2024-2 종합설계 PBL 최종보고서

OTT 컨텐츠 정보 요약 프로젝트

팀명 : 무한불성

오찬빈
김윤정
이윤기
박성원

목차

1. 프로젝트 명 ····· 3

2. 요약 ····· 3

3. 서론 ····· 3

3.1 추진 배경 및 목적 ····· 3 , 4

3.2 개발 목표와 내용 ····· 4 , 5

4. 설계 및 구현

4.1 리뷰 데이터 수집 ····· 6 - 10

4.1 데이터 베이스 ····· 11 - 15

4.1 웹/프론트엔드 ····· 16 - 24

5. 개발결과 ····· 25 - 31

6. 기대효과 ····· 31 - 34

7. 참고문헌 ····· 35

2024-2 종합설계 PBL 최종보고서

1. 프로젝트 명

- OTT 컨텐츠 정보 요약 웹 서비스

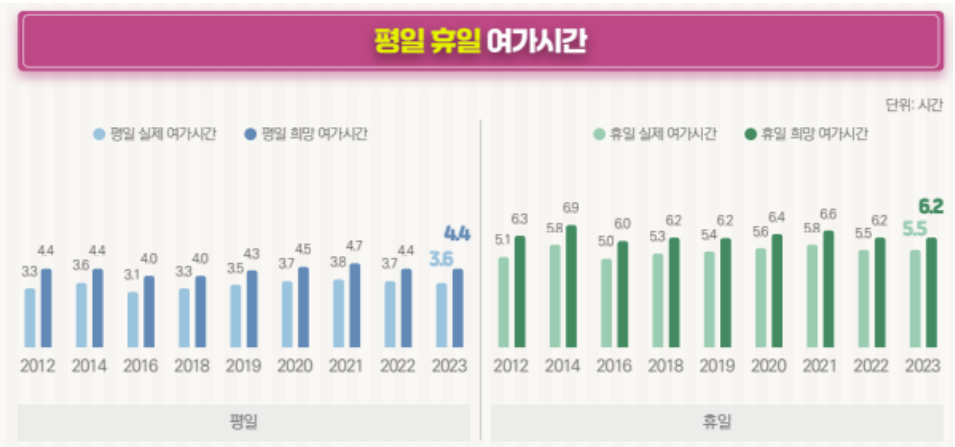
2. 프로젝트 요약

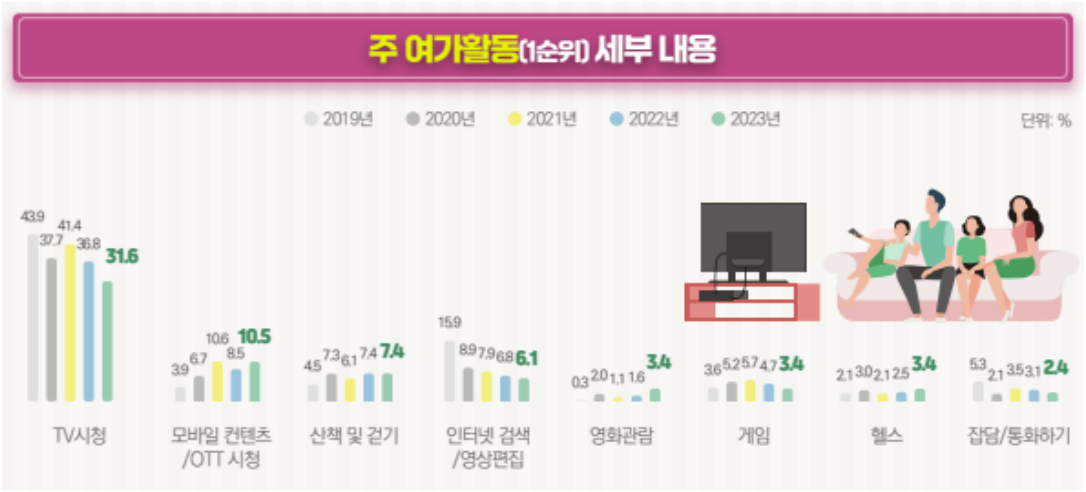
- 다양한 선택지가 있는 OTT 콘텐츠 중, 사용자가 콘텐츠를 선택하기 쉽도록 도움을 줄 수 있는 정보들을 요약하여 웹 사이트에서 볼 수 있도록 한다.
기본적인 영화 정보는 'TMDB API' 를 사용하고, 리뷰와 이외의 기타정보 등을 각각 수집하여 각 데이터에 맞는 테이블을 구성하여 반응형 웹으로 구현한다.

3. 서론

3. 1 추진 배경 및 목적

- 선택지가 많은 OTT 콘텐츠들이 때로는 현대인들에게 방해요소가 될 수 있다. 바쁜 현대사회에서 한정적인 여가 시간 속 효율적으로 본인이 원하는 콘텐츠를 접하기 위해서는 영화에 대한 모든 정보가 아닌, 필요한 정보만을 모아 효율적으로 콘텐츠를 선택하기 위해 본 프로젝트를 추진하게 되었다.





- 위 자료는 2023년 국민여가활동조사 (만 15세 이상의 남녀 10,000명을 대상으로 한 조사) 에서 '주 여가활동' 에 대한 답변을 그래프로 나타낸 것이다. 가장 많은 여가활동이 TV시청, 그 다음이 OTT 시청이며 OTT 시청은 우상향 하며 여가활동에 즐기는 수가 늘어나고 있는 추세인 점도 프로젝트트를 진행할 수 있는 배경이었다. (수요의 증가)

3. 2 개발 목표와 내용

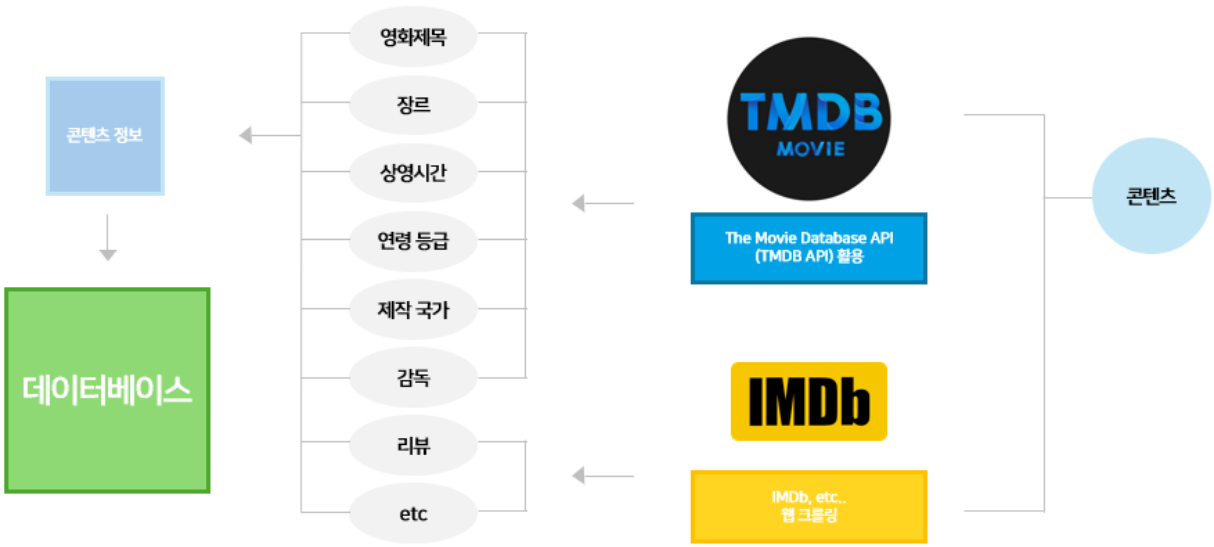
- 본 프로젝트는 'TMDB API' 를 통해 영화의 기본적인 내용에 해당되는 정보를 가져올 수 있도록 한다. (영화제목, 장르, 연령등급, 상영시간, etc..) 정리된 정보들을 각 데이터 테이블에 저장한 뒤, 반응형 웹 사이트를 구현하여 사용자에게 배포하는 것을 최종 목표이 주요 내용이다.

  이를 통해 OTT 컨텐츠를 즐기는 현대사회인들에게, 짧은 여가시간 속 효율적인 시간을 보내기 위한 수요를 충족할 수 있도록 하는 것이 최종목표이다.

