러한 문제를 해결하기 위해서 tinydb라는 json을 기반으로 하는 파이썬 기반의 nosql 데이터베이스를 사용하기로 결정하였다. 다만 ORM이 주는 편리함 또한 사용하고 싶었기때 데이터를 검증하는 Pydantic을 활용하여 service code ↔ pydantic model ↔ no_sql_data 같은 구조로 작성하였고, 그 결과 편리하면서도 원하는 기능을 빠르게 구현할 수 있었다.
    - 다만 tinydb가 단일 스레드에서 동작함으로 멀티 프로세싱, 멀티 스레딩 환경에서는 쓸 수 없다. 파이썬의 경우에는 서버를 열 때 프로세스를 여러 개를 작동시키기도 하는데 주의해야한다.
- 프론트 엔드
    - 개인적인 목표는 프론트 엔드는 최대한 단순하게 유지하면서 백엔드의 로직 부분에 집중하는 것이었다. 그러기 위해서는 리액트 같은 무거운 도구들을 쓰기보다는 오직 각 페이지 당 하나의 html 파일만을 가지도록 구성하였다. 또한 ui 부분 또한 단순성을 유지하고 싶어서 tailwind css와 daisyui를 활용하였다.
- 인증 구현 (로그인, 로그아웃, 회원가입 등)
    - 유저 정보를 입력하고 회원가입을 하면 패스워드는 자동으로 pbkdf2_sha256 알고리즘을 통해 해싱되게 구현하였다. 해싱 또한 여러가지 알고리즘이 있는데 최근 gpu의 발전 등으로 bcrypt와 같은 알고리즘보다는 pbkdf2_sha256를 쓰는게 안전하다. 또한 각 패스워드마다 salt 값을 부여함으로써 레인보우 테이블 공격을 방어하였다.
    - 해싱 알고리즘이 cpu-bound이기 때문에 이는 서버가 작동하는데 병목으로 동작할 수 있다.따라서 이를 해결하기 위해서 따로 계산만을 처리하는 스레드를 만든 후 계산을 위임하고 결과는 async/await를 활용하여 비동기적으로 받는 코드를 필요시 작성해야한다. 다만 현재 내 앱은 그 정도 사용자 양을 목적으로 하는 것이 아니기 때문에 이는 구현하지 않았다.
    - 인증 구현하는데에는 여러 방식이 있지만 나는 그 중에서도 session을 선택하였다. 이 또한 session store를 메모리에 저장하기 때문에 멀티 스레드, 멀티 프로세스 환경에서는 쓸 수 없다. 필요 시 redis 같은 인메모리 캐시 디비를 사용해야한다.
- 제한
    - LLM은 토큰 당 비용이 들기 때문에 회원가입 한 사용자는 관리자의 인증을 받도록, 인증 받은 사용자는 하루에 5번, 각 문제와 코드의 글자 수는 1500자 이하로 제한을 걸었다.

**Conclusion**

파이썬 코드 리뷰 기능을 가진 웹앱을 구현해보았다. 이러한 코드를 작성하면서 파이썬으로 웹 서버를 작성하는 방법과 값을 검증하는 Pydantic에 대해서도 알게되었다. 특히 모듈을 분리하고 설계하는 방식에 대해서도 고민을 많이 했는데 잘 설계 된 것 같아서 만족스럽다.

이때 까지는 파이썬으로 프로그램을 작성하기보다는 PS 문제를 푸는 용도로만 썼는데, 이번에 처음으로 웹앱을 만들어보면서 FastAPI, Pydantic과 같은 모듈들을 사용하게 되었다. 이러한 모듈이 가진 기능에 대해 배워보면서 파이썬 생태계가 정말 견고하다는 생각이 들었다. 개인적으로 예전에는 파이썬이라는 언어를 그렇게 좋아하지 않았는데 대학에 들어와서 파이썬에 대해 몰랐던 부분도 알게되고 다양한 모듈들을 배우면서 파이썬이라는 언어에 대해 매력을 알게 된 것 같다.

**Source Code**

github.com/cjaewon/pyreview-app