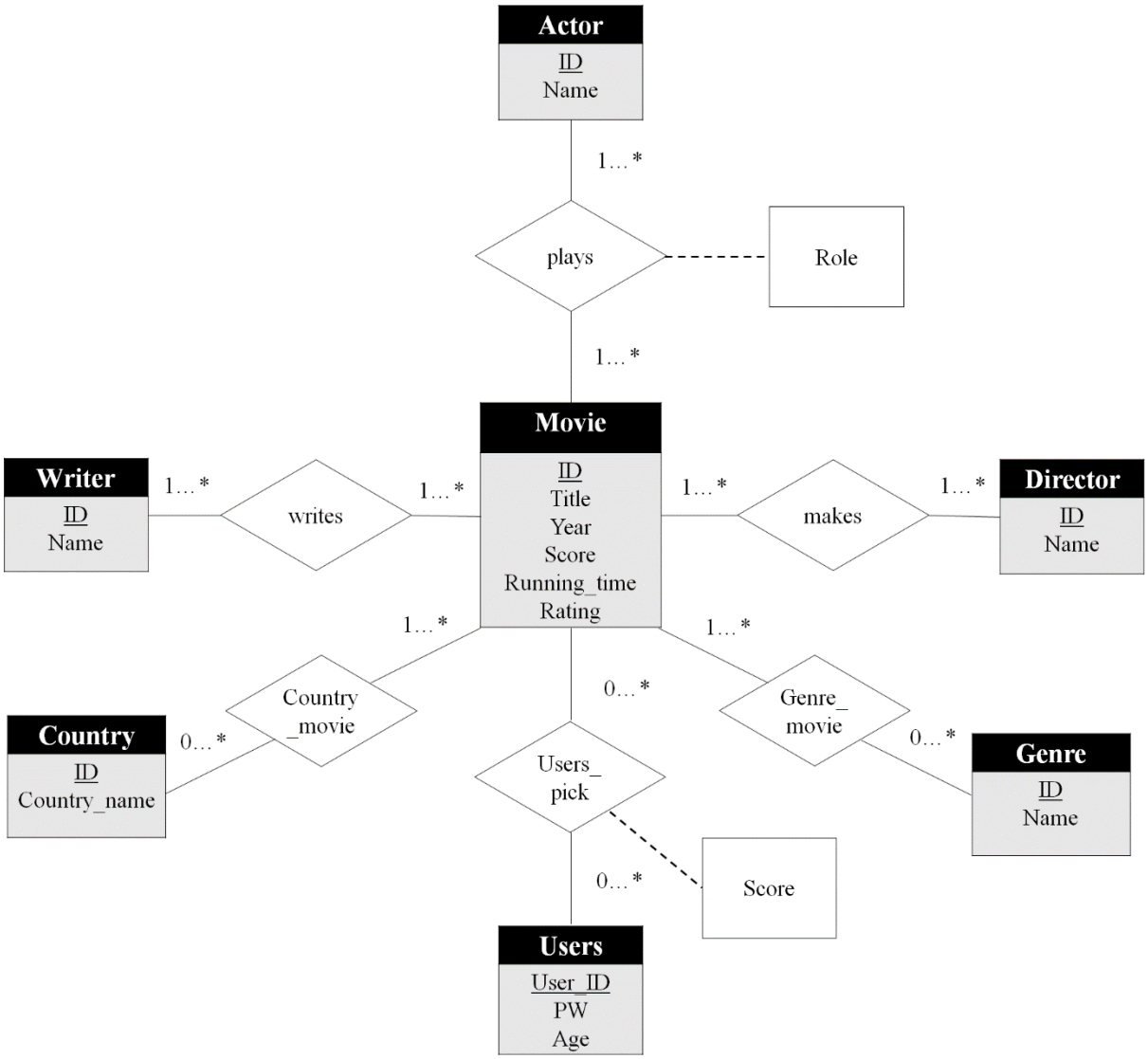
데이터베이스 Term Project

: 영화 데이터베이스 어플리케이션

2016130927 박준영

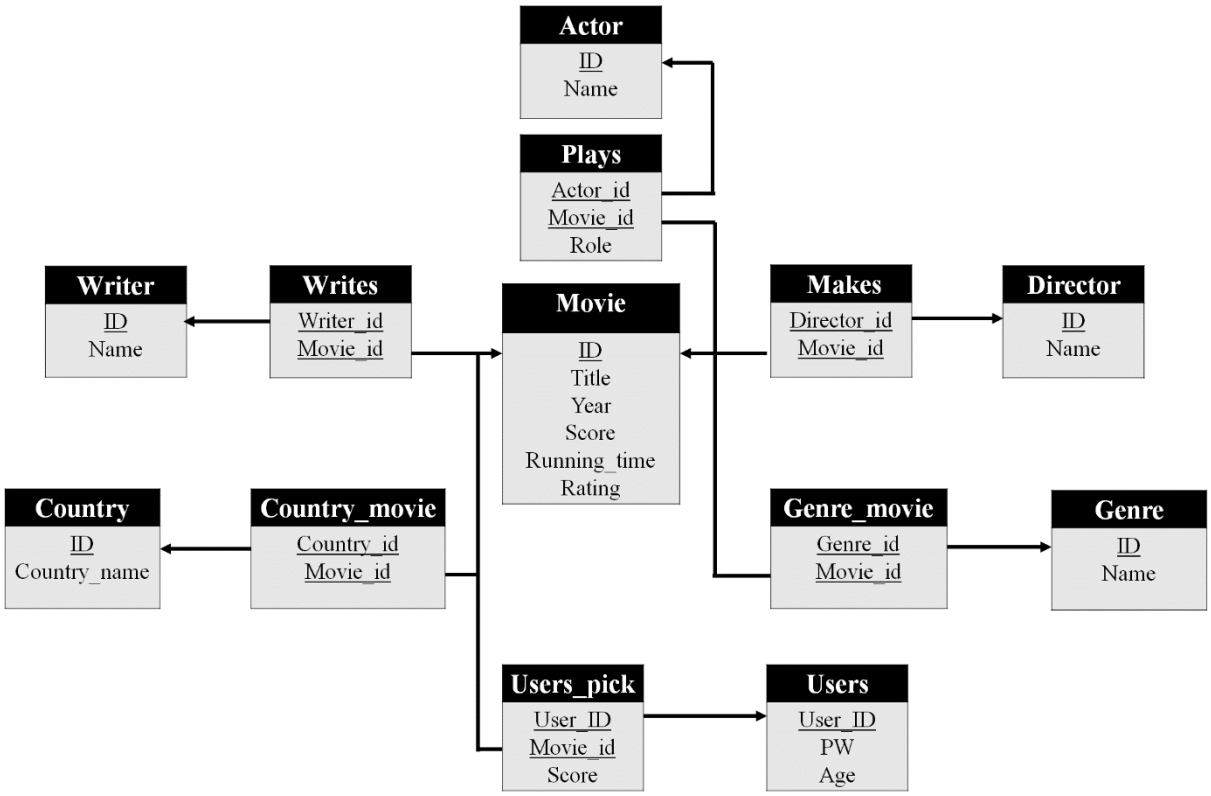
I. ERD / Schema diagram



<영화 데이터베이스 ERD>

영화를 중심으로 감독, 작가, 배우, 장르, 국가 등 정보를 포함하고 있는 엔티티를 연결한다. 제목과 제작연도, 평점, 상영시간, 관람 연령 등 특정 영화에만 속하는 요소는 영화 엔티티에 남겨두었다. 반면 영화가 중복 정보를 가질 수 있는 요소는 따로 분리했다. 가령 '매트릭스'의 감독은 워쇼스키 형제로 두 명이고, 크리스토퍼 놀란 감독은 '메멘토', '인셉션' 등 복수의 영화를 제작했다. SQL 쿼리문을 작성할 때 먼저 영화 정보를 입력한 뒤 이에 해당하는 감독, 작가, 배우 정보를 입력하는 형식이었기에 본 DB ERD에서 각 카디널리티 제약조건은 1…*이다. 한편, 장르와 국가 정보는 DB에 삽입할 영화를 고르기 이전 입력했고, 이 테이블의 값이 영화에 나타나지 않을 수

있으므로 0…*이다. 물론 영화와 이들 엔티티와의 관계는 적어도 한 개 이상의 국가에서 제작되고, 한 개 이상의 장르를 가지므로 1…*이다. 영화 정보와 별개로 본 어플리케이션을 사용하는 회원 엔티티를 구성해 회원이 임의의 영화에 평점을 매기는 관계를 부여했다. 때문에 위 속성과는 달리 회원이 영화를 고를 필요나, 영화가 회원에게 속할 필요가 없으므로 모두 0…*이다.



<영화 데이터베이스 스키마 다이어그램>

위 ERD를 기반, 엔티티 및 릴레이션을 SQL 스키마로 표현한 다이어그램이다. 작성의 용이성을 위해 만일 참조하는 어트리뷰트가 동일하고, 같은 방향으로 화살표를 이을 경우 이를 합쳐서 표기했다. 가령 plays, makes, genre_movie 테이블 그리고 writes, country_movie, users_pick의 movie_id 어트리뷰트가 모두 movie 테이블의 id 어트리뷰트를 참조하는데, 방향이 동일하다면 중간에 선을 이어주었다.

II. 영화 데이터베이스 어플리케이션

i. 개발 환경: 플라스크, HTML: 파이썬 플라스크 프레임워크를 통해 PostgreSQL에 접속해 DB를 사용, html을 렌더링한다.

