🔧 Flownote Classification Pipeline: Complete Error Resolution Guide

> **작성일:** 2025-11-09~10
> **소요 시간:** 약 16시간
> **최종 결과:** Classification Pipeline Debuggin 전체적 재점검 및 문제 해결

1. 📝 문서 개요

> 이 문서는 Flownote MVP의 분류 API를 구현하는 동안 발생한 주요 문제들과 해결 과정을 기록한 가이드입니다.
> *2025년 11월 10일 새벽까지 약 16시간 동안의 디버깅 과정*을 포함합니다.

⚠ 주의: 이 문서는 개발자의 실제 고생한 과정과 함께, 나중에 같은 문제를 겪는 개발자를 위한 참고용으로 작성됨

---

2. 📂 주요 아키텍처 구성

2.1. 시스템 구조

```
Flownote MVP Backend
├── backend/
│   ├── classifier/          # 분류기 (PARA + Keyword + Conflict Resolver)
│   ├── services/            # 서비스 레이어 (ParallelProcessor 등)
│   ├── routes/              # 라우트 설정
│   ├── api/                 # API
│   ├── config.py            # 설정 파일
│   ├── ...
│   └── data_manager.py
│
├── data/
│   ├── classifications/     # CSV 로그 저장
│   ├── users/               # 사용자 id별 저장
│   ├── context/             # 사용자 맥락 저장
│   ├── classifications/     # CSV 로그 저장
│   └── log/                 # JSON 로그 저장
│
└── tests/                   # .gitignore 처리됨
    └── test_classification_flow.py  # 테스트 파일
```

2.2. 핵심 컴포넌트

- **PARA Classifier**: LangChain 기반 카테고리 분류 (Projects/Areas/Resources/Archives)
- **Keyword Classifier**: 사용자 컨텍스트 기반 키워드 추출
- **Conflict Resolver**: 분류 충돌 해결 (다양한 가중치 적용)
- **DataManager**: 파일 데이터 관리 클래스
- **ClassificationMetadataExtender**: DB 결과 저장
- **API Router**: FastAPI 라우팅 (`/api/classify/classify`)

3. 🪲 트러블슈팅: 주요 문제들

3.1. **스냅샷 ID 관련 오류** (최초 발생한 치명적 문제)

증상 1: `snapshot_id_val` NameError

① 터미널 에러:

```
NameError: name 'snapshot_id_val' is not defined
File "backend/routes/classifier_routes.py", line 589, in classify_text
    "snapshot_id": snapshot_id_val,
File "backend/routes/classifier_routes.py", line 605, in classify_text
    "snapshot_id": snapshot_id_val,
```

② 문제 코드:

```
# ❌ 텍스트 로그에서
- 스냅샷 ID: {snapshot_id_val}

# ❌ JSON 로그에서
"snapshot_id": snapshot_id_val,
```

③ 원인:

- **변수 미정의**: `snapshot_id_val` 변수를 선언하지 않고 JSON에서 직접 사용
- **지역변수 범위**: 다른 함수에서 `snapshot_id`로 생성되지만, `locals()` 범위 밖에서 사용
- **타이밍 이슈**: 파이썬이 `locals()` 범위를 제대로 인식하지 못함

④ 해결책:

```
# ✅ 수정된 버전 (안전한 get + fallback)
snapshot_id = None
if 'para_result' in locals():
    if hasattr(para_result, 'snapshot_id') and
para_result.snapshot_id:
        snapshot_id = para_result.snapshot_id
    else:
        snapshot_id = f"snap_{int(time.time())}"

"snapshot_id": snapshot_id if snapshot_id else
f"snap_{int(time.time())}",
```

⑤ 터미널 확인 명령어:

```
python -c "
from backend.classifier.snapshot_manager import SnapshotManager
from datetime import datetime
print('스냅샷 생성 테스트:')
test_snapshot = SnapshotManager().create_snapshot('테스트 텍스트', {}, {})
print(f'생성된 ID: {test_snapshot.id}')
print(f'형태: {type(test_snapshot.id)}')
"
```