3. 2 개발 목표와 내용

기능	내용
영화 정보 가져오기 (데이터, 리뷰 등)	영화제목, 장르 등의 영화의 '기본적 내용' 에 해당되는 정보는 TMDB API를 사용하여 정보를 가져온다. 리뷰와 이외의 기타정보는 IMDb 등에서 웹 크롤링을 활용하여 정보를 가져온다.
데이터베이스 구성	테이블을 추가로 만들어 M:N 관계를 1:N의 관계로 분리하여, 데이터의 무결성과 확장성을 챙긴다. 자주 호출될 것으로 예상되는 '장르목록' 테이블의 '장르'를 성 능을 고려하여 반정규화 하였다. (spring / Maven 활용)
웹 구성	영화 배너를 클릭하면 영화 정보를 출력 하도록 한다. 편의를 위한 검색 기능 구현과, 관심목록, 리뷰 작성 등의 기능을 구현한다.
회원가입 및 로그인, 사용자 관리	사용자를 구분짓기 위해 구현해야 한다.
기타	웹 사이트 로고 제작

4. 설계 및 구현



[영화 정보 가져오기 초안]

- TMDB API 를 통해 영화에 대한 정보들을 가져오도록 한다. (영화의 정보에는 영화제목, 장르, 상영시간, 연령등급, 제작국가 등이 포함된다.)

- IMDb 라는 웹 사이트에서 크롤링을 통해 해당 영화의 여러 리뷰와 이외 정보들을 가져온다.

4. 설계 및 구현

[리뷰 데이터 수집 코드]

```
# 중복 처리된 ID 로드
processed_ids = load_processed_ids()
```

- (자료1) 수집시 중복 처리된 ID를 로드하여, 인기 영화 추출 시 중복된 값은 제거

```
# 인기 영화 목록 가져오기
all_reviews = []
movie_ids = []
new_ids = set()  # 이번 실행에서 처리된 새로운 ID
print("인기 영화 imdb 추출 중...")
for page in range(1, 21):
    popular_movies = get_popular_movies(page)
    for movie in popular_movies:
        tmdb_id = movie['id']
        if str(tmdb_id) in processed_ids:
            print(f"Skipping already processed movie: https://www.themoviedb.org/movie/{tmdb_id}")
            continue

        imdb_id = get_imdb_id(tmdb_id)
        movie_ids.append((tmdb_id, imdb_id))
print("인기 영화 imdb 추출 완료.")

i = 1
for tmdb_id, imdb_id in movie_ids:
    if imdb_id:
        print(
            f'https://www.imdb.com/title/{imdb_id}/reviews/?ref_=tt_ov_ql_2&spoilers=EXCLUDE 리뷰 수집 중 ({i}/{len(movie_ids)})')
        reviews = get_reviews(imdb_id,  limit: 1)
        for review in reviews:
            review['tmdb_id'] = tmdb_id
        all_reviews.extend(reviews)
        new_ids.add(tmdb_id)
        i += 1
```

- (자료2) **TMDB API** 사용. API를 요청하여 IMDB ID (리뷰) 데이터를 추출

```
# DataFrame 생성 후 json 파일로 저장
if all_reviews:
    df = pd.DataFrame(all_reviews)
    df.to_json( path_or_buf: 'popular_movies_reviews.json', orient="records", force_ascii=False, indent=4)
    print("json 파일이 'popular_movies_reviews.json' 이름으로 저장되었습니다.")
else:
    print("추가된 리뷰가 없습니다.")
```

- (자료3) 수집한 리뷰는 json 파일로 저장하도록 한다.

4. 설계 및 구현

```
# 새로운 TMDb ID 저장
save_processed_ids(new_ids)
print("새로운 TMDb ID가 'processed_movie_ids.txt'에 저장되었습니다.")
```

- (자료4) 종료시 새로 처리한 ID를 저장하여 이후 중복 제거에 사용하도록 한다.
(자료 1 참조)

```
def get_reviews(imdb_id, iterate=0): # iterate 만큼 '더 보기' 버튼 클릭 후 추가 리뷰 수집  1개의 사용 위치  ▲ osb1916
    time.sleep(1)
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.97 Safari/537.36'
    }
    url = f'https://www.imdb.com/title/{imdb_id}/reviews/?ref_=tt_ov_ql_2&spoilers=EXCLUDE'
    response = requests.get(url, headers=headers)

    soup = BeautifulSoup(response.text, 'lxml')
    articles = soup.select('[data-testid="review-card-parent"]')
    reviews = []
```

- (자료5) IMDB에서 리뷰를 크롤링을 통해 가져올 수 있도록 하고, 사용된 함수는 (1) get_reviews() 와, (2) get_more_reviews() 를 사용. 과도한 respone을 막기 위한 리퀘스트 타임을 설정하였고, BeautifulSoup4 를 사용하기 위해 기초적인 변수와 URL을 설정할 수 있도록 한다.

```
# '__NEXT_DATA__' 스크립트 태그 찾기
script_tag = soup.find( name 'script', attrs {'id': '__NEXT_DATA__'})

# JSON 데이터를 로드
data = json.loads(script_tag.string)

# endCursor 값 추출
end_cursor = data['props']['pageProps']['contentData']['data']['title']['reviews']['pageInfo']['endCursor']

# 모든 리뷰 추출
all_reviews = data['props']['pageProps']['contentData']['data']['title']['reviews']['edges']
```

- (자료6) TMDB API 를 요청할 때, GraphQL 방식을 사용하며 리뷰페이지의 script, __NEXT_DATAT__에 현재 페이지의 모든 정보와 다음 페이지를 가리키는 파라미터를 가지고 있다. 따라서 이를 추출하여, 해당 페이지의 리뷰와 다음 페이지를 가르키는 파라미터를 추출할 수 있도록 한다.

4. 설계 및 구현

```
if not articles:
    print("There is no review or HTML structure might have changed.")
    return []
```

- (자료7) 해당 영화가 신작 등의 이유로 리뷰가 존재하지 않는 경우에 대한 예외처리를 한다.

```
for review in all_reviews:
    node = review['node']
    title = node['summary']['originalText']

    # rating 예외처리 (값이 없거나 None인 경우)
    rating = node['authorRating'] if node['authorRating'] is not None else 'No Rating'

    author = node['author']['nickName']
    date = node['submissionDate']
    body = node['text']['originalText']['plaidHtml']  # body에서 HTML 엔티티 변환
    tree = html.fromstring(body)
    cleaned_body = tree.text_content()  # HTML 엔티티를 변환
    helpful_yes = node['helpfulness']['upVotes']
    helpful_no = node['helpfulness']['downVotes']

    # 데이터 저장
    reviews.append({
        'title': title,
        'rating': rating,
        'author': author,
        'date': date,
        'body': cleaned_body,
        'helpful_yes': helpful_yes,
        'helpful_no': helpful_no
    })
```

- (자료8) 필요한 데이터를 수집 및 저장하도록 한다.