ii. 어플 기능 및 구현 설명: Term Project 디렉토리 내 app.py 및 templates 디렉토리, Movie.sql 파일이 포함되어 있다. 이때 Movie.sql 파일을 Term Project 폴더 내 포함해 PostgreSQL에 Movie.sql의 DB 정보가 사전 설치되지 않을 때를 방지했다. 이 경우 실행 시마다 이전 실행에 작성한 정보가 다시 사라지는 구조이므로(SQL 파일 또한 DROPTABLE IF EXISTS를 사용하였다), 실제 사용자가 어플을 사용할 때에는 아래와 같이 주석 처리한다.

```
# file = open("Movie.sql", "r")
# sqlfile = file.read()
# cur.execute(sqlfile)
# connect.commit()
# file.close()
# if your PSQL has no data from movie.sql, then run this.
```

(1). 회원가입

아이디와 비밀번호, 나이를 입력해 회원 등록을 할 수 있다. 이때 1). 아이디와 비밀번호 중 특정 항목을 등록하지 않았거나, 2). 비밀번호를 비밀번호 확인 항목과 다르게 입력하였거나, 3). 가입하려는 아이디가 기존 DB에 있다면 새로 입력해야 한다. INSERT 쿼리문을 통해 users 테이블에 회원정보를 입력한다.

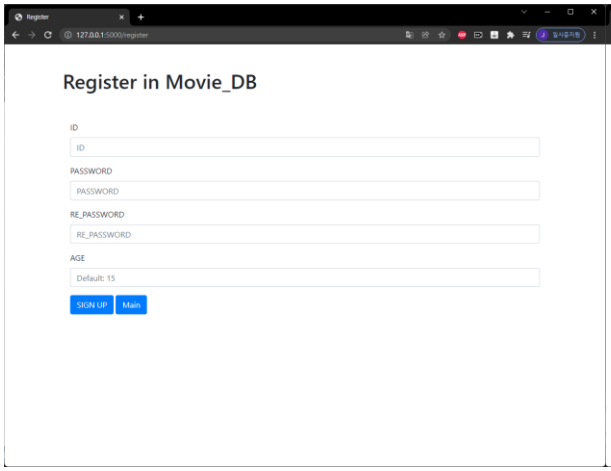
```
if not (userid and password and re_password):
    regimsg = 'empty'
    return render_template("register.html", regimsg=regimsg)
elif (password != re_password):
    regimsg = 'pw_check'
    return render_template("register.html", regimsg=regimsg)
else:
    cur.execute("INSERT INTO users (id, pw, age) VALUES ('{}', '{}', {});".format(userid,
password, age))
    connect.commit()
    regimsg = 'success'
    return render_template("register.html", regimsg=regimsg)
# if valid, insert new user info to user table
```