⑥ 기대 결과:

```
스냅샷 생성 테스트:
생성된 ID: snap_20251110_050500_123456
형태: <class 'str'>
```

3.2. `DataManager.log_classification()` 파라미터 불일치

증상:

```
WARNING:backend.routes.classifier_routes:⚠ CSV 로그 저장 실패:
DataManager.log_classification() got an unexpected keyword argument
'keyword_tags'
```

① 문제 코드:

```
# ❌ 7개 파라미터로 호출 (실제 함수는 5개만 지원)
log_result = data_manager.log_classification(
    user_id=request.user_id or "anonymous",
    file_name=request.file_id or "unknown",
    ai_prediction=conflict_result.get('final_category') if
'conflict_result' in locals() else '기타',
    user_selected=None,
    confidence=conflict_result.get('confidence', 0.0) if
'conflict_result' in locals() else 0.0,
    keyword_tags=new_keyword_tags if 'new_keyword_tags' in locals()
else ['기타'],
    user_areas=request.areas  # ❌ DataManager에는 없는 매개변수!
)
```

② `test_classification_flow.py`에서 성공한 스니펫:

```
# ✅ 5개 파라미터만 사용 (성공 확인됨)
def test_4_classification_log():
```

```
        dm = DataManager()
        result = dm.log_classification(
            user_id="test_user",
            file_name="test_file.txt",
            ai_prediction="Projects",
            user_selected=None,
            confidence=0.9
        )
        return result
```

③ 해결책:

```
    # test 파일과 정확히 동일한 5개 파라미터만 사용 →
backend/routet/classifier_routes.py에 적용
    csv_log_result = data_manager.log_classification(
        user_id=request.user_id or "anonymous",
        file_name=request.file_id or "unknown",
        ai_prediction=final_category,
        user_selected=None,
        confidence=final_confidence
        # keyword_tags, user_areas 완전 제거!
    )
```

④ `DataManager` 매개변수 확인 방법:

```
# 터미널에서 함수 시그니처 확인
    python -c "
    try:
        from backend.data_manager import DataManager
        import inspect
        sig = inspect.signature(DataManager.log_classification)
        print('DataManager.log_classification 매개변수: 5개')
        print('실제 매개변수:', list(sig.parameters.keys()))
        print('↓ 정확히 이 5개만 사용해야 함 ↓')
        print('1. user_id: str')
        print('2. file_name: str')
        print('3. ai_prediction: str')
        print('4. user_selected: None|str')
        print('5. confidence: float')
    except Exception as e:
        print('DataManager 임포트 실패:', e)
    "
```

3.3. JSON 직렬화 오류 (Snapshot 객체)

증상:

```
    ERROR:backend.routes.classifier_routes:⚠ JSON 로그 저장 실패: Object of
type Snapshot is not JSON serializable
```

① 문제 코드:

```
    # ❌ Snapshot 객체 직접 JSON에 포함 시도
    simple_log = {
        "timestamp": timestamp,
        "user_id": request.user_id,
        "result": snapshot,  # ❌ Snapshot 객체는 JSON 직렬화 불가
        "classification": classification_result
    }
```

② 원인 분석:

- **Snapshot 클래스**: LangChain 프레임워크의 객체, `json.dumps()` 불가
- **객체 속성**: `snapshot_id`는 있지만, 객체 자체는 `JSON 변환 필요`
- **타이밍**: `conflict_result`에 포함된 `Snapshot 객체`를 `JSON으로 직렬화 시도`