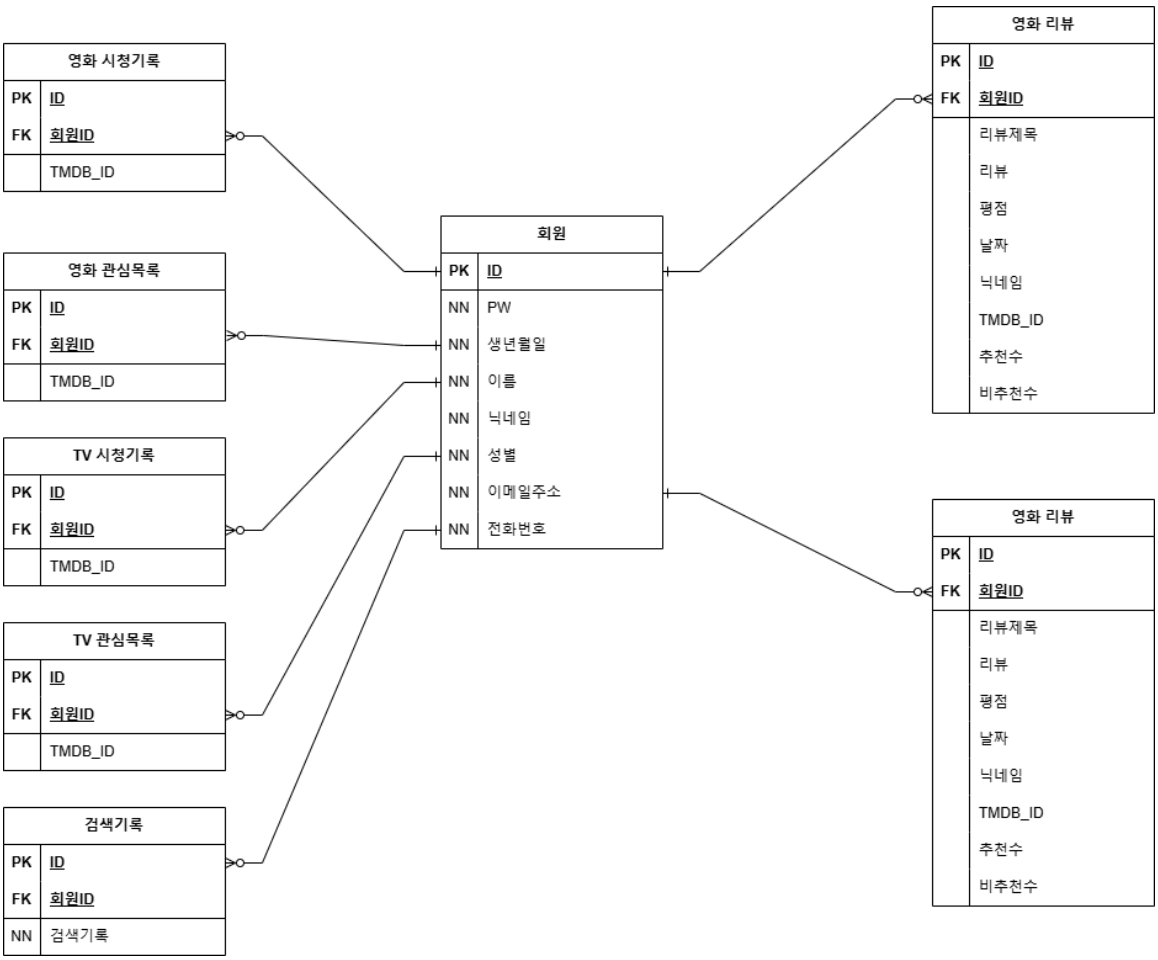
4. 설계 및 구현

```python
def get_more_reviews(imdb_id, end_cursor):  1개의 사용 위치  ocb3916
    time.sleep(1)
    url = "https://caching.graphql.imdb.com/"
    after = end_cursor
    payload = {
        "operationName": "TitleReviewsRefine",
        "variables": {
            "after": f"{after}", # 이 값은 DOM에서 endCursor 값을 찾아야 함
            "const": f"{imdb_id}",  # imdb-id
            "filter": {"spoiler": "EXCLUDE"},
            "first": 25,
            "locale": "ko-KR",
            "sort": {"by": "HELPFULNESS_SCORE", "order": "DESC"}
        },
        "extensions": {
            "persistedQuery": {
                "sha256Hash": "89aff4cd7503e060ff1dd5aba91885d8bac0f7a21aa1e1f781848a786a5bdc19",
                "version": 1
            }
        }
    }

    # POST 요청 실행
    headers = {"Content-Type": "application/json"}  # JSON 요청임을 명시
    response = requests.post(url, headers=headers, data=json.dumps(payload))

    # JSON 파싱
    parsed_data = response.json()

    # 리뷰 데이터 추출
    reviews = parsed_data['data']['title']['reviews']['edges']
    end_cursor = parsed_data['data']['title']['reviews']['pageInfo']['endCursor']

    # 결과 리스트
    extracted_data = []
```

- (자료9) IMDB에 GraphQL 방식의 요청문을 보내기 위한 payload이다. 해당 부분에 IMDB-ID와 자료6의 파라미터를 넣도록 한다. 이후 IMDB에 POST 요청을 실행하고, 결과로 다음 페이지 리뷰에 대한 json 형태의 데이터를 받는다. 이를 통해 파싱해서, 원하는 리뷰 데이터를 수집할 수 있도록 한다.

4. 설계 및 구현

[데이터베이스]



- (자료10) 데이터베이스 ERD 초안이다. 해당 초안을 토대로 제작했으며, spring / Maven 을 사용하였다.

4. 설계 및 구현

[데이터베이스]

```
DROP TABLE IF EXISTS `movie_favorite_list`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `movie_favorite_list` (
  `id` int NOT NULL AUTO_INCREMENT,
  `tmdb_id` bigint DEFAULT NULL,
  `user_id` varchar(15) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `FKf0t5onyuiuypb5qn0nv05kw2d` (`user_id`),
  CONSTRAINT `FKf0t5onyuiuypb5qn0nv05kw2d` FOREIGN KEY (`user_id`) REFERENCES `user` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `movie_favorite_list`
--

LOCK TABLES `movie_favorite_list` WRITE;
/*!40000 ALTER TABLE `movie_favorite_list` DISABLE KEYS */;
/*!40000 ALTER TABLE `movie_favorite_list` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `movie_review`
--

DROP TABLE IF EXISTS `movie_review`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `movie_review` (
  `id` int NOT NULL AUTO_INCREMENT,
  `tmdb_id` bigint DEFAULT NULL,
  `downvotes` int NOT NULL,
  `nickname` varchar(100) DEFAULT NULL,
  `rating` int DEFAULT NULL,
  `review` text,
  `review_date` datetime(6) DEFAULT NULL,
  `review_title` varchar(255) DEFAULT NULL,
  `upvotes` int NOT NULL,
  `user_id` varchar(15) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `FKdeg7cwy49odyve2l4firfl99g` (`user_id`),
  CONSTRAINT `FKdeg7cwy49odyve2l4firfl99g` FOREIGN KEY (`user_id`) REFERENCES `user` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1942 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;
```

4. 설계 및 구현

```
// 비밀번호 재설정의 가장 처음 과정을 진행하는 API
@PostMapping("/user-authentication")
public ResponseEntity<String> userAuthentication(@RequestBody Map<String, String> request) {
    // 얘네 역할은 유저의 ID와 이름으로 인증받아서 이메일 보내는 것까지
    String id = request.get("id");
    String name = request.get("name");
    User user = userService.getUserById(id);
    if (user != null && user.getName().equals(name)) {
        String email = user.getEmail();
        String verificationCode = verificationCodeService.generateCode(id); // 인증번호 생성
        boolean isSent = emailService.sendVerificationEmail(email, verificationCode); // 이메일 전송
        if (isSent) {
            return ResponseEntity.ok("Verification code has been sent to your email.");
        } else {
            return ResponseEntity.status(500).body("Failed to send verification code.");
        }
    } else {
        return ResponseEntity.status(400).body("User not found or name does not match.");
    }
}
```

- (자료11) UserController 부분이다. RequestBody를 통해 유저에게 id, name을 받아와서 해당 id를 통해 DB에 저장되어 있는 유저를 찾고, 해당 객체를 생성한다.

비밀번호 재설정시, 아이디에 해당되는 name으로 해당 유저가 맞는기 검증 IF의 조건이 위의 과정이고, 이후 인증번호를 보내기 위해서 유저객체에서 이메일을 받도록 한다.

이메일 서비스의 메서드를 통해 이메일 전송여부를 isSent를 통해 받고, isSent = TRUE 라면 전송되었다는 메시지를 주게 된다.

FALSE일 경우에는 상태코드 500을 반환하고, 이메일이 아닌 검증IF 과정에서 유저 객체의 이름이 다르거나, 유저를 찾을 수 없는 상태라면 상태코드 400을 반환하게 된다.

4. 설계 및 구현

```
// 비밀번호 재설정 엔드포인트 (인증번호 검증은 인증번호컨트롤러에서 처리)
@PostMapping("/reset-password")
public ResponseEntity<String> resetPassword(@RequestBody Map<String, String> request) {
    // 얘 역할은 그냥 비밀번호 초기화 역할만
    String id = request.get(key:"id");
    String newPassword = request.get(key:"newPassword");
    User user = userService.getUserById(id);
    // 비밀번호 재설정
    boolean isReset = userService.resetPassword(user.getEmail(), newPassword);

    if (isReset) {
        return ResponseEntity.ok(body:"Password has been reset successfully.");
    } else {
        return ResponseEntity.status(status:400).body(body:"User not found.");
    }
}
```

- (자료12) ID와 새로운 비밀번호를 받고, 받은 ID로 유저객체를 만든 후 비밀번호를 재설정하고 해쉬화 해주는 유저서비스의 리셋 패스워드 메서드를 활용한다.

```
// 이메일 업데이트
@PutMapping("/email")
public ResponseEntity<?> updateEmail(@RequestBody Map<String, String> request) {
    try {
        String userId = request.get(key:"userId");
        String newEmail = request.get(key:"value");

        userService.updateEmail(userId, newEmail);

        return ResponseEntity.ok(Map.of(
            k1:"success", v1:true,
            k2:"message", v2:"Email updated successfully."
        ));
    } catch (IllegalArgumentException e) {
        return ResponseEntity.badRequest().body(Map.of(
            k1:"success", v1:false,
            k2:"message", e.getMessage()
        ));
    }
}
```

- (자료13) 메서드 호출에서 ResponseEntity에서 OK or BadRequest를 반환한다.