*SQL: INSERT문 사용

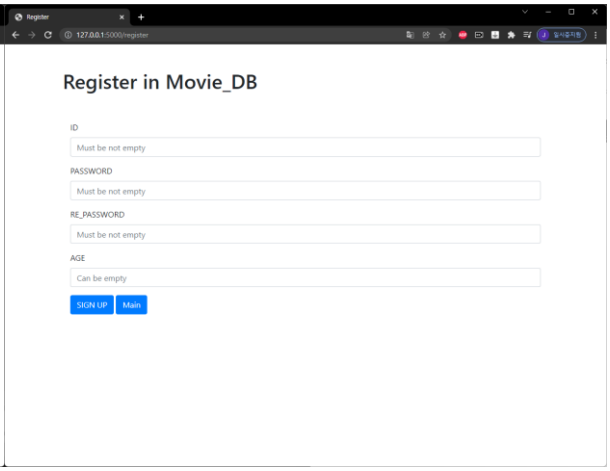
파이썬 내부 조건문을 통해 앞 두 가지 사항을 다룬 반면, 마지막 상황은 PSQL을 이용했다. 앞서 스키마 다이어그램에서 보았듯, users 테이블이 id 어트리뷰트를 프라이머리 키로 가지고 있기 때문에 프라이머리 키 제약조건을 위반하는 경우 500번 에러 코드를 반환하기 때문이다.

```
@app.errorhandler(500)
def internal_error(error):
    connect.rollback()
    return render_template('register.html', regimsg='fail'), 500
```

이 경우 INSERT 쿼리문이 실행되지 않도록 rollback해준 뒤 회원가입 페이지로 다시 돌아간다.

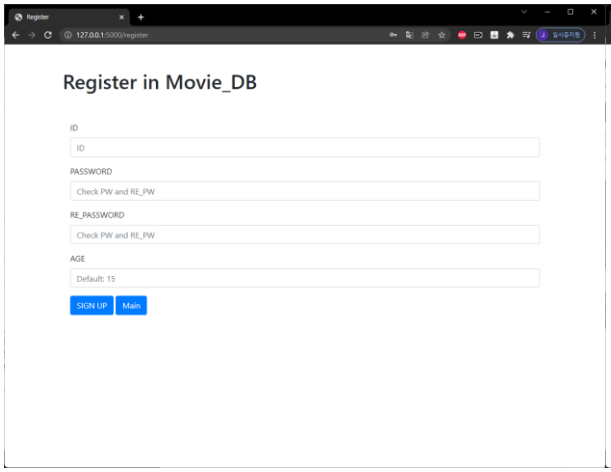


<회원가입 페이지>

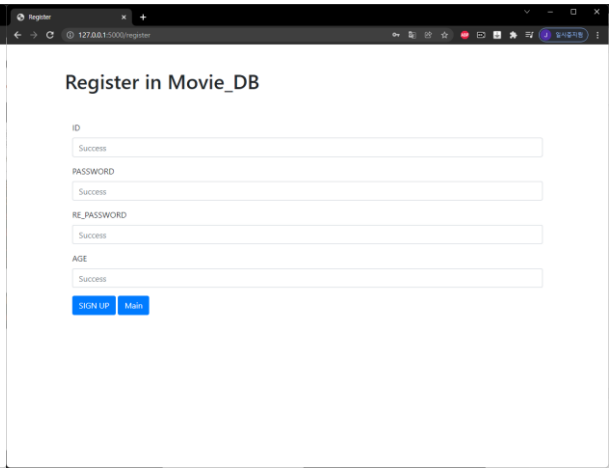


<필수 입력 항목 존재>

회원정보 어트리뷰트 중 나이는 입력하지 않아도 되며, 입력하지 않았을 때 디폴트 값 15가 설정된다. 유효한 숫자(수 형태, 사전 설정한 나이 범위 내)를 입력한 경우 그대로 입력된다.



<비밀번호 일치>

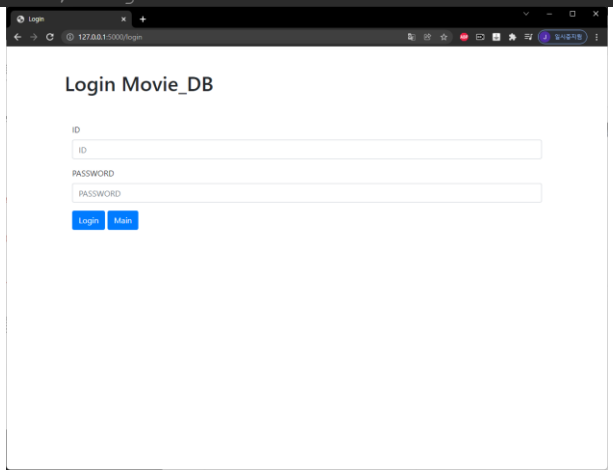


<회원가입 성공>

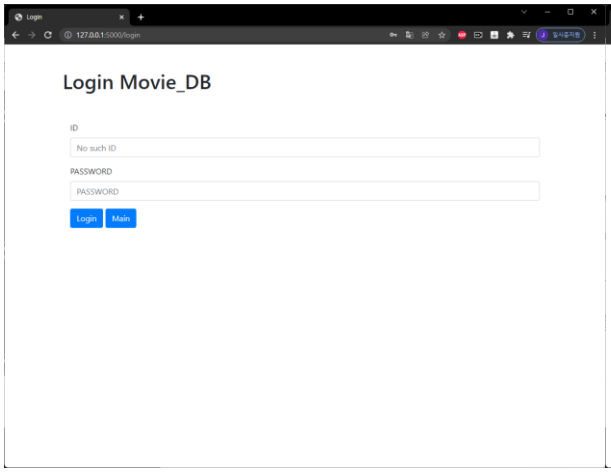
(2). 로그인/로그아웃

아이디와 비밀번호를 사용해 로그인한다. SELECT 쿼리문을 통해 사용자가 입력한 아이디 및 비밀번호가 일치하는 튜플이 users 테이블에 존재하는지 확인한다. 아이디가 유효한지, 비밀번호가 유효한지 또한 알려준다. 로그인에 성공한 경우 이후 사용자 정보를 어플에 반영하기 위해 플라스크에서 import한 session 함수를 사용, 아이디를 저장하고 메인 페이지로 자동으로 이동한다.