③ 해결책:

```
    # ✅ 안전한 직렬화 (객체를 기본 타입으로 변환)
    def safe_extract_snapshot(obj):
        """Snapshot 객체에서 필요한 정보만 안전하게 추출"""
        if obj and hasattr(obj, 'snapshot_id') and obj.snapshot_id:
            return {
                "id": str(obj.snapshot_id),
                "timestamp": str(obj.timestamp) if hasattr(obj,
'timestamp') else None,
                "text": getattr(obj, 'text', None)[:100] if hasattr(obj,
'text') else None,
                "status": "extracted"
            }
        return {"id": "N/A", "status": "missing"}

    # JSON 로그용 안전한 데이터 구성
    safe_snapshot = safe_extract_snapshot(para_result) if 'para_result' in
locals() else None

    simple_log = {
        "timestamp": timestamp,
        "user_id": request.user_id or "anonymous",
        "text_preview": request.text[:100],
        "category": category,
        "confidence": float(confidence),
        "keyword_tags": keyword_tags,
        "snapshot": safe_snapshot,
```

```
        "user_areas": user_areas
    }
```

④ 완전한 JSON 로그 저장 안전 코드:

```
    def save_json_log(data_manager_result, para_result, request, category,
confidence, keyword_tags):
        """JSON 로그를 안전하게 직렬화하여 저장"""
        from pathlib import Path
        from datetime import datetime

        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S_%f")[:-3]

        # 안전한 스냅샷 ID 추출
        snapshot_id = "N/A"
        if 'para_result' in locals():
            if hasattr(para_result, 'snapshot_id') and
para_result.snapshot_id:
                snapshot_id = str(para_result.snapshot_id)

        # 안전한 데이터만 사용
        safe_log_data = {
            "timestamp": timestamp,
            "user_id": request.user_id or "anonymous",
            "file_id": request.file_id or "unknown",
            "text_preview": request.text[:100] if hasattr(request, 'text')
else "",
            "category": category,
            "confidence": float(confidence) if confidence is not None else
0.0,
            "keyword_tags": keyword_tags if keyword_tags else ['기타'],
            "snapshot_id": snapshot_id,
            "user_areas": request.areas if hasattr(request, 'areas') else
[],
            "matched_context": user_context_matched if
'user_context_matched' in locals() else False
        }

        PROJECT_ROOT = Path(__file__).parent.parent.parent
        LOG_DIR = PROJECT_ROOT / "data" / "log"
        LOG_DIR.mkdir(parents=True, exist_ok=True)

        json_filename = LOG_DIR / f"classification_clean_{timestamp}.json"

        try:
            with open(json_filename, "w", encoding="utf-8") as f:
                json.dump(safe_log_data, f, ensure_ascii=False, indent=2)

            logger.info(f"✅ JSON 로그 저장: {json_filename.name}")
            return json_filename.name

        except Exception as json_error:
```

```
            logger.warning(f"⚠ JSON 로그 저장 실패: {json_error}")
            return None
```

3.4. f-string 포맷 문자열 구문 오류

증상:

```
    ERROR:backend.routes.classifier_routes:⚠ 텍스트 로그 저장 실패: Invalid
format specifier '.2f if 'conflict_result' in locals() else
para_result.get('confidence', 0.0):.2f if 'para_result' in locals() else
0.0' for object of type 'float'
```

① 문제 코드:

```
    # ❌ f-string 안에서 조건문 사용 (포맷과 구문 충돌)
    log_message = f"분류 신뢰도: {confidence:.2f if 'result' in locals() else
para_result.get('confidence', 0.0):.2f}%"
```

② 원인:

- **포맷 스펙 충돌**: f-string에서 {변수:포맷}과 if 조건문이 혼동됨
- **구문 해석**: 파이썬이 조건문을 포맷 스펙으로 잘못 해석
- **복잡한 조건문**: locals() 같은 함수가 f-string 내부에서 제대로 작동하지 않음

③ 해결책:

```
    # ✅ 조건문을 별도 변수에 저장 후 포맷 적용
    if 'conflict_result' in locals() and conflict_result:
        final_confidence = conflict_result.get('confidence', 0.0)
    elif 'para_result' in locals() and para_result:
        final_confidence = para_result.get('confidence', 0.0)
    else:
        final_confidence = 0.0

    # 안전한 f-string 사용
    log_message = f"분류 신뢰도: {final_confidence:.2f}%"
    print(log_message)
```

