MovieRec

서울 4반 1조 안혜성, 원기훈

MovieRec은 영화를 좋아하는 사람들을 위한 혁신적인 영화 추천 및 커뮤니티 플랫폼입니다. 사용자 경험을 기반으로 맞춤형 영화 정보와 활발한 커뮤니티를 제공합니다.

K **작성자: Kee Hoon Won**

업무 분담

안혜성

- 데이터 수집 및 가공
- 서버 로직 및 API 구현
- 인증, 비즈니스 로직 전반 구현
- 추천 알고리즘 개발
- accounts, community 화면 개발

원기훈

- 서비스 기획
- 일정 관리
- movie, layout 화면 개발
- 프로젝트 빌드 / 의존성 관리

프로젝트 개요

1. **서비스 기획:** MovieRec 핵심 기능 및 사용자 경험 설계
2. **추천 알고리즘:** 개인화된 영화 추천 시스템 개발
3. **시연:** 주요 기능 시연
4. **Q&A:** 질의응답 세션

서비스 설명

영화 정보 제공

다양한 영화에 대한 상세 정보와 리뷰를 제공합니다.

개인화 추천

사용자의 취향과 활동을 분석하여 맞춤형 영화를 추천합니다.

커뮤니티 기능

영화 팬들과 소통하고 의견을 나눌 수 있는 공간을 제공합니다.

사용자 경험 중심

직관적인 인터페이스와 편리한 기능으로 최적의 사용자 경험을 제공합니다.

서비스 구조

카테고리	화면 이름	주요 기능
Accounts	로그인(초기 화면)	- 자체 로그인 (이메일/비밀번호) - 소셜 로그인 (Google, Facebook, Kakao, Naver)
	회원가입	- 이메일 인증 - 비밀번호 유효성 검사 - 선호 장르 선택
	프로필 (나, 다른 사람)	- 사용자 정보 조회 - 좋아요/팔로우 정보 표시 - 친구 목록 조회
	회원정보 수정	- 사용자 닉네임 및 프로필 이미지 수정 - 선호 장르 수정
	알림 페이지	- 댓글, 팔로우, 좋아요 등 사용자 활동 알림 표시
	Admin 페이지	- 영화, 사용자, 리뷰 데이터 수정 가능

서비스 구조 (계속)

카테고리	화면 이름	주요 기능
Movies	영화 목록 페이지(메인 화면)	- 인기 영화 표시 - 사용자 맞춤 영화 추천 렌더링 - 검색 필터링 (장르, 평점, 연도)
	영화 검색 결과 페이지	- 제목, 배우, 감독, 장르, 관람 등급 등 필터링
	영화 상세 페이지	- 영화 정보 (줄거리, 출연진, 감독) - 영화 트레일러 - 좋아요 버튼, 관련 영화 추천 - 관련 (게시글, 기사, 한줄평) - 배우 정보 표시 (이름, 사진, 배우 프로필, 관련 영화) - 즐겨찾기 추가/삭제 기능 - 감독 정보 표시 (이름, 사진, 감독 프로필, 연출작) - 즐겨찾기 추가/삭제 기능

서비스 구조 (Community)

카테고리	화면 이름	주요 기능
Community	자유게시판(목록, 팔로우 중인 사람의 게시글 목록, 상세)	- 게시글 CRUD, 좋아요 기능 - 최신 글 목록 표시, 인기 글 목록 표시 - 커뮤니티 레벨 시스템 도입
	리뷰게시판(목록, 팔로우 중인 사람의 리뷰 목록, 상세)	- 리뷰 좋아요 기능 - 영화 리뷰 CRUD - 리뷰 신고 버튼(사이버수사대로 링크 달기) - CRUD 댓글 CRUD, 대댓글
	게시판 검색	- 게시판 제목, 내용, 작성자 기준 검색
Search	실시간 검색	- 영화 제목, 배우, 감독, 유저, 게시판 제목, 검색