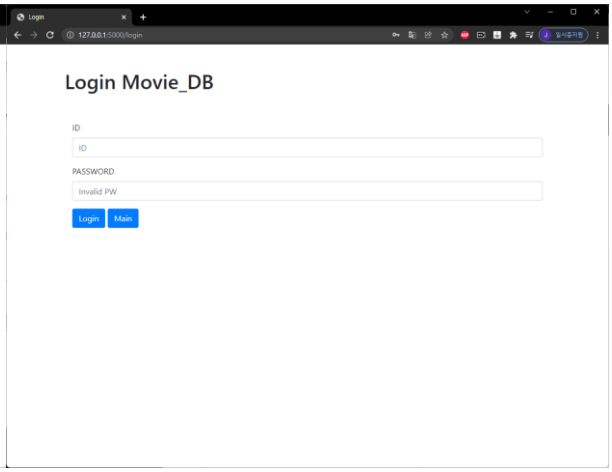
```
userid = request.form.get('userid')
password = request.form.get('password')

cur.execute("SELECT * FROM users;")
users = cur.fetchall()

for user in users:
    if user[0] == userid and user[1] == password:
        session['userid']=userid
        return redirect('/')
    elif user[0] == userid and user[1] != password:
        return render_template("login.html", logmsg='pw_error')
return render_template("login.html", logmsg='id_error')
# if user info correct, login.
```



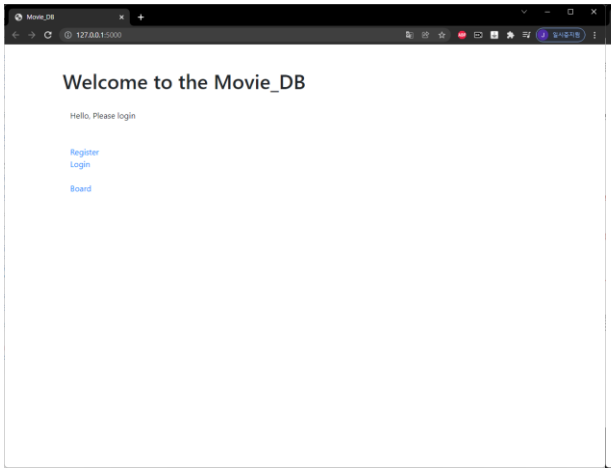
<로그인 페이지>



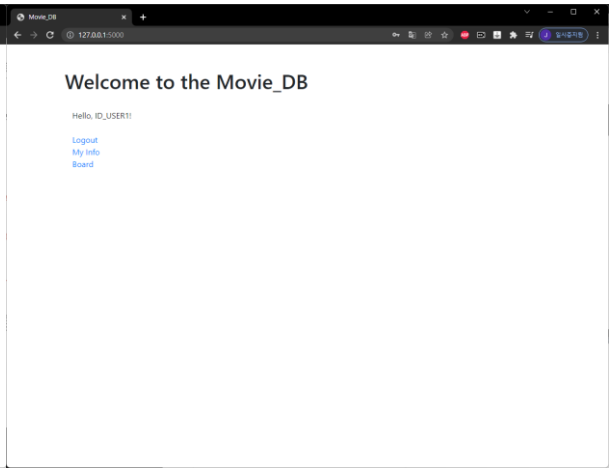
<잘못된 아이디>



<잘못된 비밀번호>



<로그인 이전 메인 페이지>



<로그인 이후 메인 페이지>

(3). 영화 검색 및 정렬 게시판

DB의 영화 목록을 SELECET를 통해 검색한다. 플라스크의 session을 통해 사용자가 게시판 페이지의 버튼 또는 입력칸을 통해 준 조건을 계속 유지한다. 정렬 조건은 1). 영화 평점 2). 제작연도 3). 상영시간 4). 사용자 기준 평점이다. 로그인한 회원은 각 영화에 평점을 주어 자신만의 목록에 추가할 수 있고, 이 목록 상의 영화가 사용자가 준 평점을 기준으로 정렬된다. 회원만이 가능하므로 로그인한 상태에서만 이 정렬 조건을 확인할 수 있다.

검색 조건은 장르 버튼 및 입력 조건이다. 이중 장르 버튼은 모든 장르, 각 세부 장르로 구성되며 정렬 조건을 버튼을 통해 계속 기억하고 있던 것과 마찬가지로 selected된 상태로 남아 있다. 입력 조건은 영화 제목, 감독, 작가, 배우 항목을 골라 텍스트를 입력했을 때 일치하는 데이터를 반환한다.

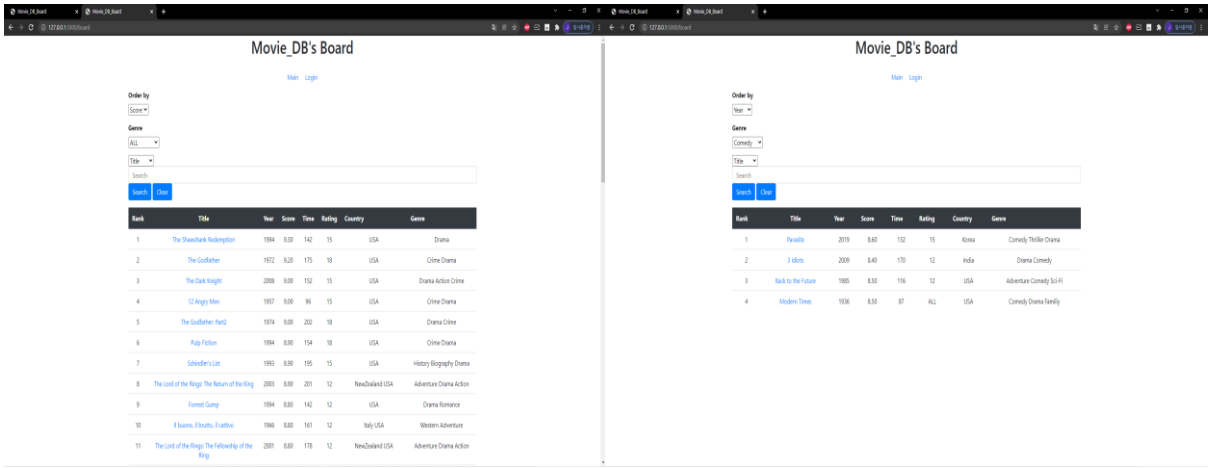
```
if rank != 'mypick':

    if search_type == 'title':
        cur.execute(
            "WITH genre_selected AS (SELECT movie.id AS genre_selected_id
```

```
FROM movie, genre, genre_movie WHERE "
          "movie.id = genre_movie.movie_id AND genre.id =
genre_movie.genre_id AND genre.name LIKE '{}' ) "
          "SELECT movie.id, movie.title, movie.year, movie.score,
movie.running_time, movie.rating, country.country_name, genre.name "
          "FROM movie, country join country_movie on (country.id =
country_movie.country_id), "
          "genre, genre_movie, genre_selected "
          "WHERE country_movie.movie_id = movie.id AND genre_movie.movie_id
= movie.id AND genre_movie.genre_id = genre.id AND movie.id =
genre_selected.genre_selected_id AND "
          "movie.title LIKE '{}' ORDER BY {} DESC;".format('%'+genre+'%',
'%' + search + '%', rank))
```

