엘라스틱 영화 검색 사이트 기능 설명

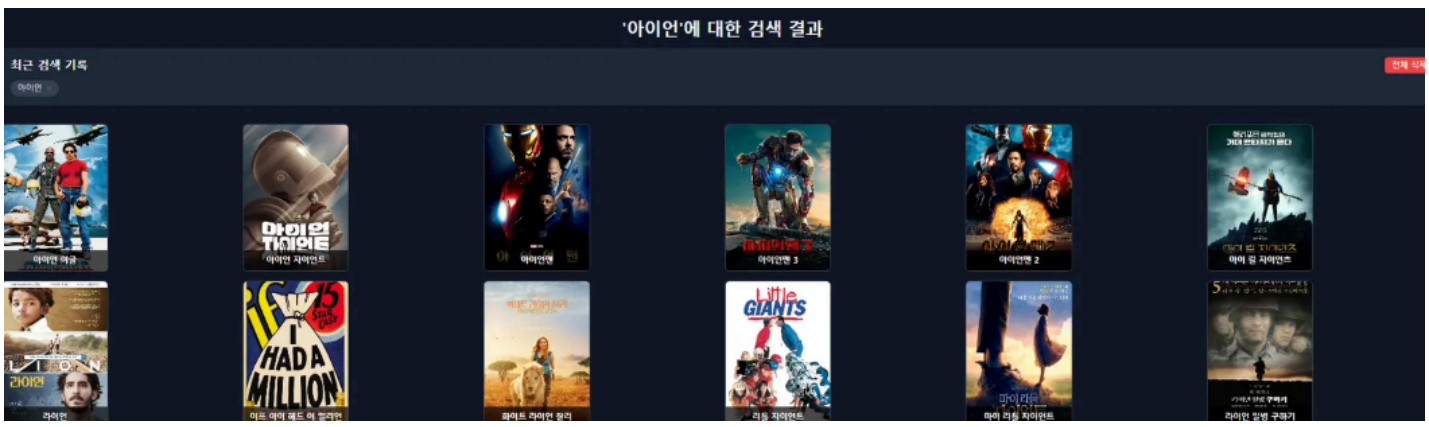
개요

프로젝트 목적 및 간단 설명

- **영화 탐색부터 추천, 리뷰 요약, 예매 정보 확인까지 하나의 흐름으로 제공하는 통합 영화 플랫폼으로, Elasticsearch 기반 검색과 퀵매치 추천을 통해 사용자가 원하는 영화를 빠르고 명확하게 선택 가능**

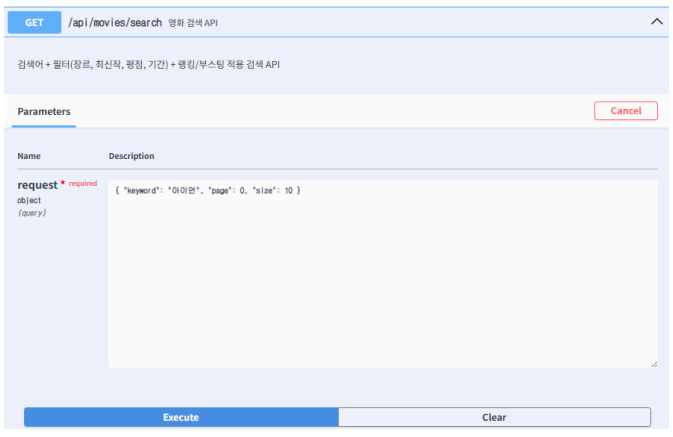
주요 기능 구현

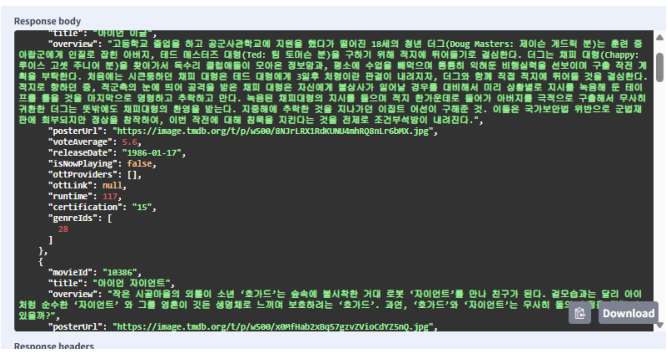
영화 검색 API



- 키워드 입력 시 연관 영화 자동완성 제공
- 키워드 선택 또는 검색 실행 시 관련 영화 목록 조회
- 검색 결과를 카드 그리드 형태로 시각화하여 직관적인 탐색 지원