4. 설계 및 구현

```
//회원가입
public void registerUser(String username, String rawPassword) {
    String encodedPassword = passwordService.encodePassword(rawPassword);
    User user = new User();
    user.setId(username);
    user.setPw(encodedPassword); // 해시화된 비밀번호 저장
    userRepository.save(user);
}
```

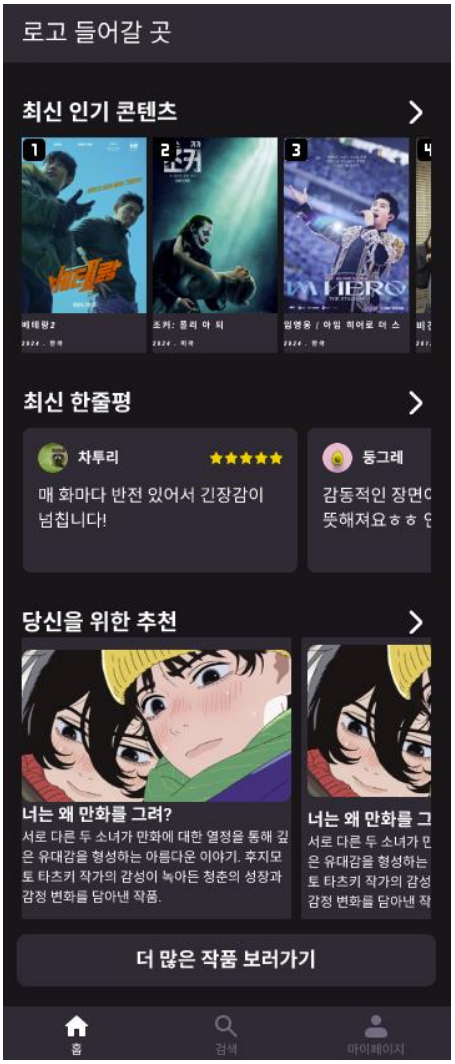
- (자료14) 회원가입 및 비밀번호 재설정 시, 사용자에게 입력받은 비밀번호를 바로 해쉬화 하여 DB에 저장하도록 한다.

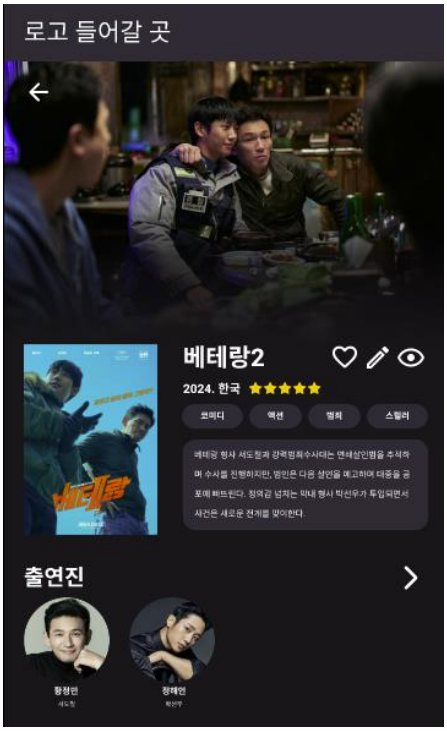
```
// 비밀번호 검증
public boolean matches(String rawPassword, String encodedPassword) {
    return passwordEncoder.matches(rawPassword, encodedPassword);
}
```

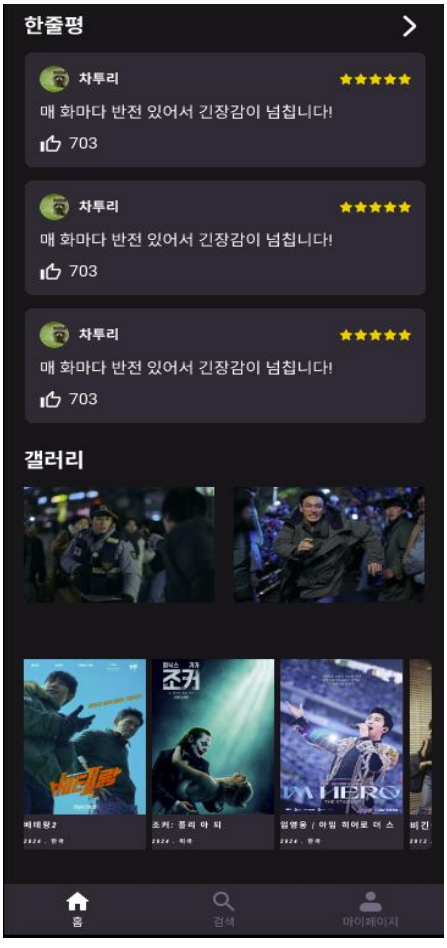
- (자료15) 로그인시, 회원이 입력하는 비밀번호를 바로 해쉬화하여 DB의 비밀번호와 동일한지 검사하도록 한다.

4. 설계 및 구현

[웹/프론트엔드]

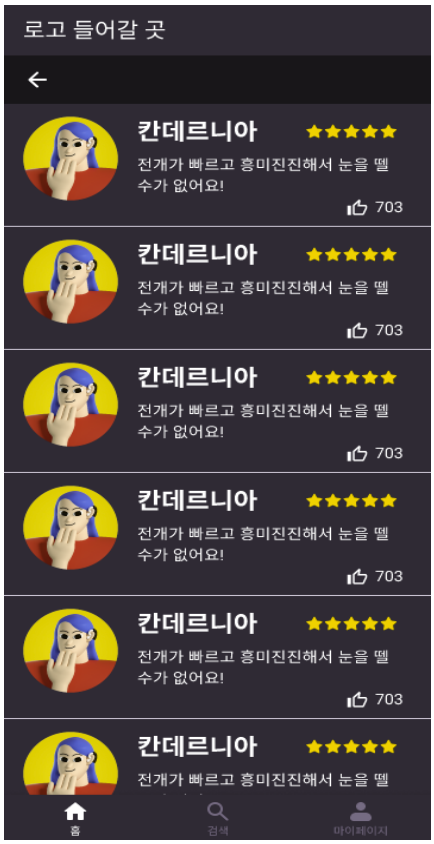




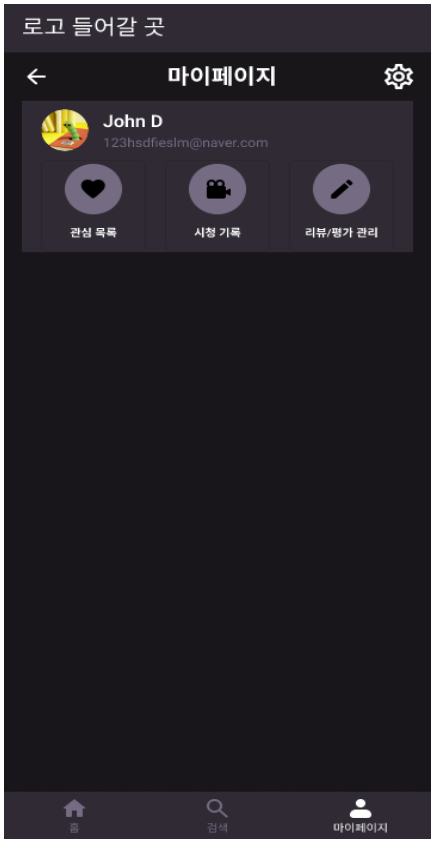


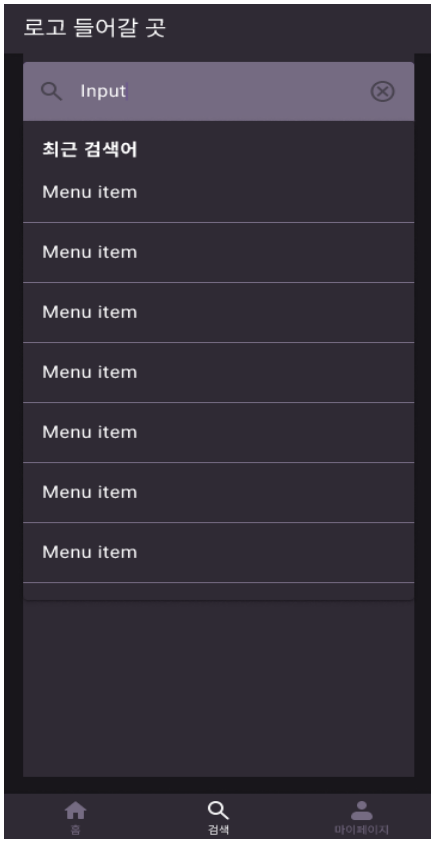
- (자료16) 웹/프론트엔드 초안이다.
피그마를 통해 작성 및 연동하였다.