*SQL JOIN, CARTESIAN PRODUCT 등 사용

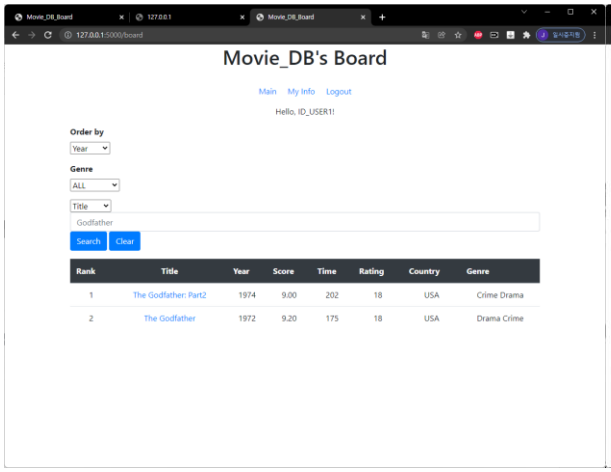
정렬 조건 중 회원 정보를 열람하는 네 번째 경우와 나머지를 나누었고, 입력 조건 중 항목(영화 제목, 감독, 작가, 배우)을 구분한다. 검색을 위해 입력한 데이터와 장르 정보는 SELECT의 LIKE '%' + input + '%'을 통해 공백을 입력하더라도 전체 데이터가 출력될 수 있도록 조정하였다. 위 코드는 정렬 조건이 사용자 기준 평점이 아니고, 검색 유형이 영화 제목인 경우이다(전체 코드는 app.py 참조).

또한 영화 테이블의 어트리뷰트와 이 영화가 제작된 국가명, 해당 장르명 등을 검색한다. 영화는 장르 한 개 이상을 가질 수 있기 때문에 genre_selected라는 테이블을 만들어 genre와 일치하는 영화 아이디를 받아오지 않고 SELECT 문 내에서 곧바로 장르 정보를 검색할 경우, 그 영화가 담고 있는 다른 장르를 읽어올 수 없으므로 주의해야 한다. 이후 SELECT 쿼리문이 반환한 값은 중복 정보가 존재하므로 파이썬 함수 내부에서 중복을 제거, 딕셔너리 파일로 만든 뒤 html을 렌더링할 때 입력값으로 준다.

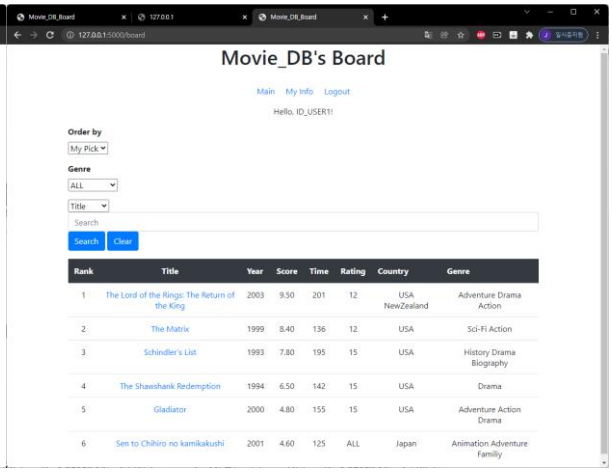


<영화 목록 게시판> <제작연도 정렬, 코미디 장르, 검색 입력 X>

게시판 목록의 디폴트 설정값은 각각 평점 내림차순, 모든 장르, 특정 검색 입력 X이다. 로그인 유무와 관계없이 사용자는 게시판을 통해 자신이 원하는 조건대로 DB 상에 저장된 영화를 다룰 수 있다. 로그인 이후 회원이 사전에 평점을 고른 영화만을 확인할 수 있고, 이 또한 각 장르 및 검색 입력을 통해 데이터를 필터링할 수 있다. 또한 목록 중 영화 제목을 클릭했을 때 상세 정보를 담은 페이지로 렌더링해주었다.



<제작연도 정렬, 모든 장르, ‘Godfather’ 검색>

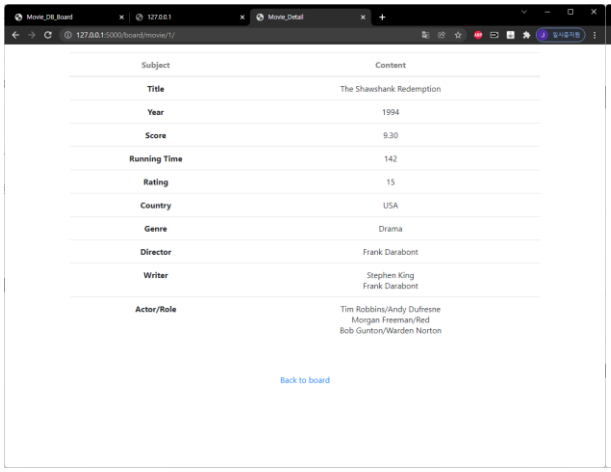


<MyPick 정렬, 모든 장르, 검색 입력 X>

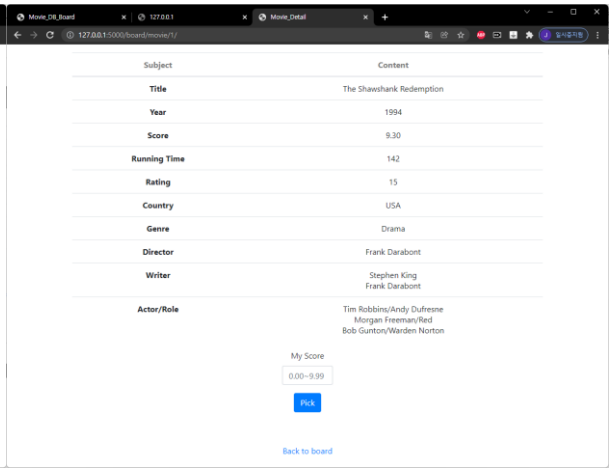
클리어 버튼을 통해 파이썬 내부 session에 기록된 정렬 조건, 장르 조건, 검색값 등을 초기화 가능하다.

(4). 영화 상세 정보 게시판

영화 목록 게시판의 영화 제목을 클릭하면 감독, 작가, 배우/역할 등 상세 정보가 추가된 게시판으로 연결된다.