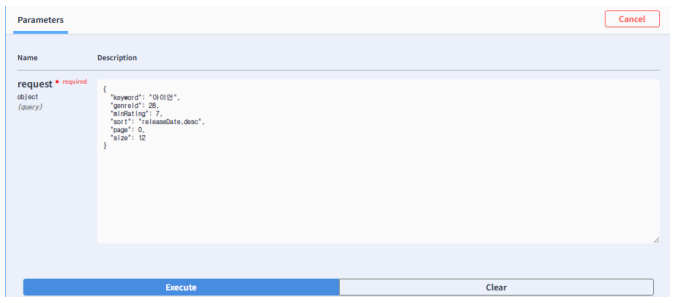
- **Swagger**





> 💡 **검색 조건을 기반으로 영화 목록을 조회하는 검색 API를 구현하고, Swagger로 요청·응답을 검증함**

필터 옵션 API



💡 **검색 화면에서 사용할 필터 옵션을 제공하는 API를 구현하고, Swagger로 응답 구조를 검증함**

영화 리뷰

- **AI 리뷰 요약**

- 내부 리뷰와 외부 리뷰를 종합해 AI 기반 리뷰 요약 생성
- 긍정·부정·중립 비율을 함께 제공하여 영화 평가를 한눈에 파악 가능

- **외부 리뷰 연동 API**

- TMDB API를 통해 외부 영화 리뷰를 수집하여 내부 리뷰와 함께 제공
- 영어 리뷰는 번역 후 DB에 저장하여 한글 리뷰로 제공

- **관련 코드**

```
@Configuration  amy5664
@EnableScheduling
public class SchedulerConfig {  //스케줄러 켜기
}
```

스케줄러 활성화로 요약/번역/캐시 데이터를 주기적으로 갱신하는 배치 작업 기반을 구성

```
/**
 * 30초마다 번역 안 된 TMDB 리뷰 최대 100개씩 번역
 */
@Scheduled(fixedDelay = 30000)  amy5664
public void translatePendingReviews() {
    List<TmdbReviewEntity> pending =
            tmdbReviewRepository.findTop100ByTranslatedContentIsNullOrderByCreatedAtAsc();

    if (pending.isEmpty()) {
        return;
    }

    log.info("번역 대기 TMDB 리뷰 {}건 처리 시작", pending.size());

    for (TmdbReviewEntity e : pending) {
        String translated = reviewTranslationService.translateToKorean(e.getOriginalContent());
        if (translated == null) {
            log.warn("번역 실패: reviewId={}, movieId={}", e.getId(), e.getMovieId());
            continue;
        }

        e.setTranslatedContent(translated);
        e.setUpdatedAt(LocalDateTime.now());
        tmdbReviewRepository.save(e);
    }
```

TMDB 리뷰 중 번역 안 된 데이터만 30초마다 최대 100개씩 자동 번역
번역 결과를 DB에 저장해 이후 조회 시 즉시 한국어 리뷰 제공(캐싱)

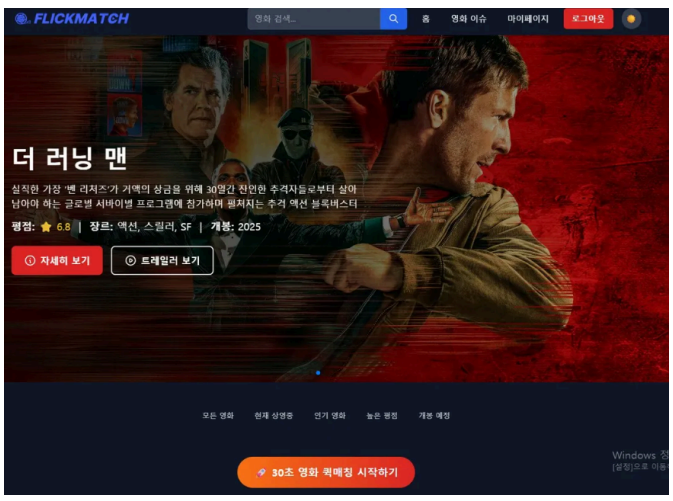
```java
// 리뷰 요약만 반환
@Transactional  1개 사용 위치  amy5664
public ReviewSummaryDto getSummaryOnly(String movieId) {

    // 0) 기존 요약 있으면 바로 리턴
    return movieReviewSummaryRepository.findByMovieId(movieId) Optional<MovieReviewSummary>
            .map(this::toDto) Optional<ReviewSummaryDto>
            .orElseGet(() -> {
                try {
                    // 요약 생성 시도
                    return createSummary(movieId);
                } catch (DataIntegrityViolationException e) {
                    // 누군가 먼저 INSERT 했을 때 여기로 떨어짐
                    return movieReviewSummaryRepository.findByMovieId(movieId) Optional<MovieReviewSummary>
                            .map(this::toDto) Optional<ReviewSummaryDto>
                            .orElseThrow(() -> e);
                }
            });
}
```