4. 설계 및 구현

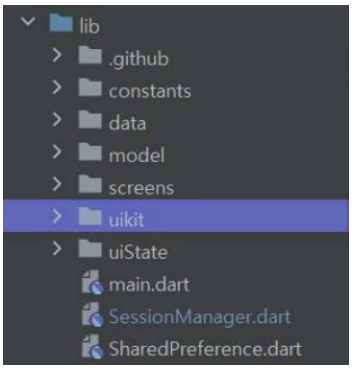

로고 들어갈 곳
칸데르니아
전개가 빠르고 흥미진진해서 눈을 뗄 수가 없어요!
703
칸데르니아
전개가 빠르고 흥미진진해서 눈을 뗄 수가 없어요!
703
칸데르니아
전개가 빠르고 흥미진진해서 눈을 뗄 수가 없어요!
703
칸데르니아
전개가 빠르고 흥미진진해서 눈을 뗄 수가 없어요!
703
칸데르니아
전개가 빠르고 흥미진진해서 눈을 뗄 수가 없어요!
703
칸데르니아
전개가 빠르고 흥미진진해서 눈을 뗄
홈
검색
마이페이지


로고 들어갈 곳
액션
코미디
공포
스릴러
최신순
홈
검색
마이페이지


로고 들어갈 곳
마이페이지
John D
123hsdfieslm@naver.com
관심 목록
시청 기록
리뷰/평가 관리
홈
검색
마이페이지


로고 들어갈 곳
Input
최근 검색어
Menu item
Menu item
Menu item
Menu item
Menu item
Menu item
Menu item
홈
검색
마이페이지

4. 설계 및 구현



- (자료17) 다음은 파일구조에 대한 부분이다.

- .github : 깃허브 페이지 배포파일
- Constants : 색상, 장르 등 변하지 않는 값 정의
- Data/req : 로그인 요청
- Data/res : 로그인 응답
- Data의 나머지는 영화, 리뷰, 사용자 데이터 API
- Model : 각 데이터에 대한 모델 클래스
- Screens/edit_profile_screen : 마이페이지 프로필 설정 편집 화면
- Screens/ edit_review_screen : 리뷰 편집 화면. 기존에 리뷰 수정 또는 삭제 기능
- Screens/ favorite_movie_screen : 관심있는 영화 목록 화면
- Screens/ home_screen : 웹 접속시 제일 먼저 뜨는 홈화면
- Screens/ inquiry_screen: 문의사항
- Screens/ login_screen : 로그인 페이지
- Screens/ main_screen : 화면 관리
- Screens/ movie_detail_screen : 영화 상세화면
- Screens/ movie_list_screen: 영화 목록 화면
- Screens/ notices_screen : 공지사항
- Screens/ profile_screen : 마이페이지
- Screens/ review_list_screen : 리뷰 목록, 최신에 쓴 리뷰
- Screens/ sign_up_screen : 회원가입 화면
- Screens/ user_review_screen : 특정 사용자가 작성한 리뷰 목록
- Screens/ watch_movie_screen : 시청했던 영화 목록 화면
- Uikit/widgets : 배우, 버튼, 영화, 리뷰 카드 등 자주 쓰이는 위젯들을 정의
- Uikit/ uiState: 사용자 프로필 정의
- SessionManager과 SessionPreference는 섹션 관리용도. 로그인 섹션 관리

4. 설계 및 구현

password 1

```dart
Future<void> sendVerificationCode(BuildContext context) async {
  if (idController.text.isEmpty || nameController.text.isEmpty) {
    final snackBar = SnackBar(
      content: Text('ID와 이름을 입력해 주세요.'),
      backgroundColor: Colors.red,
    );  // SnackBar
    ScaffoldMessenger.of(context).showSnackBar(snackBar);
    return;
  }

  try {
    final response = await http.post(
      Uri.parse('https://contentspick.site/api/users/user-authentication'),
      headers: {'Content-Type': 'application/json; charset=UTF-8'},
      body: json.encode({
        'id': idController.text,
        'name': nameController.text,
      }),
    );
```

- (자료18) 입력 받은 사용자의 아이디와 이름을 전달하도록 한다.

```dart
if (response.statusCode == 200) {
  Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) => PasswordResetStep2(
        id: idController.text,
      ),  // PasswordResetStep2
    ),  // MaterialPageRoute
  );
} else {
  final snackBar = SnackBar(
    content: Text('일치하는 아이디가 없습니다.'),
    backgroundColor: Colors.red,
  );  // SnackBar
  ScaffoldMessenger.of(context).showSnackBar(snackBar);
}
} catch (e) {
final snackBar = SnackBar(
  content: Text('오류가 발생했습니다. 다시 시도해주세요.'),
  backgroundColor: Colors.red,
);  // SnackBar
```

- (자료19) 정상적인 응답이라면 (200), password2 페이지로 넘어가도록 한다.
- 이때, 아이디와 함께 전달하도록 한다.

4. 설계 및 구현

password2

```
Future<void> verifyCode(BuildContext context) async {
  if (codeController.text.isEmpty) {
    final snackBar = SnackBar(
      content: Text('인증번호를 입력해 주세요.'),
      backgroundColor: Colors.red,
    );  // SnackBar
    ScaffoldMessenger.of(context).showSnackBar(snackBar);
    return;
  }

  try {
    final response = await http.post(
      Uri.parse('https://contentspick.site/verification/verify'),
      headers: {'Content-Type': 'application/json; charset=UTF-8'},
      body: json.encode({
        'userId': id,
        'code': codeController.text,
      }),
    );
```

- (자료20) 사용자로부터 인증번호를 입력 받으면, 아이디와 인증번호를 전달한다.
- 이때, 아이디는 password1 페이지에서 전달 받은 아이디 그대로 전달한다.
- 응답코드 200이면 password3 페이지로 넘어가도록 한다. (아이디 전달)
- password3 에서는 아이디와 새로운 비밀번호를 입력 받고, 비밀번호 재입력 필드가 일치하지 않는다면 다시 입력하도록 한다. 이후 아이디와 새로운 비밀번호를 서버에 전송하고, 성공적으로 바뀐다면 응답코드 200을 리턴한다.

```
Future<void> resetPassword(BuildContext context) async {
  if (passwordController.text.isEmpty || confirmPasswordController.text.isEmpty) {
    final snackBar = SnackBar(
      content: Text('모든 필드를 입력해 주세요.'),
      backgroundColor: Colors.red,
    );  // SnackBar
    ScaffoldMessenger.of(context).showSnackBar(snackBar);
    return;
  }

  if (passwordController.text != confirmPasswordController.text) {
    final snackBar = SnackBar(
      content: Text('비밀번호가 일치하지 않습니다. 다시 입력해주세요.'),
      backgroundColor: Colors.red,
    );  // SnackBar
    ScaffoldMessenger.of(context).showSnackBar(snackBar);
    return;
  }
```

4. 설계 및 구현

- (자료21)

SessionManager

isLogin() : 사용자가 로그인 했는지 확인하는 정적 메서드로, 공유된 설정에 사용자 ID가 저장되어 있는지를 확인한다.

login(REQ_L001 reqL001) : 비동기적으로 사용자를 로그인시키고, UserData와 상호작용하여 사용자 ID를 공유된 설정에 저장하도록 한다.

Logout() : 비동기적으로 사용자를 로그아웃 시키고, 공유된 설정에서 사용자 ID를 제거한다.

deleteAcoount() : 저장된 사용자 ID를 기반으로 사용자의 계정을 삭제하고 로그아웃 시킨다.

getCurrentUser() : 저장된 사용자 ID를 이용해 서버에서 현재 사용자 프로필 업데이트, Json 응답을 파싱하여 User 객체를 생성하도록 한다.

SharePrefManager

init() : SharedPreferences 인스턴스를 초기화 한다.

saveUserId(String userId) : 사용자 ID를 저장한다.

getUserId() 및 getUserNickname() : 각각 사용자 ID와 닉네임을 업데이트 한다.

removeUserId() : 저장된 사용자 ID를 제거한다.

isLoggedIn() : SessionManager의 메서드와 유사하게 로그인 상태를 확인한다.

isInitialized() : SharedPreferences가 초기화되었는지 확인한다.

4. 설계 및 구현

main

WidgetsFlutterBinding.ensureInitialized() / await SharePrefManager.init()

: 플러터, SharePreferences 초기화 하도록 한다.

checkLoginStatus() : SharedPreferences를 사용하여 저장된 userId 기준으로 사용자의 로그인 상태를 확인 하도록 한다.

main_screen

StatefulWidget : 사용자가 선택한 탭에 따라 화면 상태 변경

int _selectedIndex : 현재 선택된 탭 인덱스 저장

List<Widget> _screens : 각각의 인덱스에 해당하는 화면. _onItemTapped; 탭 선택 시 호출한다. 선택한 탭이 2(마이페이지)이면 로그인 상태를 확인하여, 로그인 되어있지 않으면 로그인 화면으로, 되어 있으면 프로필화면으로 이동한다. 이때, 메인 스크린은 하단에 내비바를 통해 화면 이동 관리를 하는 역할이다.