<영화 상세 정보 게시판, 비회원>

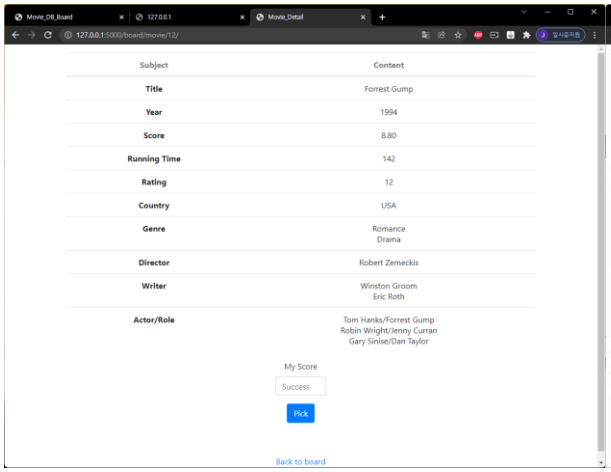


<영화 상세 정보 게시판, 회원>

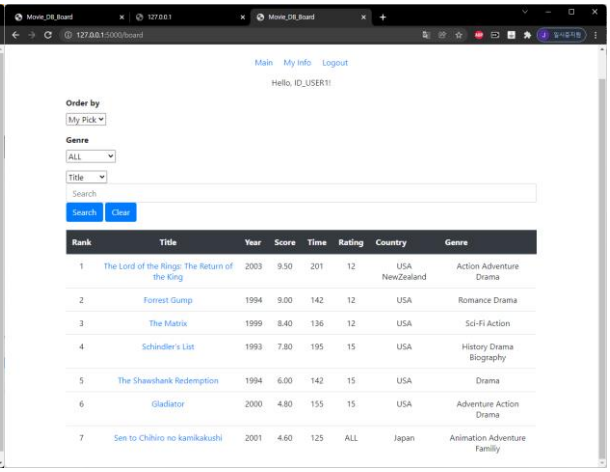
로그인한 경우, 해당 영화에 평점을 주고 목록에 추가할 수 있다. 입력칸에 보이는 0 이상 10 미만 수를 입력해 회원 선택 영화 목록에 추가할 수 있고, 기존 값이 있을 경우 자동으로 갱신되도록 설정하였다. 실패하는 경우는 숫자가 아니거나 범위를 넘어선 경우에 한한다.

```
if scoremsg == 'Success':
    cur.execute("INSERT INTO users_pick (user_id, movie_id, score) VALUES
('{}', '{}', '{}') "
                "ON CONFLICT (user_id, movie_id) DO UPDATE SET score = '{}';"
                .format(userid, movie_id, score, score))
    connect.commit()
```

*SQL INSERT, UPDATE 등 사용



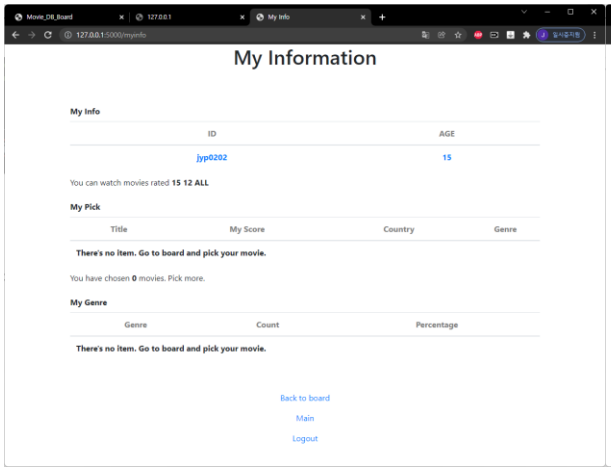
<Forrest Gump 회원 평점 성공>



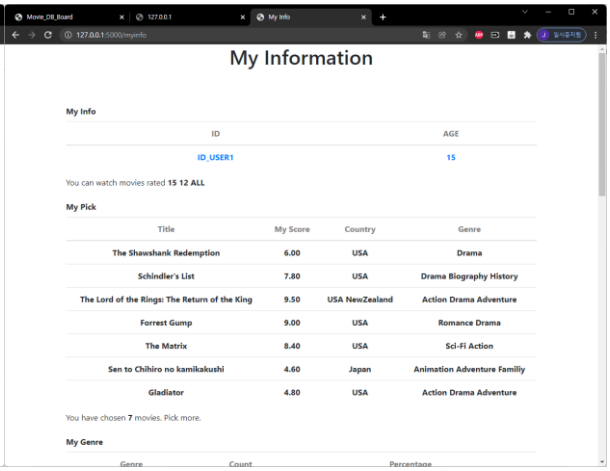
<MyPick을 통한 Forrest Gump 추가 확인>

(5). 회원 정보

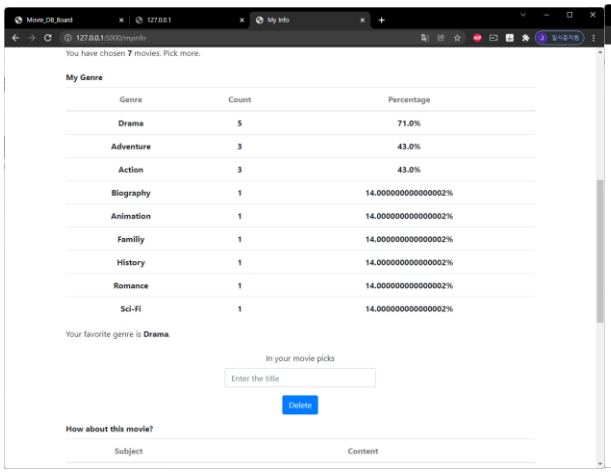
로그인한 경우 메인 페이지 또는 게시판 페이지의 링크를 통해 회원 정보 페이지로 이동할 수 있다. 해당 페이지는 각각 회원 정보(아이디, 나이) 및 수정 가능 링크, 회원이 평점을 준 영화 목록, 본 영화 목록을 통해 구성한 선호 장르 및 추천 영화를 보여준다.