DB에 요약이 있으면 즉시 반환, 없으면 createSummary(movieId)로 생성
동시에 여러 요청이 들어와도 중복 INSERT 발생 시 예외 잡고 다시 조회해서 반환

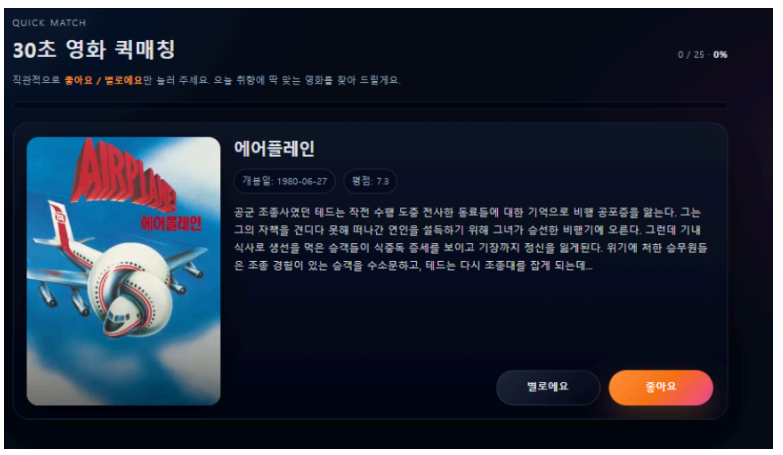
```java
private String callLlm(String prompt) {  1개 사용 위치  amy5664
    long start = System.currentTimeMillis();
    try {
        Map<String, Object> body = new HashMap<>();

        body.put("model", "gpt-4o-mini");
        body.put("response_format", Map.of( k1: "type",  v1: "json_object"));

        List<Map<String, String>> messages = List.of(
                Map.of( k1: "role",  v1: "system",
                        k2: "content",  v2: "너는 영화 리뷰를 한국어로 요약하고 감정을 분석하는 어시스턴트다. 반드시 유효한 JSON만 출력한다."),
                Map.of( k1: "role",  v1: "user",  k2: "content", prompt)
        );
        body.put("messages", messages);

        JsonNode response = openAiClient.post() RequestBodyUriSpec
                .uri( uri: "/chat/completions") RequestBodySpec
                .header( headerName: "Authorization",  ...headerValues: "Bearer " + openAiApiKey)
                .header( headerName: "Content-Type",  ...headerValues: "application/json")
                .body(body)
                .retrieve() ResponseSpec
                .body(JsonNode.class);
```

OpenAI로 리뷰를 요약하고 감성 비율(긍/부정/중립)을 JSON 형태로 생성
JSON 출력 강제로 파싱 오류를 줄이고 DB 저장/재사용을 안정화

