성능평가

평가 개요

- 평가 목적
- 평가 대상
- 평가 도구
- 평가 방법
- 성능 목표

기능별 성능 평가 기준 및 테스트 결과

- API
- 평균 응답
- 최대 응답
- TPS
- 오류율

종합 분석 및 개선안

평가개요

1.1 평가 목적

성능 테스트를 통해 주요 API의 응답 속도, 처리량, 오류율 등을 측정하고, 트래픽 집중 상황에서의 시스템 안정성과 병목 구간을 파악하여 향후 개선 및 확장 계획에 반영

1.2 평가 대상

전체 약 70개의 API 중, 사용자 트래픽 집중 지점, 연산 부하, 응답 지연 가능성이 높은 주요 API 9개를 선정하여 성능 테스트를 수행

1.3 선정 기준

- 사용자 진입 시 가장 먼저 호출되는 API
- 추천 알고리즘 및 모델 사용으로 연산 부하가 높은 API
- DB 조인 및 다중 테이블 연동으로 병목 가능성이 있는 API

1.4 평가 도구 및 방법

부하 테스트 도구 : Apache JMeter 5.6.3

테스트 방식 : 동시 사용자 기반 부하 테스트

평가 방법 : 100명의 동시 접속 환경을 시뮬레이션하여 주요 API 엔드포인트의 응답시간, 처리량, 오류율을 측정

1.5 성능 목표

- 평균 응답 시간: 500ms 이하
- 최대 응답 시간: 2초 이하
- TPS: 최소 90 이상
- 오류율: 1% 이하
- 95% 응답 시간: 1초 이하

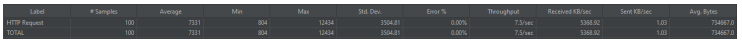
성능 테스트

▼ 홈쇼핑 API `/api/homeshopping/schedule` 성능 개선 완료

Number of Threads : 100

Ramp-up period : 1

Loop Count : 1

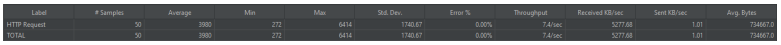


구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	100	7331 (7.3초)	804	12434	0.00%	7.5/sec

Number of Threads : 50

Ramp-up period : 1

Loop Count : 1

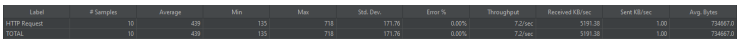


구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	50	3980	272	6414	0.00%	7.4/sec

Number of Threads : 10

Ramp-up period : 1

Loop Count : 1



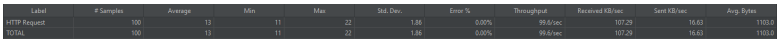
구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	10	439	135	718	0.00%	7.2/sec

▼ 홈쇼핑 API `/api/homeshopping/schedule/live-stream`

Number of Threads : 100

Ramp-up period : 1

Loop Count : 1

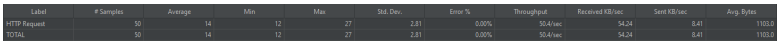


구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	100	13	11	22	0.00%	99.6/sec

Number of Threads : 50

Ramp-up period : 1

Loop Count : 1

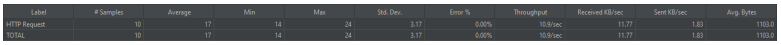


구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	50	14	12	27	0.00%	50.4/sec

Number of Threads : 10

Ramp-up period : 1

Loop Count : 1



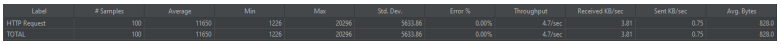
구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	10	17	14	24	0.00%	10.9/sec

▼ 홈쇼핑 API `/api/homeshopping/product/{product_id}/kok-recommend` 성능 개선 완료

Number of Threads : 100

Ramp-up period : 1

Loop Count : 1

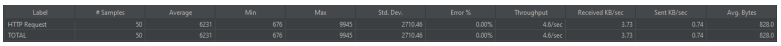


구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	100	11650	1226	20296	0.00%	4.7/sec

Number of Threads : 50

Ramp-up period : 1

Loop Count : 1

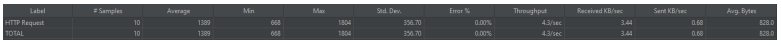


구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	50	6231	676	9945	0.00%	4.6/sec