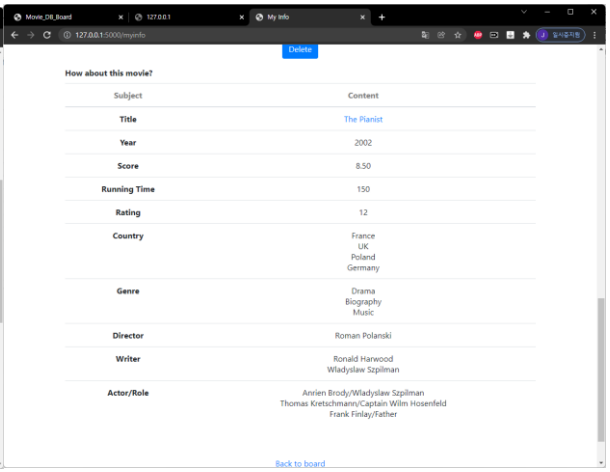
<회원 정보 페이지: 사용자 평점 목록 X>



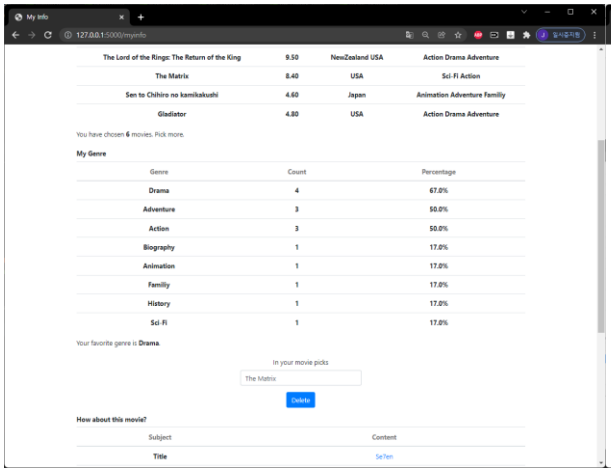
<회원 정보 페이지1: My Info, My Pick>



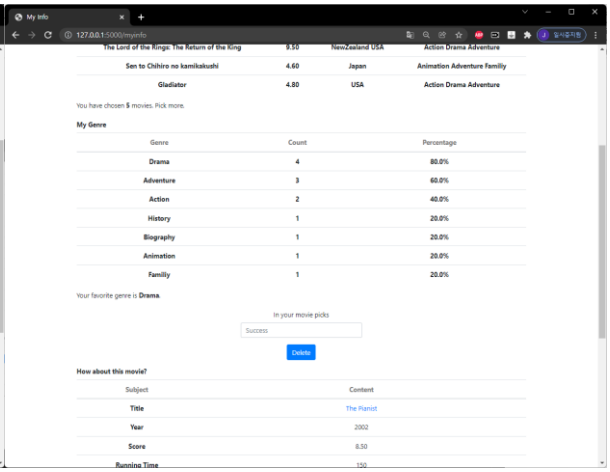
<회원 정보 페이지2: My Genre>



<회원 정보 페이지3: How about this movie?>



<사용자 평점 영화 삭제: The Matrix>



<사용자 평점 영화 삭제 확인>

```
cur.execute("DELETE FROM users_pick WHERE user_id = '{}' AND movie_id = "
        "(SELECT id FROM movie WHERE title = '{}');".format(userid, title))
connect.commit()
```

*SQL DELETE사용

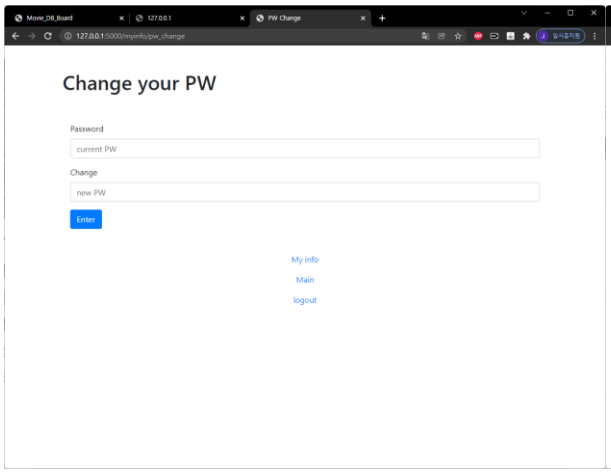
입력한 영화 제목을 통해 users_pick 테이블이 가진 영화와 일치 여부를 확인하고, 정확히 일치한 경우 DELETE 쿼리문을 통해 users_pick의 튜플을 삭제한다.

```
cur.execute("SELECT genre.name, count(*) as num_genre"
            " FROM movie, users_pick, genre, genre_movie"
            " WHERE movie.id = users_pick.movie_id AND"
            " users_pick.user_id = '{}' AND"
            " genre.id = genre_movie.genre_id AND"
            " genre_movie.movie_id = movie.id GROUP BY genre.name ORDER BY
num_genre DESC;".format(userid))
genre_info = cur.fetchall()
total_count = len(users_info)

if genre_info:
    genre_fav = genre_info[0][0]
    cur.execute("SELECT id FROM movie WHERE id IN (SELECT
genre_movie.movie_id FROM genre, genre_movie WHERE "
            "genre_movie.genre_id = genre.id AND genre.name = '{}') "
            "AND id NOT IN (SELECT movie_id FROM users_pick WHERE user_id =
'{}');".format(genre_fav, userid))
    movie_ids = cur.fetchall()
    movie_id = random.sample(movie_ids, 1)
    movie_id = movie_id[0][0]
```

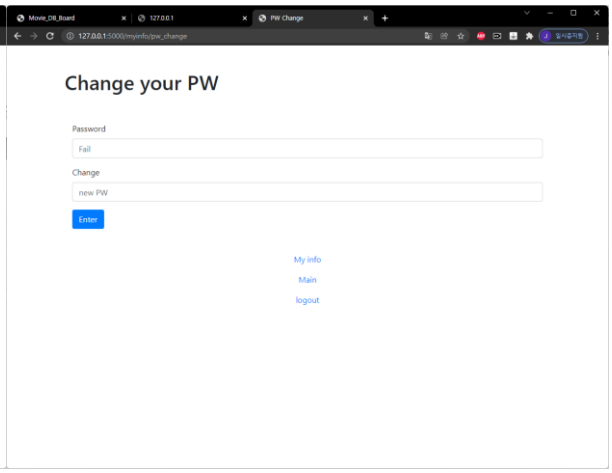
*SQL AGGREGATE(COUNT), NESTED SUBQUERY 사용

회원이 평점을 준 영화 목록이 존재할 때, 어떤 종류의 장르를 선호하는지 보여주는 테이블을 만든다. GROUP BY를 통해 장르별 이름으로 구별해 총 개수를 세고, 전체 영화 중 비율을 보여준다. 이렇게 선호 장르가 결정되면 이를 이용해 다른 영화를 추천할 수도 있다. 즉 선호하는 장르의 영화 중 현재 회원이 평점을 주지 않은 영화를 무작위로 골라 회원 정보 페이지 하단부에 입력하였다. 이 경우 random 함수를 사용하므로 해당 페이지에 접속하는 매 순간 다른 영화를 추천한다.