기술 스택

Frontend

React를 사용한 동적이고 반응형 웹 인터페이스 구현

Backend

Django를 활용한 강력하고 안전한 서버 로직 구현

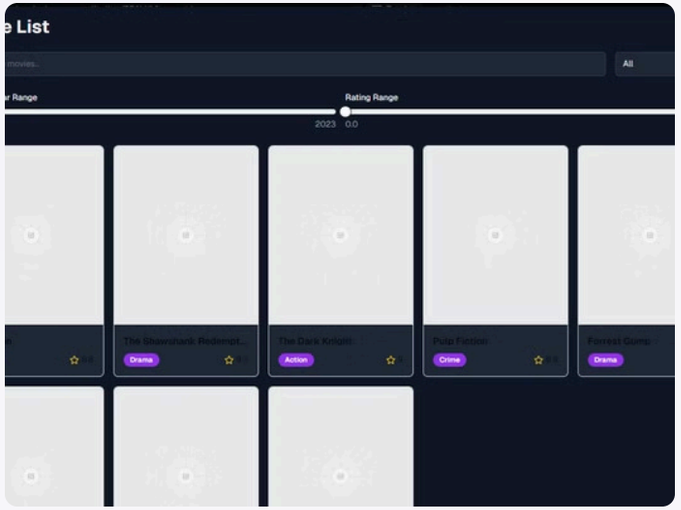
협업 도구

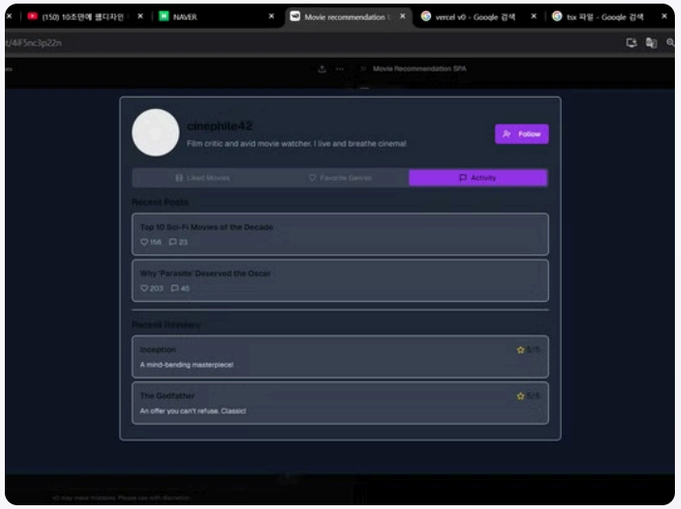
Notion과 Mattermost를 통한 효율적인 협업 및 커뮤니케이션