Number of Threads : 10

Ramp-up period : 1

Loop Count : 1



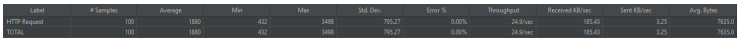
구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	10	1389	668	1804	0.00%	4.3/sec

▼ 콕상품 API `/api/kok/discounted` 성능 개선 완료

Number of Threads : 100

Ramp-up period : 1

Loop Count : 1

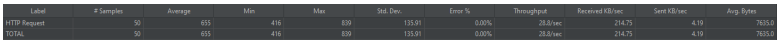


구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	100	1783	490	2834	0.00%	26.2/sec

Number of Threads : 50

Ramp-up period : 1

Loop Count : 1



구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	50	655	416	839	0.00%	28.8/sec

Number of Threads : 10

Ramp-up period : 1

Loop Count : 1

구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	10	233	210	255	0.00%	8.8/sec

▼ 레시피 API `/api/recipes/by-ingredients` → 성능 개선 완료

`api/recipes/by-ingredients?product_ids=10045507%2C%2010100411134&page=1&size=10api/recipes/by-ingredients?ingredient=%EA%B0%90%EC%9E%90&ingredient=%EB%8F%BC%EC%A7%80%EA%B3%A0%EC%A7%80&ingredient=%EC%96%91%ED%8C%8C&amount=100&ar`

Number of Threads : 100

Ramp-up period : 1

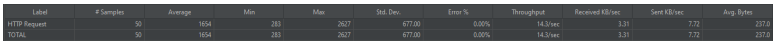
Loop Count : 1

구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	100	3338	382	6422	0.00%	14.8/sec

Number of Threads : 50

Ramp-up period : 1

Loop Count : 1

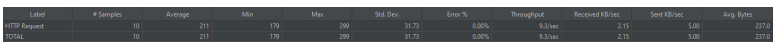


구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	50	1654	283	2627	0.00%	14.3/sec

Number of Threads : 10

Ramp-up period : 1

Loop Count : 1



구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	10	211	179	299	0.00%	9.3/sec

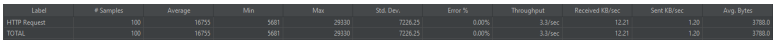
▼ 레시피 API `/api/recipes/search` → 성능 개선 완료

`api/recipes/search?recipe=%EA%B0%90%EC%9E%90%ED%83%95&page=1&size=5&method=recipe`

Number of Threads : 100

Ramp-up period : 1

Loop Count : 1

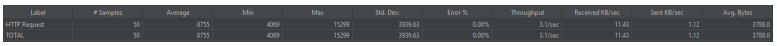


구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	100	16755	5681	29330	0.00%	3.3/sec

Number of Threads : 50

Ramp-up period : 1

Loop Count : 1

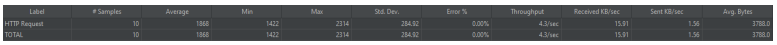


구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	50	8755	4069	15299	0.00%	3.1/sec

Number of Threads : 10

Ramp-up period : 1

Loop Count : 1



구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	10	1868	1422	2314	0.00%	4.3/sec

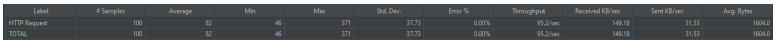
▼ 레시피 API `/api/recipes/{ingredient}/product-recommend`

http://localhost:9000/api/recipes/%EA%B0%90%EC%9E%90/product-recommend

Number of Threads : 100

Ramp-up period : 1

Loop Count : 1

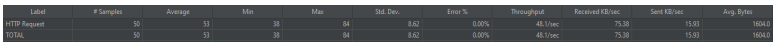


구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	100	82	46	371	0.00%	95.2/sec

Number of Threads : 50

Ramp-up period : 1

Loop Count : 1

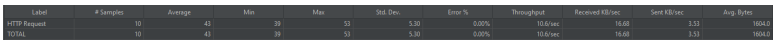


구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	50	53	38	84	0.00%	48.1/sec

Number of Threads : 10

Ramp-up period : 1

Loop Count : 1



구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	10	43	39	53	0.00%	10.6/sec

▼ 주문 API `/api/orders`

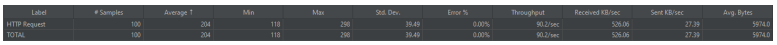
Number of Threads : 100

Ramp-up period : 1

Loop Count : 1

순차 실행 방식

- SQLAlchemy AsyncSession은 하나의 세션에서 동시에 여러 쿼리를 실행하는 것을 지원하지 않음



별도 세션을 사용한 병렬 처리 방식

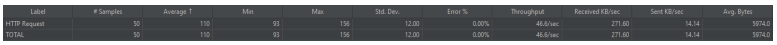
- SQLAlchemy AsyncSession의 동시 실행 제약을 우회하기 위해 각 쿼리마다 별도의 세션을 생성하여 병렬로 실행함