④ 안전한 f-string 사용 패턴:

```
    # ❌ 복잡한 조건문 직접 포맷에 포함
    f"총시간: {total_time:.2f if total_time is not None else 'N/A':.2f}s"
```

```
    # ✅ 별도 변수에 조건문 먼저 저장
    if total_time is not None:
        time_display = f"{total_time:.2f}s"
    else:
        time_display = "시간 없음"

    print(f"총시간: {time_display}")

    # 또는 포맷이 단순한 경우
    time_display = total_time if total_time is not None else 0.0
    print(f"총시간: {time_display:.2f}s" if time_display else "시간 없음")
```

⑤ 콘솔 로그 포맷 에러 확인:

```
    ERROR:backend.routes.classifier_routes:⚠ 텍스트 로그 저장 실패: Invalid
format specifier '.2f if ...'
```

3.5. 변수 스코프 및 locals() 함수 문제

증상:

```
    # Step 5에서 선언한 변수가 Step 6에서 접근 불가
    if 'log_info' in locals():
        response.log_info = log_info  # NameError 발생
```

① 문제:

- **함수 간 변수 전달**: Step 4에서 선언한 변수가 Step 5에서 접근 불가
- **`locals()` 제한**: `locals()` 함수가 예상대로 작동하지 않음
- **스코프 경계**: 함수 경계에서 변수 범위 문제

② 해결책:

```
    # ❌ locals() 함수 의존 (권장 안 함)
    def step_5():
        log_info = generate_logs()
        if 'log_info' in locals():  # 불안정함!
            return log_info

    def step_6():
        if 'log_info' in locals():  # 이전 함수 결과가 보장 안됨
            use_log_info(log_info)

    # ✅ 함수 반환값으로 명확한 데이터 전달
    def step_5():
        log_info = generate_logs()
```

```
        return log_info

    def step_6(previous_result):
        log_info = previous_result.get('log_info')
        use_log_info(log_info)

    # 더 명확한 방법: 명시적 함수 체이닝
    def process_classification_full(request):
        # Step 1-4 처리
        intermediate_result = process_steps_1_4(request)

        # Step 5: 로그 생성
        log_info = create_log_info(intermediate_result)

        # Step 6: 응답 생성
        response = build_response(intermediate_result, log_info)

        return {
            "response": response,
            "log_info": log_info,
            "success": True
        }
```

③ 클래스 기반 안전한 상태 관리:

```
    # ✅ 클래스 속성 사용으로 변수 스코프 문제 해결
    class ClassificationService:
        def __init__(self, request):
            self.request = request
            self.intermediate_results = {}
            self.log_info = {}

        async def process_text(self):
            # Step 1-3
            self.intermediate_results['step3'] = await self.step3()

            # Step 4: 병렬 처리
            self.intermediate_results['step4'] = await self.step4()

            # Step 5: 로그 생성 (클래스 속성 사용)
            self.log_info = await self.log_results()
            return self.intermediate_results

        def step_5_log_results(self):
            # self.intermediate_results에 직접 접근 가능
            result = self.intermediate_results['step3']
            return generate_logs(result)
```

3.6. 컨텍스트 인젝션 및 사용자 프로필 처리

증상:

```
    INFO:backend.routes.classifier_routes:   - Areas: ['개인 생활']  # 입력
    INFO:backend.classifier.para_agent:사용자 책임 영역('기술 역량 개발')  # 다른
영역 참조
```

① 문제:

- **사용자 프로필**: `request.occupation`, `request.areas` 정상 입력
- **PARA 분류기**: 내부적으로 하드코딩된 영역('기술 역량 개발') 참조
- **컨텍스트 불일치**: 사용자 영역과 분류기가 다른 컨텍스트 사용

② 해결책:

```
    # ✅ 사용자 컨텍스트를 명확히 전달
    class ContextInjector:
        def inject_context(self, request):
            return {
                "occupation": request.occupation,
                "areas": request.areas,
                "interests": request.interests,
                "context_keywords":
self.generate_context_keywords(request.areas)
            }

    def run_para_agent_with_context(text, context):
        # 사용자 컨텍스트를 명확히 전달
        prompt = f"""
        사용자 프로필: 직업={context['occupation']}, 책임영역={context['areas']},
관심사={context['interests']}
        분류할 텍스트: {text}

        사용자가 {context['occupation']}이고, 책임영역은 {context['areas']}이며,
        관심사는 {context['interests']}인 점을 고려하여 PARA 분류.
        """

        return run_llm(prompt)
```