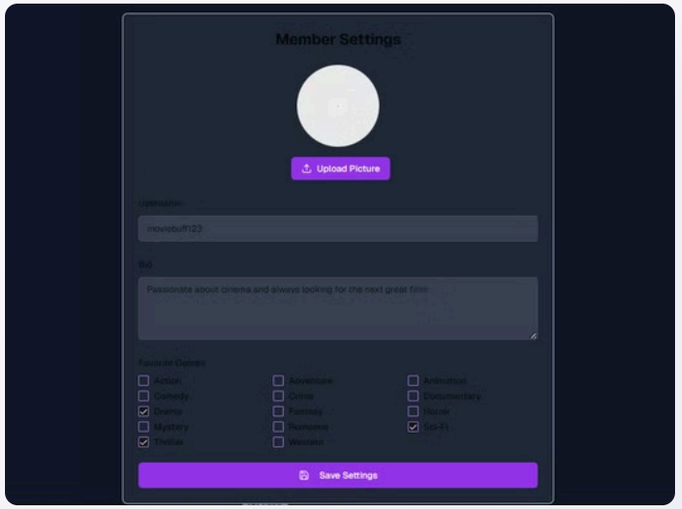
생성형 AI

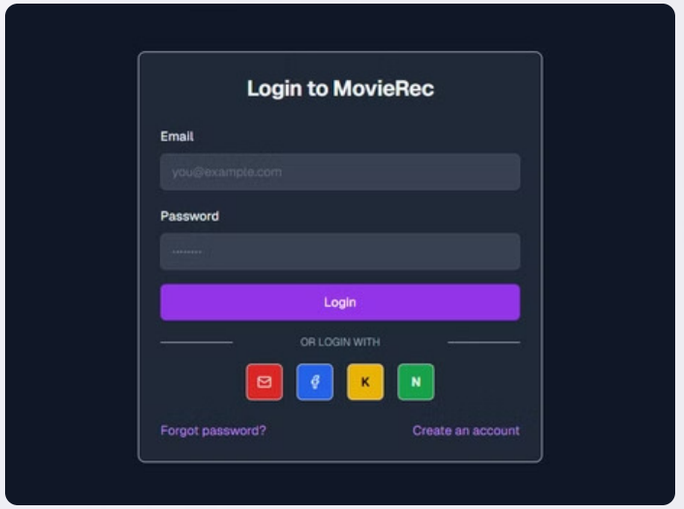
최신 AI 기술을 활용한 혁신적인 기능 구현

프로토타입



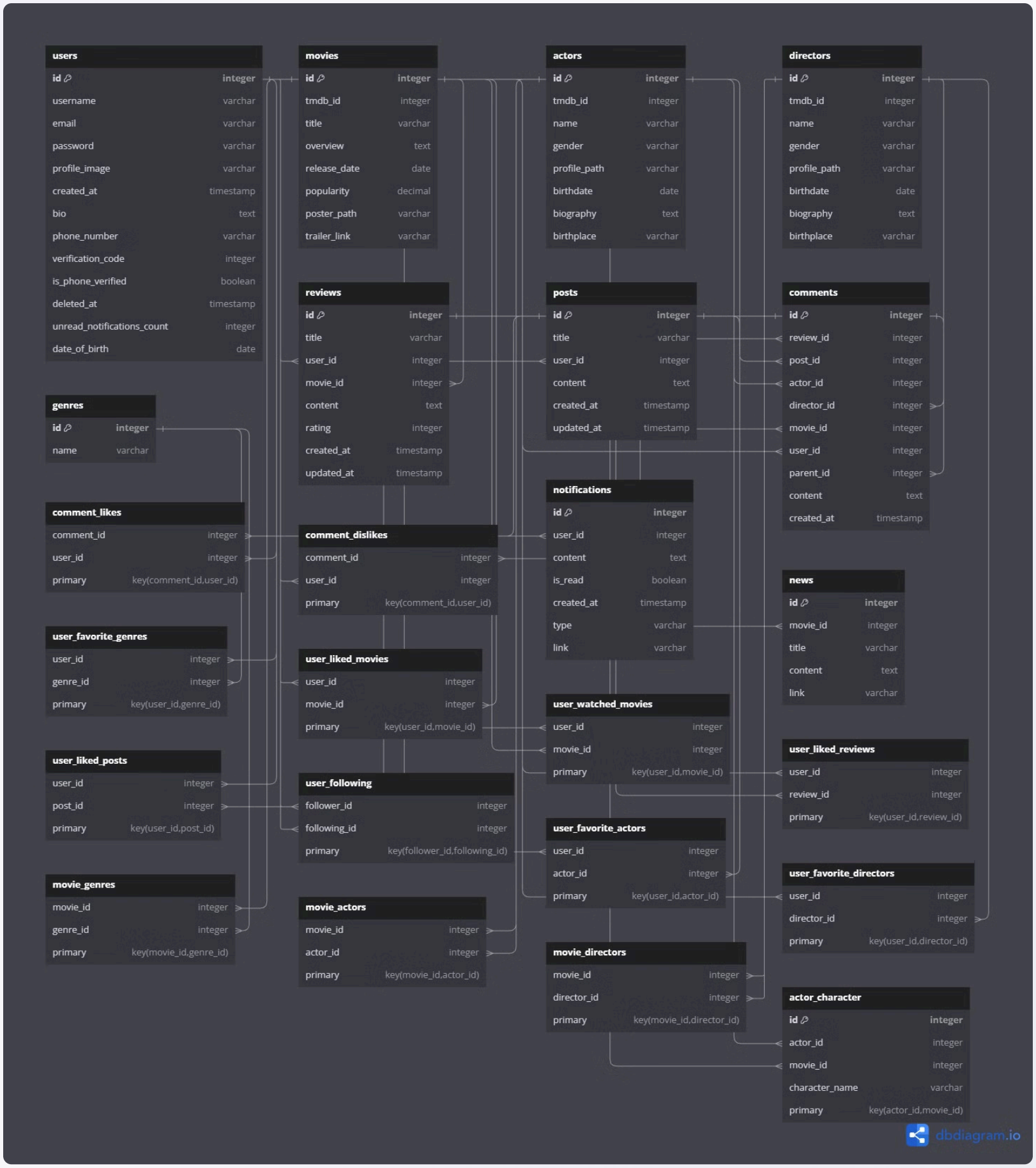






MovieRec의 초기 프로토타입 디자인입니다. 사용자 인터페이스의 주요 요소와 레이아웃을 보여줍니다.