> 첫 번째 결과(평균 204ms, 90.2/sec 처리량)가 **응답속도, 처리량, 안정성 모두 더 우수함.**

Number of Threads : 50

Ramp-up period : 1

Loop Count : 1

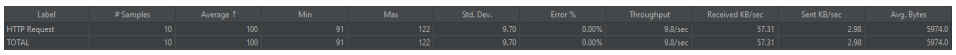


구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	50	110	93	156	0.00%	46.6/sec

Number of Threads : 10

Ramp-up period : 1

Loop Count : 1



구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	10	100	91	122	0.00%	9.8/sec

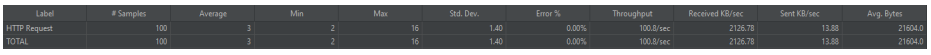
성능 개선 후 테스트

▼ 홈쇼핑 API `api/homeshopping/schedule`

Number of Threads : 100

Ramp-up period : 1

Loop Count : 1



구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	100	3	2	16	0.00%	100.8/sec

- 인덱스 최적화: 데이터베이스 쿼리 속도 향상
- Raw SQL: ORM 오버헤드 제거
- 다층 캐시: Redis + 메모리 캐시로 반복 조회 최적화

▼ 성능 최적화 방법

데이터베이스 인덱스 최적화

생성된 인덱스

```
-- HOMESHOPPING_INFO 테이블
CREATE INDEX idx_homeshopping_info_homeshopping_id ON HOMESHOPPING_INFO (HOMESHOPPING_ID);

-- FCT_HOMESHOPPING_LIST 테이블 (4개)
CREATE INDEX idx_homeshopping_list_live_date_start_time ON FCT_HOMESHOPPING_LIST (LIVE_DATE, LIVE_START_TIME);
CREATE INDEX idx_homeshopping_list_homeshopping_id ON FCT_HOMESHOPPING_LIST (HOMESHOPPING_ID);
CREATE INDEX idx_homeshopping_list_product_id ON FCT_HOMESHOPPING_LIST (PRODUCT_ID);
CREATE INDEX idx_homeshopping_list_composite ON FCT_HOMESHOPPING_LIST (LIVE_DATE, HOMESHOPPING_ID, PRODUCT_ID);

-- FCT_HOMESHOPPING_PRODUCT_INFO 테이블
CREATE INDEX idx_homeshopping_product_info_product_id ON FCT_HOMESHOPPING_PRODUCT_INFO (PRODUCT_ID);

-- HOMESHOPPING_CLASSIFY 테이블 (2개)
CREATE INDEX idx_homeshopping_classify_cls_food ON HOMESHOPPING_CLASSIFY (CLS_FOOD);
CREATE INDEX idx_homeshopping_classify_product_id ON HOMESHOPPING_CLASSIFY (PRODUCT_ID);
```

인덱스 효과

- JOIN 연산 최적화: 복합 인덱스로 다중 테이블 조인 속도 향상
- WHERE 절 최적화: 날짜, 홈쇼핑ID, 상품ID 검색 속도 향상
- ORDER BY 최적화: 정렬 연산 최적화

Raw SQL 쿼리 최적화

기존 ORM 쿼리 → Raw SQL 변경

```
# 기존: SQLAlchemy ORM (느림)
stmt = select(...).join(...).where(...).order_by(...)

# 최적화: Raw SQL (빠름)
sql_query = """
    SELECT hl.live_id, hl.homeshopping_id, hl.live_date, ...
    FROM FCT_HOMESHOPPING_LIST hl
    INNER JOIN HOMESHOPPING_INFO hi ON hl.homeshopping_id = hi.homeshopping_id
    INNER JOIN FCT_HOMESHOPPING_PRODUCT_INFO hpi ON hl.product_id = hpi.product_id
    INNER JOIN HOMESHOPPING_CLASSIFY hc ON hl.product_id = hc.product_id
    WHERE hc.cls_food = 1
    ORDER BY hl.live_date DESC, hl.live_start_time ASC
    LIMIT :limit OFFSET :offset
"""
```