3.7. API 응답 직렬화 및 모델 확장

증상:

```
    # 응답에서 log_info 필드 누락
    "category": "Resources",
    "confidence": 0.85,
    # log_info 필드가 보이지 않음
```

① 문제:

- **Pydantic 모델**: ClassifyResponse에 `log_info` 필드 정의 안됨
- **직렬화 무시**: 정의되지 않은 필드는 기본값으로 무시됨
- **모델 확장 필요**: 로그 정보 응답에 포함 필수

② 해결책:

```
    # backend/api/models.py 수정
    from pydantic import BaseModel
    from typing import Optional, List, Dict, Any

    class LogInfo(BaseModel):
        csv_log: str = "data/classifications/classification_log.csv"
        db_saved: bool = False
        json_log: Optional[str] = None
        log_directory: str = "data/classifications"

    class ClassificationContext(BaseModel):
        user_id: str = ""
        file_id: Optional[str] = None
        occupation: str = ""
        areas: List[str] = []
        interests: List[str] = []
        context_keywords: Dict[str, List[str]] = {}

    class ClassifyResponse(BaseModel):
        category: str
        confidence: float
        snapshot_id: str = "N/A"
        conflict_detected: bool = False
        requires_review: bool = False
        keyword_tags: List[str] = []
        reasoning: str = ""
        user_context_matched: bool = False
        user_areas: List[str] = []
        user_context: ClassificationContext = ClassificationContext()
        context_injected: bool = False
        log_info: LogInfo = LogInfo()  # 필수 필드 추가!

        class Config:
            json_encoders = {
                datetime: lambda v: v.isoformat() if isinstance(v,
datetime) else None,
            }
```

3.8. 임포트 및 모듈 경로 문제

증상:

```
    ImportError: cannot import name 'DataManager' from
'backend.data_manager'
    ModuleNotFoundError: No module named
'backend.database.metadata_schema'
```

① 문제:

- **프로젝트 루트**: 잘못된 디렉토리에서 실행
- **Python 경로**: `sys.path`에 프로젝트 루트 추가 안됨
- **상대 임포트**: 모듈 경로가 제대로 구성 안됨

② 해결책:

```
    # backend/routes/classifier_routes.py 맨 위에 추가
    import sys
    from pathlib import Path

    # 프로젝트 루트 동적 추가
    PROJECT_ROOT = Path(__file__).parent.parent.parent
    sys.path.insert(0, str(PROJECT_ROOT))

    # 안전한 임포트
    try:
        from backend.data_manager import DataManager
        from backend.classifier.keyword_classifier import
KeywordClassifier
        from backend.database.metadata_schema import
ClassificationMetadataExtender
        print("✅ 모든 임포트 성공")
    except ImportError as e:
        print(f"❌ 임포트 실패: {e}")
        sys.exit(1)

    # 또는 터미널에서 환경변수 설정
    export PYTHONPATH="/Users/jay/ICT-projects/flownote-mvp:$PYTHONPATH"
```

③ 터미널에서 임포트 경로 확인:

```
    # 1. 현재 작업 디렉토리 확인
    pwd  # /Users/jay/ICT-projects/flownote-mvp 여야 함

    # 2. Python 경로 확인
    python -c "
    import sys
    print('현재 작업 디렉토리:', os.getcwd())
    print('Python 경로:')
    for path in sys.path:
        print('  -', path)
```

```
print('프로젝트 루트 포함 여부:', os.getcwd() in sys.path)
"

# 3. sys.path 수동 추가 테스트
python -c "
import sys
from pathlib import Path
project_root = Path('/Users/jay/ICT-projects/flownote-mvp')
sys.path.insert(0, str(project_root))
print('수정된 경로:', sys.path)

try:
    from backend.data_manager import DataManager
    print('✅ DataManager 임포트 성공')
    from backend.api.models import ClassifyResponse
    print('✅ API 모델 임포트 성공')
except ImportError as e:
    print('❌ 임포트 실패:', e)
"
```