데이터베이스 ERD



MovieRec 서비스의 데이터베이스 구조를 보여주는 ERD(Entity-Relationship Diagram)입니다. 주요 엔티티와 그들 간의 관계를 시각화하여 데이터 모델을 명확히 표현하고 있습니다.

API 엔드포인트

API	HTTP 메서드	URL	관련 모델	주요 필드	화면 기능
로그인	POST	/dj-rest-auth/login/	User	username, password	
로그아웃	POST	/dj-rest-auth/logout/	-	-	사용자가 계정을 로그아웃
회원가입	POST	/dj-rest-auth/registration/	User	username, email, password, favorite_genres	
비밀번호 변경	POST	/dj-rest-auth/password/change/	User	old_password, new_password	비밀번호를 변경
영화 목록 조회	GET	/api/movies/	Movie, Genre	title, poster_path, popularity, genres	

API 엔드포인트 (계속)

API	HTTP 메서드	URL	관련 모델	주요 필드	화면 기능
영화 상세 조회	GET	/api/movies/<id>/	Movie, Actor, Director	title, overview, release_date, popularity, poster_path, trailer_link	
배우 프로필 조회	GET	/api/actors/<id>/	Actor	name, profile_path, birthdate, biography, birthplace, movies	
감독 프로필 조회	GET	/api/directors/<id>/	Director	name, profile_path, birthdate, biography, birthplace, movies	
게시물 작성	POST	/api/posts/	Post	content, created_at, user	커뮤니티 게시물 작성
게시물 목록 조회	GET	/api/posts/	Post	content, created_at, user	

게시물 수정	PUT/PATCH	/api/posts/<id>/	Post	content, updated_at	특정 게시물 수정	
게시물 삭제	DELETE	/api/posts/<id>/	Post	-	특정 게시물 삭제	
리뷰 작성	POST	/api/reviews/	Review	movie, content, rating, created_at, user	영화에 대한 리뷰 작성	
리뷰 목록 조회	GET	/api/reviews/	Review	movie, content, rating, created_at, user	리뷰 목록 표시	
리뷰 수정	PUT/PATCH	/api/reviews/<id>/	Review	content, rating, updated_at	특정 리뷰 수정	
리뷰 삭제	DELETE	/api/reviews/<id>/	Review	-	특정 리뷰 삭제	
댓글 작성	POST	/api/comments/	Comment	review 또는 post, content, created_at, user	댓글/대댓글 작성	
댓글 목록 조회	GET	/api/comments/	Comment	content, created_at, user, parent	댓글 목록 조회	
댓글 삭제	DELETE	/api/comments/<id>/	Comment	-	특정 댓글 삭제	
알림 목록 조회	GET	/api/notifications/	Notification	content, is_read, type, link, created_at	사용자 활동 알림 표시	

추천 알고리즘

1

사용자 정보 기반

사용자의 선호 장르, 좋아요한 영화, 즐겨찾기한 배우와 감독 정보를 활용하여 개인화된 추천을 제공합니다.

2

키워드 기반 추천

사용자가 입력한 키워드를 기반으로 영화 제목과 줄거리의 유사도를 계산하여 관련 영화를 추천합니다.

3

최적화 및 캐싱

임베딩 캐싱과 배치 처리를 통해 추천 알고리즘의 성능을 최적화합니다.

4

API 엔드포인트

개인화 추천과 키워드 기반 추천을 하나의 API 엔드포인트로 통합하여 제공합니다.

사용자 정보 기반 추천 알고리즘

사용자의 선호 장르, 좋아요한 영화, 즐겨찾기한 배우와 감독 정보를 활용하여 개인화된 영화 추천을 제공합니다. 아래는 알고리즘의 주요 함수들을 보여줍니다.

장르 유사도 계산

```
def calculate_genre_similarity(movie, user):
    """ 사용자 선호 장르와 영화 장르 간의 유사도를 계산합니다. """
    user_genres = set(user.favorite_genres.all())
    movie_genres = set(movie.genres.all())
    common_genres = user_genres.intersection(movie_genres)
    return len(common_genres) / len(user_genres) if user_genres else 0
```

좋아요한 영화 유사도 계산

```
def calculate_liked_movies_similarity(movie, user):
    """ 사용자가 좋아요한 영화와 현재 영화의 장르 유사도를 계산합니다. """
    liked_movies = user.liked_movies.all()
    similarities = [1 for lm in liked_movies if lm.genres.filter(id__in=movie.genres.values_list('id', flat=True)).exists()]
    return sum(similarities) / len(liked_movies) if liked_movies else 0
```