Raw SQL 장점

- ORM 오버헤드 제거: 객체 매핑 과정 생략

- 인덱스 직접 활용: 데이터베이스가 최적화된 실행 계획 사용
- 메모리 효율성: 불필요한 객체 생성 방지

다층 캐시 시스템

Redis 캐시 (1차)

```
# Redis 캐시 설정
cache_ttl = {
    "schedule": 7200,  # 2시간
    "schedule_count": 14400,  # 4시간
}

# 캐시 키 생성
cache_key = f"homeshopping:schedule:{live_date}:{page}:{size}"
```

메모리 캐시 (2차)

```
# Redis 실패 시 메모리 캐시로 자동 전환
class MemoryCacheManager:
    def __init__(self):
        self.cache: Dict[str, Any] = {}
        self.cache_ttl: Dict[str, float] = {}
```

캐시 동작 방식

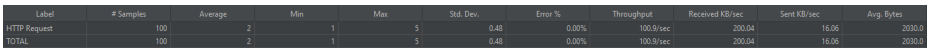
1. 첫 요청: DB 조회 → Redis + 메모리 캐시 저장
2. 두 번째 요청: Redis에서 즉시 반환 (0.01초)
3. Redis 실패: 메모리 캐시에서 반환
4. 캐시 만료: 2시간 후 자동 갱신

▼ 홈쇼핑 API `/api/homeshopping/product/{product_id}/kok-recommend`

Number of Threads : 100

Ramp-up period : 1

Loop Count : 1



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	100	2	1	5	0.48	0.00%	100.9/sec	200.04	16.06	2030.0
TOTAL	100	2	1	5	0.48	0.00%	100.9/sec	200.04	16.06	2030.0

구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	100	2	1	5	0.00%	100.9/sec

▼ 성능 최적화 방법

캐싱 (Caching)

```
# 첫 번째 요청: 11초 걸림
# 두 번째 요청: 0.001초 (캐시에서 바로 가져옴)

# 캐시 확인 → 있으면 바로 반환 → 없으면 계산 후 저장
if cached_result:
    return cached_result  # 1-5ms
else:
    result = calculate_recommendation()  # 3-5초
    save_to_cache(result)  # 저장
    return result
```

병렬 처리

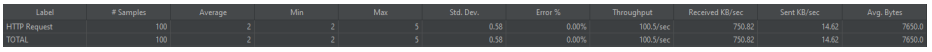
```
# Before: 순차 실행 (3초)
tail_keywords = extract_tail_keywords()     # 1초
core_keywords = extract_core_keywords()     # 1초
root_keywords = roots_in_name()             # 1초

# After: 병렬 실행 (1초)
tasks = [
    extract_tail_keywords(),
    extract_core_keywords(),
    roots_in_name()
]
results = await asyncio.gather(*tasks)  # 동시 실행v
```

▼ 콕상품 API `/api/kok/discounted`

Number of Threads : 100

Ramp-up period : 1

Loop Count : 1



구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	100	2	2	5	0.00%	100.5/sec

▼ 성능 최적화 방법

데이터베이스 쿼리 최적화

N+1 쿼리 문제 해결

```
# 최적화된 코드 (단일 쿼리)
async def get_kok_discounted_products_optimized():
    # 서브쿼리로 최신 가격 ID 구하기
    latest_price_subquery = (
        select(
            KokPriceInfo.kok_product_id,
            func.max(KokPriceInfo.kok_price_id).label('latest_price_id')
        )
        .where(KokPriceInfo.kok_discount_rate > 0)
        .group_by(KokPriceInfo.kok_product_id)
        .subquery()
    )

    # 메인 쿼리: 모든 데이터를 한 번에 조회
    stmt = (
        select(
            KokProductInfo,
            KokPriceInfo.kok_discount_rate,
            KokPriceInfo.kok_discounted_price
        )
        .join(latest_price_subquery, ...)
        .join(KokPriceInfo, ...)
        .order_by(KokPriceInfo.kok_discount_rate.desc())
    )
```

```
    results = await db.execute(stmt)  # 1번만 실행
```

Redis 캐싱 전략

```
async def get_kok_discounted_products(db, page=1, size=20, use_cache=True):
    # 1. 캐시에서 데이터 조회 시도
    if use_cache:
        cached_data = cache_manager.get('discounted_products', page=page, size=size)
        if cached_data:
            return cached_data  # 캐시 히트 - DB 접근 없음

    # 2. 캐시 미스 - DB에서 데이터 조회
    products = await fetch_from_database(db, page, size)

    # 3. 조회한 데이터를 캐시에 저장
    if use_cache:
        cache_manager.set('discounted_products', products, page=page, size=size)

    return products
```