3.9. 테스트 케이스와 프로덕션 코드 불일치

`test_classification_flow.py` 분석:

① 성공한 테스트 코드:

```
def test_4_classification_log():
    """테스트 4: classification_log.csv 기록 테스트"""

    from backend.data_manager import DataManager
    dm = DataManager()

    # 정확히 5개 파라미터만 사용 (성공)
    result = dm.log_classification(
        user_id="test_user",
        file_name="test_file.txt",
        ai_prediction="Projects",
        user_selected=None,
        confidence=0.9
    )

    print(f"✅ 로그 기록 결과: {result}")

    # 로그 파일 확인
    log_file = Path("data/classifications/classification_log.csv")
    if log_file.exists():
        with open(log_file, 'r', encoding='utf-8') as f:
            lines = f.readlines()
            print(f"✅ 로그 파일 총 {len(lines)}줄")
            if len(lines) > 1:
```

```
                    print(f"\n마지막 줄:")
                    print(f"  {lines[-1].strip()}")
```

② API에서 실패한 부분:

```
# ❌ 테스트와 다른 파라미터 사용
def log_result_in_api():
    data_manager = DataManager()
    result = data_manager.log_classification(
        user_id=request.user_id,
        file_name=request.file_id,
        ai_prediction=category,
        user_selected=None,
        confidence=confidence,
        # 테스트에는 없는 추가 파라미터들
        keyword_tags=keyword_tags,
        user_areas=request.areas
    )
```

③ 해결책:

```
# ✅ 테스트와 완전히 동일한 5개 파라미터만 사용 → 성공
def log_result_in_api():
    data_manager = DataManager()
    result = data_manager.log_classification(
        user_id=request.user_id or "anonymous",
        file_name=request.file_id or "unknown",
        ai_prediction=category,
        user_selected=None,
        confidence=float(confidence)  # float()으로 타입 보장
    )
    return result
```

3.10. **logging 구성 및 best practices**

효과적인 로깅 설정:

```
# classifier_routes.py에서
import logging
from pathlib import Path

# 개발 시 디버깅을 위한 설정
logging.basicConfig(
    level=logging.INFO,  # 개발: DEBUG, 프로덕션: INFO
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
```

```
            logging.FileHandler('data/debug.log'),
            logging.StreamHandler()  # 터미널 출력
        ]
    )

    logger = logging.getLogger(__name__)

    # 안전한 로깅 예시
    def safe_log_classification(text, category, confidence):
        """안전한 로깅: 중요한 메시지만, 민감한 데이터 보호"""
        logger.info(f"분류 시작 - 카테고리: {category}, 신뢰도:
{confidence:.1f}")
        logger.debug(f"입력 텍스트: {text[:50]}...")  # 50자 미리보기만
        logger.warning(f"민감한 전체 텍스트는 로그 안남!")  # 전체 텍스트는 안남

        try:
            result = process_text(text, category)
            logger.info(f"분류 완료: {result['category']} (시간:
{result['processing_time']:.2f}s)")
            return result
        except Exception as e:
            logger.error(f"분류 실패: {e}", exc_info=True)
            raise
```

① 로그 레벨별 사용:

- **DEBUG**: 상세한 내부 로직 추적 (개발용)
- **INFO**: 주요 단계 완료, 결과 요약
- **WARNING**: 비정상적이지만 계속 가능한 상황
- **ERROR**: 심각한 오류, 시스템 중단

② 민감한 데이터 보호:

```
    # ❌ 전체 텍스트 로깅 (보안 문제)
    logger.debug(f"전체 텍스트: {full_sensitive_text}")

    # ✅ 안전한 텍스트 미리보기
    TEXT_PREVIEW_LENGTH = 50
    safe_text = full_sensitive_text[:TEXT_PREVIEW_LENGTH] + "..." if
len(full_sensitive_text) > TEXT_PREVIEW_LENGTH else full_sensitive_text
    logger.debug(f"텍스트 미리보기: {safe_text}")
```

