프로젝트 리뷰 보고서

작성자: DeepSeek API
작성일자: 2025-01-22

1. 리뷰 데이터 요약

PR ID	제목	평균 등급	작성일자
6	Test PR Review 1	B	2025-01-19
12	Revert "[feat/#57] 보고서 제작에 DEEPSEEK API 연동 구현"	B	2025-01-20
20	Revert "[feat/#57] 보고서 제작에 DEEPSEEK API 연동 구현"	B	2025-01-20
26	Revert "[feat/#57] 보고서 제작에 DEEPSEEK API 연동 구현"	A	2025-01-21
27	Revert "[feat/#57] 보고서 제작에 DEEPSEEK API 연동 구현"	A	2025-01-21

2. 분석 내용

2-1. 리뷰 결과 통계

- **분석된 PR 수**: 5
- **Clean 모드**: 1개의 리뷰
- **Optimize 모드**: 0개의 리뷰
- **Study 모드**: 0개의 리뷰
- **newbie 모드**: 0개의 리뷰
- **basic 모드**: 0개의 리뷰

2-2. 주요 취약점 및 개선 우선순위

취약한 유형 통계 및 개선 방향:
- **취약점 유형 문제점**:

- 코드 가독성 부족: 변수명이 직관적이지 않거나, 코드 구조가 복잡하여 이해하기 어려움.

- 중복 코드: 동일한 로직이 여러 곳에서 반복되어 유지보수가 어려움.

- 예외 처리 미흡: 예외 상황에 대한 처리가 충분히 고려되지 않아 안정성이 떨어짐.

- **개선 방향**:

- **코드 가독성**: 변수명을 직관적으로 변경하고, 함수를 작은 단위로 분리하여 가독성을 높임.

- **중복 코드 제거**: 공통 로직을 함수로 추출하여 재사용성을 높임.

- **예외 처리 강화**: 모든 예외 상황을 고려하여 적절한 예외 처리를 추가함.

- **안좋은 예시와 좋은 예시**:

- **안좋은 예시**:

```python

def calc(a, b):

return a + b

```

- 변수명 `a`, `b`가 무엇을 의미하는지 명확하지 않음.

- **좋은 예시**:

```python

def calculate_sum(num1, num2):

return num1 + num2

```

- 변수명이 직관적이며, 함수명도 명확함.

2-3. 개인화된 피드백 및 권장사항

사용자 맞춤 개선 방향:
- **가장 낮은 점수를 받은 평가 기준**: 코드 가독성
- **개선 방향**:

- 변수명과 함수명을 직관적으로 변경하여 코드의 의도를 명확히 표현.

- 함수를 작은 단위로 분리하여 각 함수가 하나의 역할만 수행하도록 함.

- 주석을 적절히 추가하여 코드의 흐름을 설명.

- **추가 권장사항**:

- 코드 리뷰를 통해 동료 개발자와의 협업을 강화하여 코드 품질을 높임.

- 리팩토링을 주기적으로 수행하여 코드의 유지보수성을 높임.

2-4. 종합 결론

- **총평**:

- 프로젝트의 전체적 성향 및 평균 등급을 출력하고, 코드 가독성 부분에서 개선 여지가 가장 큽니다. 코드의 구조와 명명법을 개선하여 더 나은 품질의 코드를 작성할 수 있습니다.

- **강점**:

1. 기본적인 기능 구현이 잘 되어 있습니다.

2. 코드의 논리적 구조가 명확합니다.

3. 필요한 기능을 빠르게 구현할 수 있는 능력이 있습니다.

- **약점**:

1. 코드 가독성이 떨어져 이해하기 어렵습니다.

2. 중복 코드가 많아 유지보수가 어렵습니다.

3. 예외 처리가 미흡하여 안정성이 떨어집니다.

- **향후 권장 사항**:

- Clean 모드를 사용하며 코드의 가독성과 유지보수성을 높이는 데 집중하세요.

- 리팩토링을 통해 중복 코드를 제거하고, 예외 처리를 강화하여 더 안정적인 코드를 작성하세요.

첨부 자료

- **추천 학습 자료**:

- "Clean Code" by Robert C. Martin

- "Refactoring: Improving the Design of Existing Code" by Martin Fowler

- **관련 예시 코드**:

```python

# 리팩토링 전
```

```
def process_data(data):
result = []
for item in data:
if item > 10:
result.append(item * 2)
return result

# 리팩토링 후
def filter_and_transform_data(data):
return [item * 2 for item in data if item > 10]
```

- 리팩토링을 통해 코드의 가독성과 간결성을 높인 예시입니다.