4. 설계 및 구현

main

WidgetsFlutterBinding.ensureInitialized() / await SharePrefManager.init()

: 플러터, SharePreferences 초기화 하도록 한다.

checkLoginStatus() : SharedPreferences를 사용하여 저장된 userId 기준으로 사용자의 로그인 상태를 확인 하도록 한다.

main_screen

StatefulWidget : 사용자가 선택한 탭에 따라 화면 상태 변경

int _selectedIndex : 현재 선택된 탭 인덱스 저장

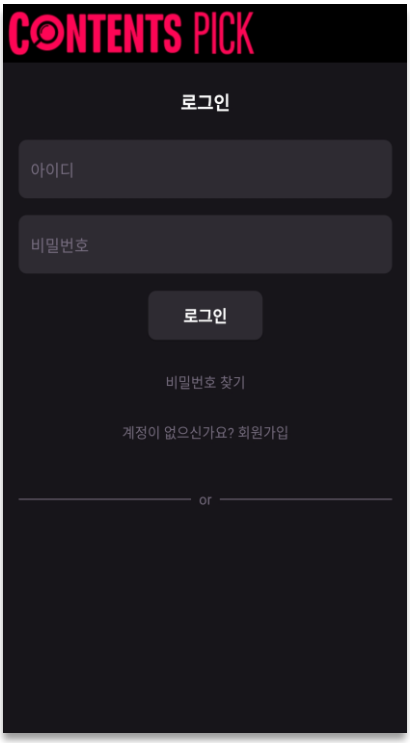
List<Widget> _screens : 각각의 인덱스에 해당하는 화면. _onItemTapped; 탭 선택 시 호출한다. 선택한 탭이 2(마이페이지)이면 로그인 상태를 확인하여, 로그인 되어있지 않으면 로그인 화면으로, 되어 있으면 프로필화면으로 이동한다. 이때, 메인 스크린은 하단에 내비바를 통해 화면 이동 관리를 하는 역할이다.

4. 설계 및 구현

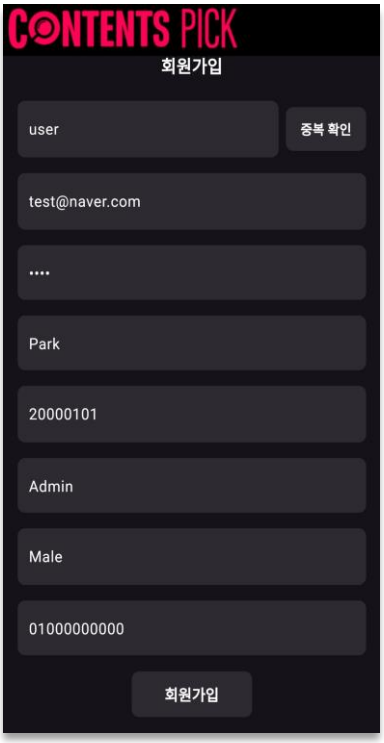
[웹 사이트 로고]

- (자료22) 웹 사이트의 배경을 검은색으로 정하고, 배경에서 눈에 띄기 위해 시안성이 좋은 색을 정한 후, 쉽게 콘텐츠를 고르라는 취지/의미에 맞게 '콘텐츠 픽' 이라고 정함.

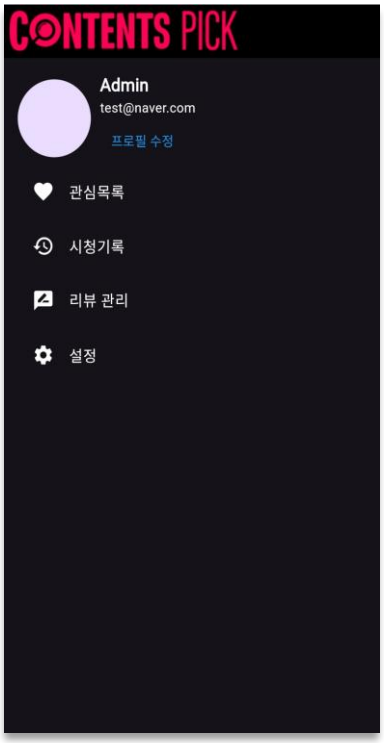
5. 개발 결과



[로그인 화면]



[회원가입 화면]



[마이 페이지]

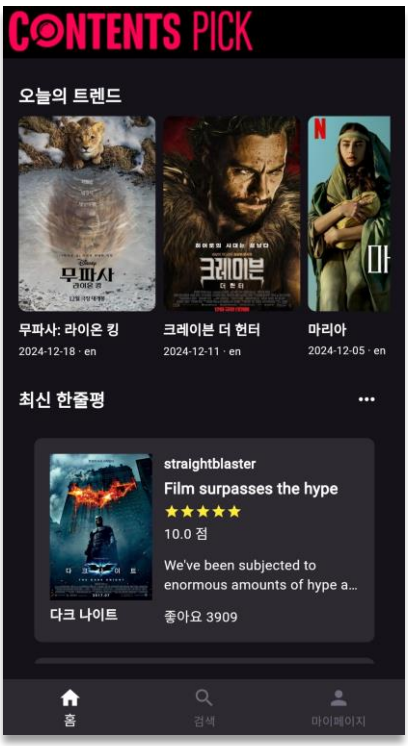
(자료23)

- 로그인 화면에서는 아이디와 비밀번호를 기입할 수 있고, 하단부에 비밀번호 찾기와 회원가입을 할 수 있도록 함.

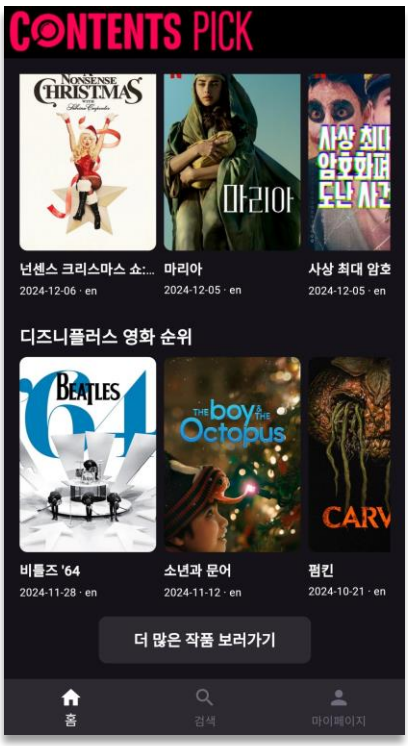
- 회원가입에서는 아이디의 중복이 일어나면 안되기에, 중복확인을 할 수 있도록 하고 그 외 user객체에 필요한 정보들을 받도록 함. (아이디, 이메일, 비밀번호, 이름, 생년월일, 닉네임, 성별, 전화번호)

- My페이지에서는 관심목록, 시청기록, 작성한 리뷰 등을 관리할 수 있다.

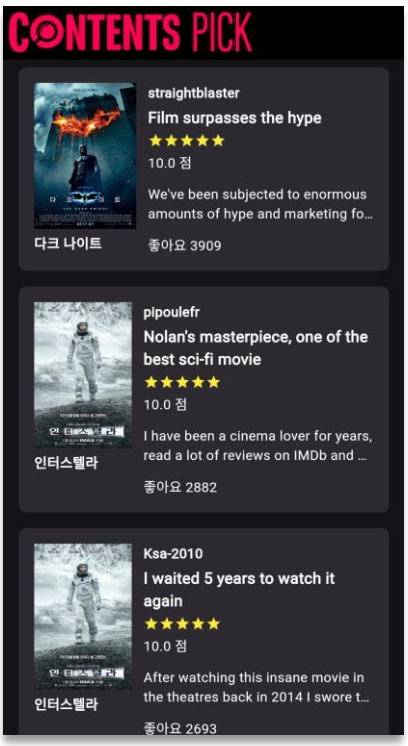
5. 개발 결과



[메인 페이지 1]



[메인 페이지 2]



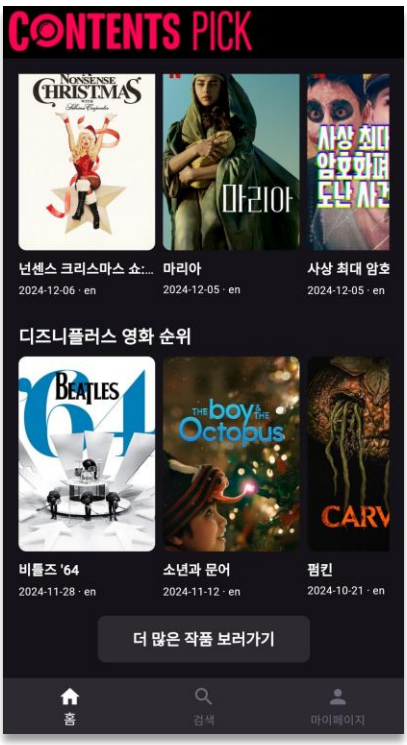
[리뷰 페이지]

(자료24)

- 메인 화면에서는 '오늘의 트렌드' / '넷플릭스 영화 순위' / '디즈니플러스 영화 순위' / '리뷰' 에 관한 영화를 한 눈에 볼 수 있고, '오늘의 트렌드' 의 경우 프로젝트에 사용된 TMDB API 내의 인기도 지표와 조회수를 반영하여 순위를 매기고 매일 갱신, 홈페이지에 반영된다.