---

4. 🔧 단계별 디버깅 체크리스트

4.1. 체크리스트 ①: 환경 및 경로 설정

```
    # 1. 프로젝트 루트 디렉토리 확인
    pwd                                          # 루트 디렉토리여야 함
```

```
# 2. 가상환경 활성화 확인
which python                                    # 가상환경 경로 확인
pip list | grep fastapi                         # FastAPI 설치 확인

# 3. 파일 구조 검증
ls -la backend/                                 # 주요 디렉토리 존재 확인
ls -la backend/routes/                          # API 라우트 파일들 확인
ls -la backend/classifier/                      # 분류기 파일들 확인
ls -la data/                                    # 로그 디렉토리들 확인

# 4. Python 경로 디버깅
python -c "
import sys
from pathlib import Path
project_root = Path('/Users/jay/ICT-projects/flownote-mvp')
sys.path.insert(0, str(project_root))
print('프로젝트 루트 추가:', str(project_root))
print('현재 Python 경로:', sys.path[:3])  # 처음 3개 경로만 출력

# 핵심 임포트 테스트
from backend.data_manager import DataManager
print('✅ DataManager 임포트 성공')
except Exception as e:
    print('❌ DataManager 임포트 실패:', e)
"
```

4.2. 체크리스트 ②: API 엔드포인트 검증

```
# 1. 서버 상태 확인
curl -X GET "http://localhost:8000/" | grep -i fastapi

# 2. OpenAPI 스키마 다운로드
curl -X GET "http://localhost:8000/openapi.json" -o api_schema.json

# 3. API 엔드포인트 개수 확인
python -c "
import json
with open('api_schema.json', 'r') as f:
    schema = json.load(f)
endpoints = len(schema['paths'])
print(f'총 {endpoints}개 API 엔드포인트 확인')
for path in schema['paths'].keys():
    if '/classify' in path:
        print(f'  ✅ 분류 API: {path}')
"

# 4. 기본 POST 테스트 (최소 파라미터)
curl -X POST "http://localhost:8000/api/classify/classify" \
-H "Content-Type: application/json" \
-d '{"text": "기본 테스트"}' | jq '.'
```

```
    # 5. 완전한 컨텍스트 테스트
    curl -X POST "http://localhost:8000/api/classify/classify" \
    -H "Content-Type: application/json" \
    -d '{
        "text": "디버그: 완전한 파라미터로 테스트",
        "user_id": "debug_complete",
        "occupation": "디버그 직업",
        "areas": ["디버그 영역"],
        "interests": ["디버그 관심사"]
    }' | jq '.'
```

4.3. 체크리스트 ③: 로그 시스템 검증

```
    # 1. 로그 디렉토리 생성 및 권한
    mkdir -p data/{classifications,log}
    chmod 755 data/ data/classifications/ data/log/

    # 2. CSV 헤더 초기화 (새 테스트용)
    echo
"timestamp,user_id,file_name,ai_prediction,user_selected,confidence" >
data/classifications/classification_log.csv

    # 3. 테스트 실행 후 로그 확인
    curl -X POST "http://localhost:8000/api/classify/classify" \
    -H "Content-Type: application/json" \
    -d '{"text": "로그 테스트"}' | jq '.'

    # 4. CSV 누적 확인
    if [ -f data/classifications/classification_log.csv ]; then
        wc -l data/classifications/classification_log.csv               #
헤더 + 새 행
        tail -n 2 data/classifications/classification_log.csv          #
최근 2개 행
    else
        echo "❌ CSV 파일 생성 실패"
    fi

    # 5. JSON 로그 확인
    ls -la data/log/ | grep classification_clean_                      #
새 파일 생성 확인
```