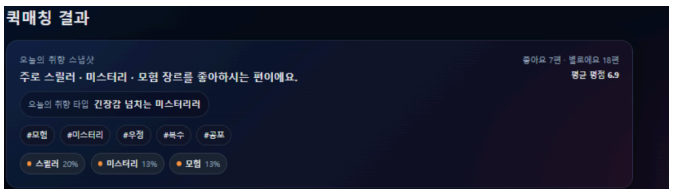
퀵매치 추천

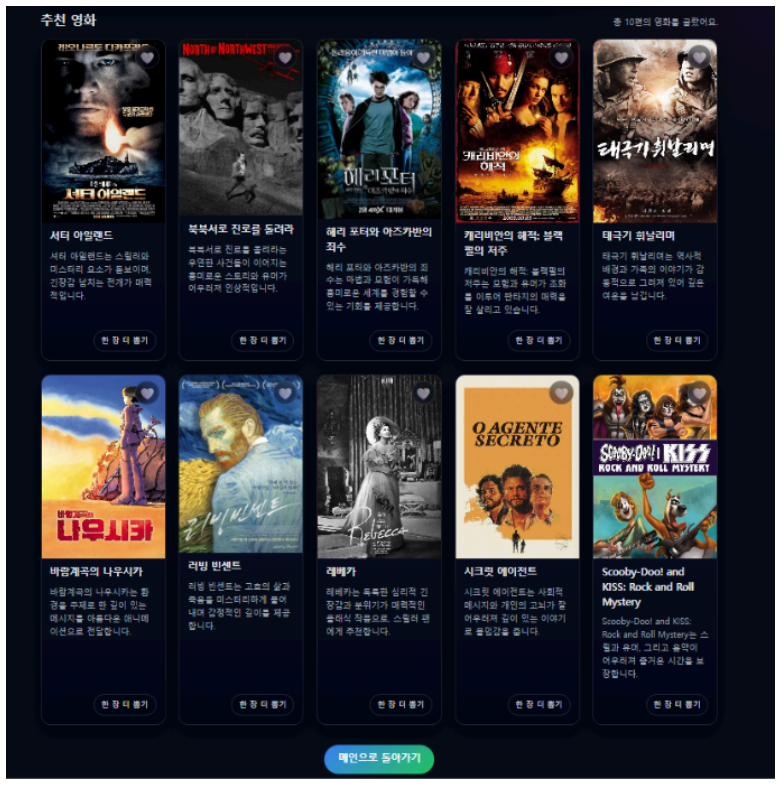


- 퀵매치 버튼 클릭 →



- 좋아요/별로예요 선택에 따라 다음 추천과 진행도가 실시간으로 반영





- 사용자 선택을 기반으로 분석된 취향 요약 및 선호 장르 시각화
- 분석된 취향을 바탕으로 추천 영화 목록을 카드 그리드 형태로 제공
- 찜하기 및 한 장 더 뽑기 기능으로 추천 결과를 빠르게 조정 가능

- **관련 코드**

```java
public class QuickMatchSession {

    @Id
    private String id;      // UUID 문자열 그대로 저장

    @Column(name = "user_id", nullable = false)
    private Long userId;    // users.id FK

    @Column(name = "target_count", nullable = false)
    private Integer targetCount;   // 목표 평가 개수 (예: 25)

    @Column(name = "rated_count", nullable = false)
    private Integer ratedCount;    // 지금까지 평가한 개수

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private SessionStatus status;  // IN_PROGRESS / COMPLETED
```

퀵매치는 한 번의 플레이를 세션으로 관리하며, 목표 평가 수(targetCount)와 진행도(ratedCount)를 기준으로 진행 상태를 판단

```java
// 기존 진행 중 세션 있으면 종료 처리
sessionRepository.findFirstByUserIdAndStatus(
        userId,
        QuickMatchSession.SessionStatus.IN_PROGRESS
).ifPresent( QuickMatchSession s -> {
    s.setStatus(QuickMatchSession.SessionStatus.COMPLETED);
    s.setCompletedAt(LocalDateTime.now());
    sessionRepository.save(s);
});

QuickMatchSession session = QuickMatchSession.builder()
        .id(UUID.randomUUID().toString())
        .userId(userId)
        .targetCount(targetCount != null ? targetCount : 25) // 기본 25개
        .ratedCount(0)
        .status(QuickMatchSession.SessionStatus.IN_PROGRESS)
        .createdAt(LocalDateTime.now())
        .build();
```

새로 시작 시 기존 진행 중 세션을 종료하고, UUID 기반의 새로운 세션(기본 25개 평가)을 생성

```
boolean existsBySessionIdAndMovieId(  1개 사용 위치   amy5664
        String sessionId,
        String movieId
);
```

```
// 중복 피드백 체크 (세션 + 영화 기준)
if (!feedbackRepository.existsBySessionIdAndMovieId(sessionId, movieId)) {
    QuickMatchFeedback feedback = QuickMatchFeedback.builder()
            .sessionId(sessionId)
            .userId(userId)
            .movieId(movieId)
            .action(action)
            .createdAt(LocalDateTime.now())
            .build();

    feedbackRepository.save(feedback);

    // 처음 보는 영화일 때만 카운트 증가
    session.setRatedCount(session.getRatedCount() + 1);
}

// 목표 개수에 도달하면 세션 종료
if (session.getRatedCount() >= session.getTargetCount()) {
    session.setStatus(QuickMatchSession.SessionStatus.COMPLETED);
    session.setCompletedAt(LocalDateTime.now());
}
```

세션 + 영화 기준 중복 평가를 방지하여 동일 영화에 대한 반복 클릭으로 진행도 증가하지 않도록 처리
목표 평가 수 도달 시 세션을 COMPLETED로 전환하여 결과 화면 조건 충족

```
// 여기서 한 번만 AI 호출해서 reason 리스트 받아오기
List<String> reasons = aiRecommendationService.generateReasons(summaryDto, selected);
```

```
Set<String> excludeIds = new HashSet<>(seenMovieIds);
excludeIds.add(currentMovieId);

List<MovieDoc> filtered = candidates.stream()
        .filter( MovieDoc m -> m.getMovieId() != null)
        .filter( MovieDoc m -> !excludeIds.contains(m.getMovieId()))
        .collect(Collectors.toList());
```

추천 카드에는 AI가 생성한 '추천 이유 1문장'을 함께 제공
한 장 더 뽑기'는 전체 추천을 다시 생성하지 않고, 현재 카드만 교체하며 이미 본 영화와 현재 영화는 제외