데이터베이스 인덱스 최적화

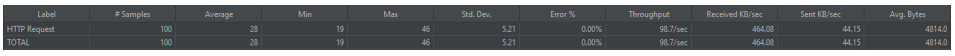
```
-- 1. 복합 인덱스 (자주 함께 사용되는 컬럼)
CREATE INDEX idx_kok_price_discount_rate_product_id
ON FCT_KOK_PRICE_INFO (KOK_DISCOUNT_RATE, KOK_PRODUCT_ID);

-- 2. 조건부 인덱스 (WHERE 절 최적화)
CREATE INDEX idx_kok_price_discount_rate_gt_zero
ON FCT_KOK_PRICE_INFO (KOK_DISCOUNT_RATE)
WHERE KOK_DISCOUNT_RATE > 0;

-- 3. 정렬 최적화 인덱스
CREATE INDEX idx_kok_product_review_cnt_score
ON FCT_KOK_PRODUCT_INFO (KOK_REVIEW_CNT, KOK_REVIEW_SCORE);
```

▼ 레시피 API `/api/recipes/by-ingredients`

Number of Threads : 100

Ramp-up period : 1

Loop Count : 1



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	100	28	19	46	5.21	0.00%	98.7/sec	464.08	44.15	4814.0
TOTAL	100	28	19	46	5.21	0.00%	98.7/sec	464.08	44.15	4814.0

구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	100	28	19	46	0.00%	98.7/sec

▼ 성능 최적화 방법

데이터베이스 쿼리 최적화

최적화 내용:

- 단일 쿼리로 후보 레시피 조회: N+1 쿼리 문제 해결
- IN 절을 사용한 재료 필터링: Material.material_name.in_(ingredients)

- GROUP BY와 ORDER BY 최적화: 인기순 정렬 (desc(Recipe.scrap_count))

최적화 내용:
단일 쿼리로 후보 레시피 조회: N+1 쿼리 문제 해결
IN 절을 사용한 재료 필터링: Material.material_name.in_(ingredients)
GROUP BY와 ORDER BY 최적화: 인기순 정렬 (desc(Recipe.scrap_count))

▼ 레시피 API `/api/recipes/search`

Number of Threads : 100

Ramp-up period : 1

Loop Count : 1

구분 (Label)	샘플 수 (# Samples)	평균 응답시간 (ms)	최소 (ms)	최대 (ms)	오류율 (Error %)	처리량 (Throughput)
TOTAL (합계)	100	33	20	82	0.00%	99.0/sec

▼ 성능 최적화 방법

병렬 처리 도입

```
# 기존: 순차 실행 (느림)
exact_df = await search_exact_matches()  # 300ms
vec_ids = await search_vector_matches()  # 200ms
# 총 500ms

# 최적화: 병렬 실행 (빠름)
exact_df, vec_ids = await asyncio.gather(
    search_exact_matches(),    # 300ms
    search_vector_matches()    # 200ms
)
# 총 300ms (동시 실행)
```

데이터베이스 쿼리 최적화

```
-- 추가된 인덱스들
CREATE INDEX idx_recipe_cooking_name ON FCT_RECIPE(COOKING_NAME);
CREATE INDEX idx_material_name ON FCT_MTRL(MATERIAL_NAME);
CREATE INDEX idx_recipe_scrap_count ON FCT_RECIPE(SCRAP_COUNT DESC);
```

N+1 쿼리 문제 해결

```
# 기존: N+1 문제 (느림)
for recipe_id in recipe_ids:
    materials = await get_materials(recipe_id)  # 개별 쿼리

# 최적화: 배치 쿼리 (빠름)
materials = await get_materials_batch(recipe_ids)  # 한 번에 조회
```

페이지네이션 최적화

```
# 기존: 전체 데이터 조회 후 슬라이싱
all_ids = await get_all_recipe_ids()  # 10만개 조회
page_ids = all_ids[start:end]         # 15개만 사용
```

```
# 최적화: 필요한 만큼만 조회
page_ids = await get_recipe_ids_with_pagination(start, end)  # 15개만 조회
```

캐시 시스템 강화

```
# 기존: 기본 캐시만
cache = SimpleLRUCache(max_size=200)

# 최적화: 검색 전용 캐시 추가
cache = SimpleLRUCache(max_size=500)
search_cache = SimpleLRUCache(max_size=300, ttl=900)  # 15분 TTL
```