4.4. 체크리스트 ④: 성능 및 응답 검증

```
    # 1. 응답 시간 측정
    time curl -X POST "http://localhost:8000/api/classify/classify" \
    -H "Content-Type: application/json" \
    -d '{"text": "성능 테스트"}' | jq '.'
```

```
    # 2. 연속 요청 테스트 (중복 처리 방지)
    for i in {1..3}; do
        echo "테스트 $i:"
        curl -X POST "http://localhost:8000/api/classify/classify" \
        -H "Content-Type: application/json" \
        -d "{
            \"text\": \"연속 테스트 $i: 다른 사용자 $i\",
            \"user_id\": \"perf_test_$i\"
        }" | jq '.'
    done

    # 3. CSV 누적 검증
    wc -l data/classifications/classification_log.csv               # 4줄
(헤더 + 3개 테스트)

    # 4. JSON 파일 개수 확인
    ls data/log/classification_clean_*.json | wc -l                 # 3개
JSON 파일
```

---

5. 📊 16시간 디버깅 결과 요약

5.1. **성공한 컴포넌트들:**

기능	상태	테스트 성공률
텍스트 분류 (PARA)	✅ 100%	7/7 케이스
키워드 추출	✅ 100%	모든 컨텍스트 매칭
CSV 로그 누적	✅ 100%	7개 행 저장
JSON 로그 생성	✅ 100%	7개 파일 생성
DB 저장	✅ 100%	ClassificationMetadataExtender 성공
스냅샷 ID 생성	✅ 100%	모든 요청에 유효 ID
사용자 컨텍스트 매칭	✅ 100%	다양한 직업/영역 지원

분류 결과 다양성:

- **Projects**: 프로젝트 마감일 (0.95)
- **Areas**: 건강 관리, 기술 학습 (0.90, 0.85)
- **Resources**: 요리, 디자인 연구, 기술 문서 (0.85)
- **Archives**: (아직 테스트 안함, 예상 가능)

5.2. **성공 지표:**

```
📈 최종 통계 (7개 테스트 완료)
├── API 응답: 100% 성공 (7/7)
├── CSV 누적: 100% 성공 (7개 행)
```

```
├── JSON 로그: 100% 성공 (7개 파일)
├── DB 저장: 100% 성공 (file_id 57~69)
├── 컨텍스트 매칭: 100% (user_context_matched: true)
├── 카테고리 다양성: 80% (Projects, Areas, Resources)
└── 처리 시간: 평균 7.5초 (3.47~13.86초)
```

🎯 **완전 성공! 16시간 디버깅 마라톤 완주!**

---

6. 📚 결론 및 다음 단계

6.1. 배운 교훈들:

1. **단계적 접근**: 큰 기능부터 작은 디테일로 나누어 해결
2. **매개변수 일치**: 테스트 코드와 실제 코드의 파라미터 정확히 맞춤
3. **변수 스코프 관리**: 함수 반환값으로 명확한 데이터 전달
4. **JSON 직렬화**: 객체를 기본 타입으로 변환 후 사용
5. **로깅 필수**: 각 단계별로 `logger.info()` 사용
6. **환경 설정**: `sys.path`, 프로젝트 루트 경로 확인

6.2. 다음 구현 권장사항:

① 파일 업로드 API:

- **PDF, DOCX, TXT** 지원 텍스트 추출
- **현재 API 호출**: 텍스트 추출 후 동일한 분류 로직 사용
- **확장성**: OCR 추가 고려 (이미지/스캔 문서)

② 프론트엔드 연동:

- **React/Next.js** 파일 업로드 컴포넌트
- **드래그 앤 드롭** UI 개선
- **실시간 미리보기**: 업로드 전 분류 결과 프리뷰

③ 성능 최적화:

- **LM Studio 캐싱**: 자주 사용하는 프롬프트 캐시
- **비동기 처리**: 병렬로 PARA + Keyword 실행
- **배치 처리**: 대량 요청 시 효율적 처리

④ 사용자 경험:

- **시각적 결과**: 카테고리별 색상 코딩
- **키워드 하이라이트**: 실제 텍스트에서 추출된 키워드 강조
- **배치 분류**: 여러 파일 한번에 처리