- 리뷰 페이지에서는 크롤링 한 데이터를 토대로 웹 사이트에 최신순으로 반영한다.

5. 개발 결과



[메인페이지 3]

[필터링 1]

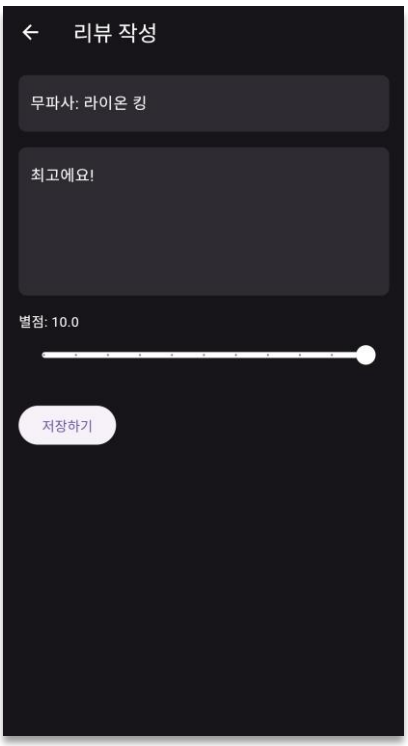
[필터링 2]

(자료25)

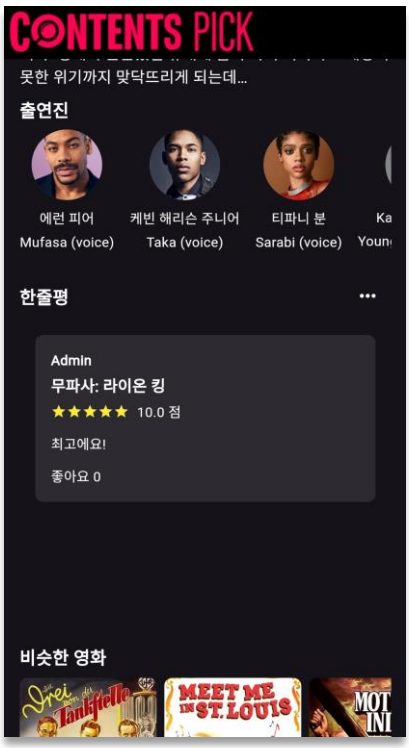
- 메인페이지 하단부에서는, '더 많은 작품 보러가기' 를 누를 수 있고 해당 페이지에 들어가면 필터링 기능이 생기도록 한다.

- 장르별로 필터링 할 수 있도록 하고 (필터링1 참조), 이를 다시 최신순, 인기순, 별점 높은 순 (필터링2 참조) 으로 정렬 할 수 있도록 한다.

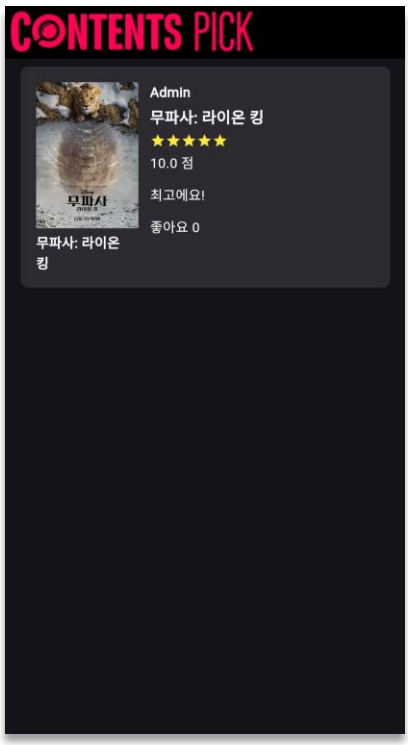
5. 개발 결과



[리뷰 작성]



[리뷰 확인]

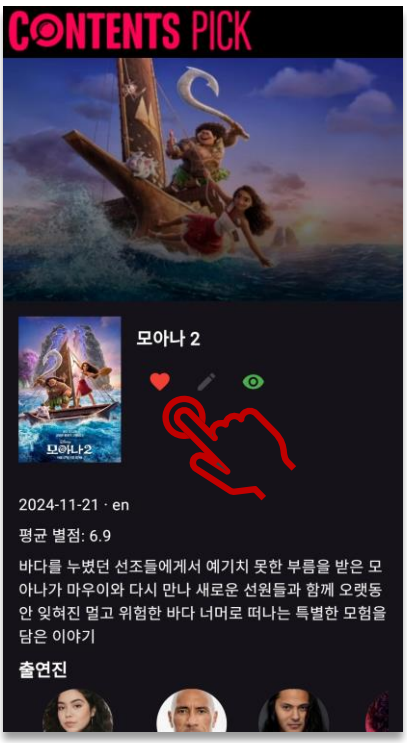


[리뷰 관리]

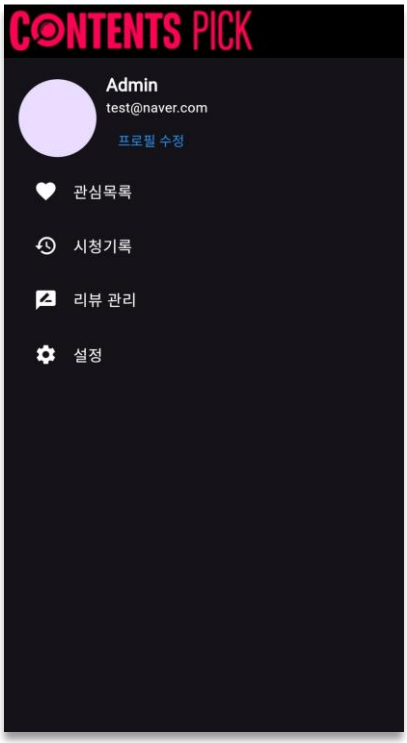
(자료26)

- 영화에 대한 리뷰는 별점과 함께 작성할 수 있다. (리뷰작성 참조)
- 리뷰 작성 후 영화 내 정보에 바로 반영된다.
- 이후 My페이지에서 수정 및 삭제 등의 관리를 할 수 있다.

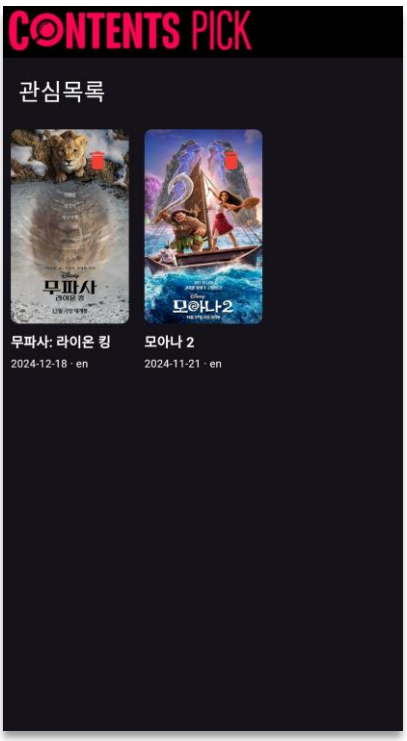
5. 개발 결과



[영화 정보]



[관심목록/시청기록]



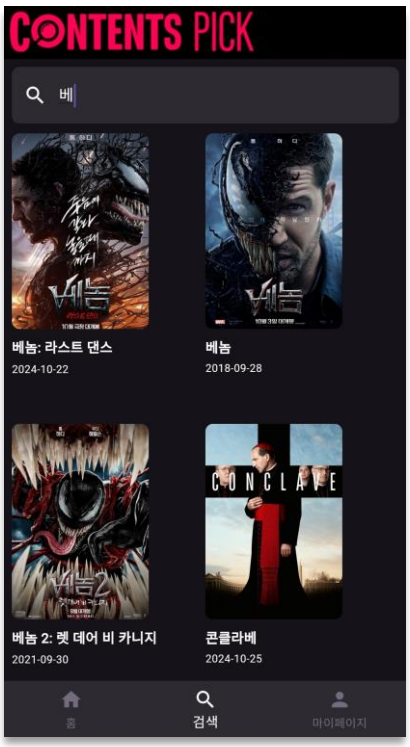
[관심목록]