배우 유사도 계산

```
def calculate_actor_similarity(movie, user):
    """ 사용자가 즐겨찾기한 배우가 출연한 영화인지 확인합니다. """
    favorite_actors = set(user.favorite_actors.all())
    movie_actors = set(movie.actor_movies.all())
    common_actors = favorite_actors.intersection(movie_actors)
    return len(common_actors) / len(favorite_actors) if favorite_actors else 0
```

감독 유사도 계산

```
def calculate_director_similarity(movie, user):
    """ 사용자가 즐겨찾기한 감독이 연출한 영화인지 확인합니다. """
    favorite_directors = set(user.favorite_directors.all())
    movie_directors = set(movie.director_movies.all())
    common_directors = favorite_directors.intersection(movie_directors)
    return len(common_directors) / len(favorite_directors) if favorite_directors else 0
```

친구 활동 유사도 계산

```
def calculate_friend_activity(movie, user):
    """ 팔로우한 친구들이 좋아요한 영화와의 유사도를 계산합니다. """
    friends = user.following.all()
    friend_liked_movies = set(Movie.objects.filter(liked_movies__in=friends).distinct())
    return 1 if movie in friend_liked_movies else 0
```

개인화 추천 함수

```
def personalized_recommendations(user):
    """ 사용자 개인화 추천 리스트를 생성합니다. """
    movies = Movie.objects.prefetch_related('genres', 'actor_movies', 'director_movies').all()
    recommendations = []
    for movie in movies:
        genre_score = calculate_genre_similarity(movie, user)
        liked_movie_score = calculate_liked_movies_similarity(movie, user)
        actor_score = calculate_actor_similarity(movie, user)
        director_score = calculate_director_similarity(movie, user)
        friend_score = calculate_friend_activity(movie, user)

        total_score = (0.3 * genre_score + 0.2 * liked_movie_score + 0.2 * actor_score + 0.2 * director_score + 0.1 *
friend_score)
        recommendations.append({"movie": movie, "score": total_score})

    recommendations = sorted(recommendations, key=lambda x: x["score"], reverse=True)
    return recommendations[:10]
```

키워드 기반 추천 알고리즘

사용자가 입력한 키워드를 기반으로 영화 제목과 줄거리의 유사도를 계산하여 관련 영화를 추천합니다. 임베딩과 코사인 유사도를 사용하여 효율적으로 추천합니다.

영화 임베딩 가져오기 (캐싱 포함)

```
movie_embeddings_cache = {}

def get_movie_embeddings(movie):
    """ 영화의 제목과 설명 임베딩을 반환합니다. 캐싱을 사용하여 중복 계산을 방지합니다. """
    if movie.id not in movie_embeddings_cache:
        movie_embeddings_cache[movie.id] = {
            "title_embedding": model.encode(movie.title),
            "overview_embedding": model.encode(movie.overview or "")
        }
    return movie_embeddings_cache[movie.id]
```

키워드 기반 추천 함수 (최적화)

```
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

def keyword_based_recommendations_optimized(keyword):
    """ 최적화된 키워드 기반 영화 추천을 수행합니다. """
    movies = Movie.objects.prefetch_related('genres').all()
    keyword_embedding = model.encode(keyword)

    title_embeddings = []
    overview_embeddings = []
    movie_list = []

    for movie in movies:
        embeddings = get_movie_embeddings(movie)
        title_embeddings.append(embeddings["title_embedding"])
        overview_embeddings.append(embeddings["overview_embedding"])
        movie_list.append(movie)

    title_similarities = cosine_similarity([keyword_embedding], title_embeddings)[0]
    overview_similarities = cosine_similarity([keyword_embedding], overview_embeddings)[0]

    recommendations = []
    for i, movie in enumerate(movie_list):
        total_score = 0.6 * title_similarities[i] + 0.4 * overview_similarities[i]
        recommendations.append({"movie": movie, "score": total_score})

    return sorted(recommendations, key=lambda x: x["score"], reverse=True)[:10]
```

시연

MovieRec의 주요 기능들을 실제로 시연하는 세션입니다. 사용자 등록부터 영화 추천, 커뮤니티 참여까지 전체적인 사용자 경험을 보여드리겠습니다.

Q&A

프로젝트에 대한 질문과 답변 세션입니다. MovieRec의 기능, 개발 과정, 향후 계획 등에 대해 자유롭게 질문해 주시기 바랍니다.