[팝업알림]

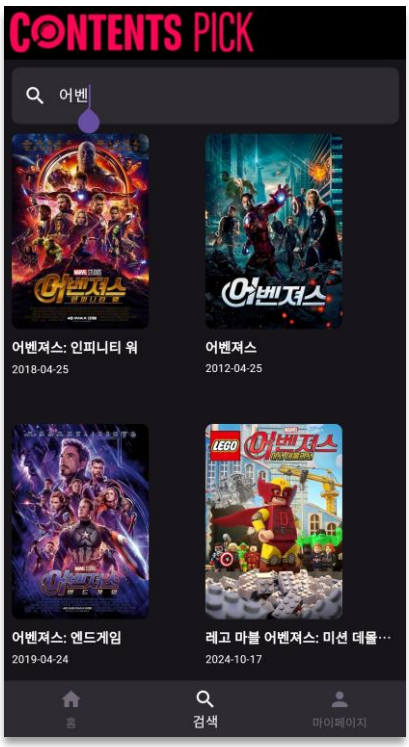
(자료27)

- 유저가 보고 싶은 영화라면 하트를 눌러 관심목록에 담을 수 있고, 이미 본 영화라면 눈을 눌러 시청기록에 담을 수 있다. 이때, 추가 및 삭제 할 경우 웹 사이트 하단부에 팝업 알림도 구현하였다.
- My페이지에서 마찬가지로 유저의 관심목록, 시청기록을 확인할 수 있으며 삭제도 가능하다.

5. 개발 결과



[검색 페이지 1]

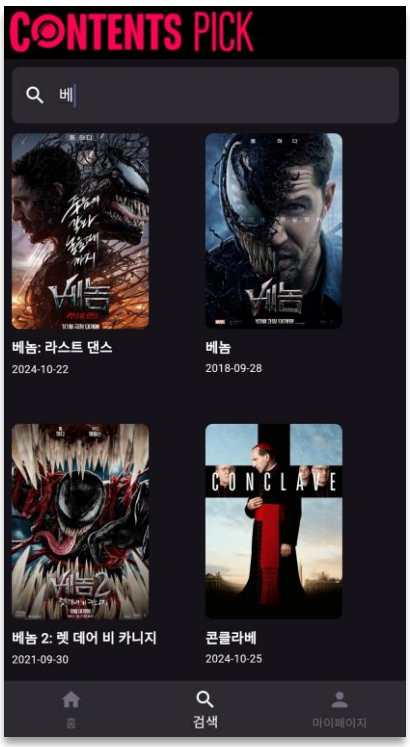


[검색 페이지 2]

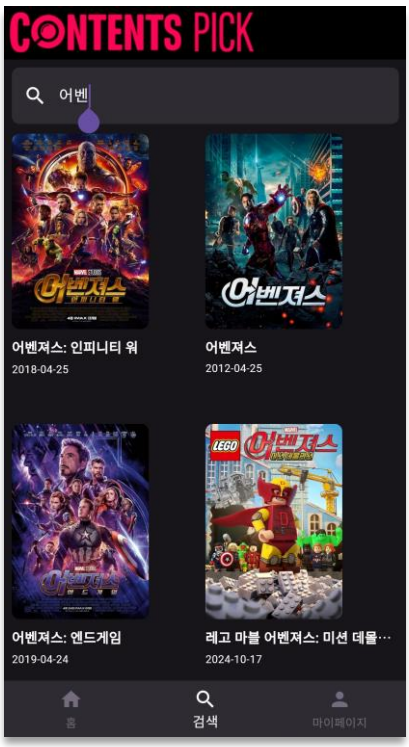
(자료28)

- 검색 기능은 제목의 순서 (차순) 상관 없이 한 글자라도 일치한다면, 결과값에 맞는 영화를 나타내어 사용자의 검색 편의를 도와준다.

5. 개발 결과



[검색 페이지 1]



[검색 페이지 2]

(자료28)

- 검색 기능은 제목의 순서 (차순) 상관 없이 한 글자라도 일치한다면, 결과값에 맞는 영화를 나타내어 사용자의 검색 편의를 도와준다.

6. 기대효과

인터넷 소비자 10명 중 5명은 온라인 광고에 '부정적'

박주하 | 입력 2018.02.23 16:30 | 댓글 0

[뉴스메카 박주하 기자] 인터넷 소비자 10명 중 5명은 온라인 광고에 '부정적'인 인식을 가지고 있는 것으로 조사됐다.

지난 21일 한국인터넷진흥원(KISA)이 발표한 '2017 온라인광고 산업 동향 조사' 결과에 따르면, 2017년 온라인광고 시장규모는 약 4조 4,285억 원으로 전년 대비 2,716억 원 성장했다. 특히, 모바일 광고가 전년 대비 13.9% 성장한 2조 2,585억 원의 매출을 기록하며 성장의 원동력이 됐다.

온라인 광고 시장 규모, 단위: %(사진= 2017 온라인광고 산업 동향 조사)

상승세와 달리 온라인 광고에 대한 인터넷 사용자들의 인식은 좋지 않다. 인터넷 이용자를 대상으로 시행한 온라인 광고에 대한 인식 조사 결과, 응답자 중 절반 이상(51.7%)이 온라인 광고를 부정적으로 생각한다고 답변했다.

[KISA 온라인광고 산업 동향 조사 결과에 대한 뉴스메카 기사]

온라인 광고가 부정적인 이유, 단위: %(사진= 2017 온라인광고 산업 동향 조사)

6. 기대효과

- 위 자료는 과거 인터넷 소비자 온라인 광고에 대해 조사한 결과이다.
인터넷 소비자 10명 중 절반인 5명은 온라인 광고에 부정적인 인식을 갖고 있으며,
부정적인 이유로 가장 큰 사유는 '콘텐츠 이용을 방해해서' 라고 답변한 결과가 있다.

그래서 우리는 본 프로젝트를 진행하면서 이후 수익화에 대한 고민을 할 때, 대다수가
수익모델로 정하는 광고에 대한 부분에 의견을 모았고, 결과로는 초기에 수익성을 조금
놓치더라도 광고를 최소한으로만 운영하여 커뮤니티 활성화를 더욱 중점에 두고
운영하여 트래픽을 점차 늘리는 방식을 택하기로 하였다.

이후 많아진 표본 (사용자) 의 데이터를 필요한 곳에 판매 및 제공할 수 있을 것으로
기대하고, 수익성을 챙길 수 있을 것으로 예상된다.

왓챠, CJ CGV와 영화 데이터 통합 분석 플랫폼 사업 확대 '맞손'

정유림 기자 입력 2020.11.10 09:30

댓글 0

최병환 CJ CGV 대표(왼쪽)와 박태훈 왓챠 대표가 영화 콘텐츠 기반 데이터 통합 분석 플랫폼 사업 영역 확대를 위한 업무협약(MOU)을 체결하고 기념 촬영을 하고 있다. [사진:왓챠]

[디지털투데이 정유림 기자] 온라인 동영상 스트리밍(OTT) 서비스 왓챠는 CJ CGV와 영화 콘텐츠 기반 데이터 통합 분석 플랫폼 사업 영역 확대를 위한 업무협약(MOU)을 체결했다고 10일 밝혔다.

[왓챠-CJ CGV 데이터 분석 사업 확대 사례]

6. 기대효과

- 사용자에게 주는 보상은 고객사에게 데이터를 판매하며 판매금의 일부는 시사회/상영회 등 영화 매니아들에게 희소성이 있는 것으로 받은 뒤, 특정 간격으로 (매주, 매월) 추천수가 많은 리뷰를 남겨주거나 혹은 활성화를 위한 다양한 활동을 해주는 사용자에게 보상을 줄 수 있도록 한다.

물론 사용자에 대한 패널티도 존재해야 한다. 활성화에 대한 보상을 받기 위해 무작위의 내용, 혹은 반복적으로 내용을 업로드 하는 사람에게는 블라인드에 대한 패널티, 그리고 추천/반대수에 대한 기능으로 반대수가 일정 수 이상 많다면 패널티를 주는 방안이 있다.

사용자는 'CONTENTS PICK' 플랫폼에 데이터에 대한 공급을 주고, 고객사는 'CONTENTS PCIK' 데이터에 대한 수요가 있으니 본 플랫폼은 수요와 공급을 조율하는 중개플랫폼으로 방향성을 제시할 수 있다.

7. 참고문헌

- 2023 문화체육관광부 국민여가 활동조사 통계

- TMDB API (developer.themoviedb.org)

- IMDB (imdb.com)

- KISA 온라인광고 산업 동향 조사 결과