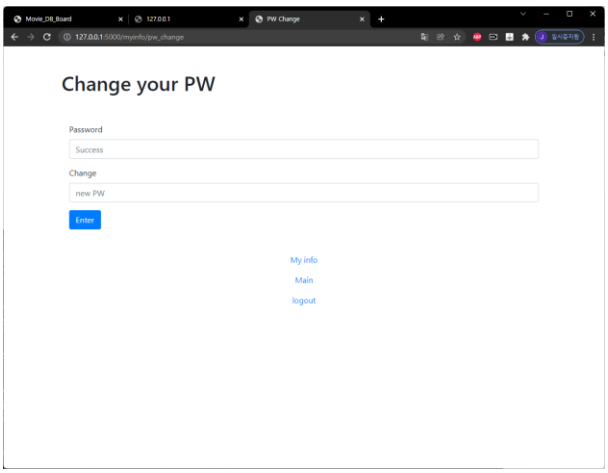
(6). 회원 정보 수정(비밀번호, 나이)



<비밀번호 수정 페이지>



<비밀번호 수정 실패>



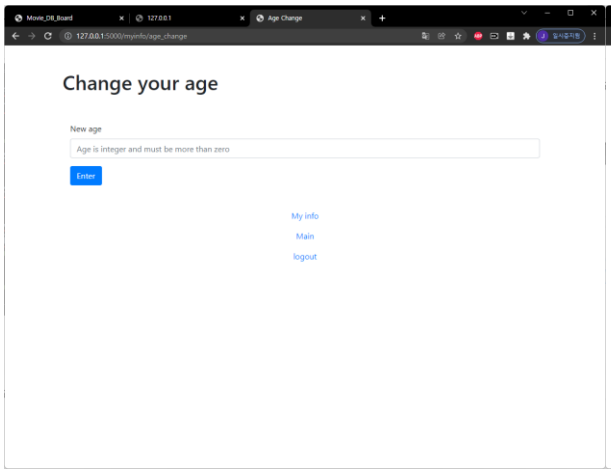
<비밀번호 수정 성공>

```
if request.method == 'POST':
    password = request.form.get('password')
    new_password = request.form.get('new_password')

    cur.execute("SELECT pw FROM users WHERE id = '{}';".format(userid))
    cur_password = cur.fetchall()[0][0]

    if password == cur_password:
        cur.execute("UPDATE users SET pw = '{}' WHERE id =
'{}';".format(new_password, userid))
        connect.commit()
        changemsg = 'Success'
    else:
        changemsg = 'Fail'
# change your pw after current pw confirmed.
```

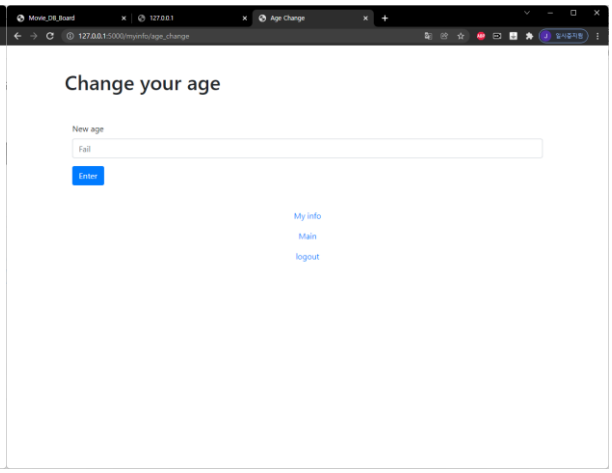
*SQL UPDATE 사용

입력한 비밀번호가 기존 비밀번호와 일치할 때만 비밀번호 갱신이 가능하다. SELECT 문을 통해 비밀번호가 같은지 확인하고, UPDATE 문을 통해 갱신하자.

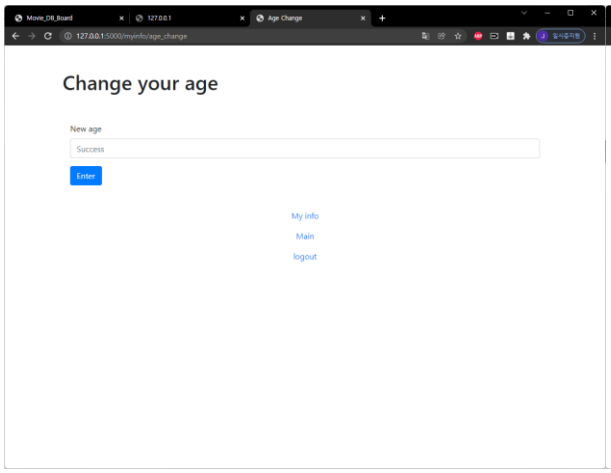
다음은 나이를 변경하는 페이지이다. 유효한 값을 입력한 경우 나이를 변경할 수 있다.



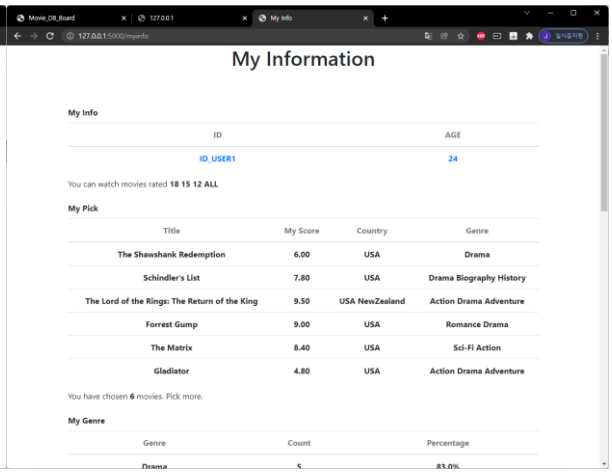
<나이 수정 페이지>



<나이 수정 실패>



<나이 수정 성공>



<15세 → 24세 변경, 관람 가능 목록 변경>

```
if changemsg == 'Success' and (age<=0 or age>100):
        changemsg = 'Fail'
    if changemsg == 'Success':
        cur.execute("UPDATE users SET age = '{}' WHERE id =
'{}';".format(age, userid))
        connect.commit()
# change your age.
```

*SQL UPDATE 사용

비밀번호 변경과 마찬가지로 나이를 UPDATE 문을 통해 갱신 가능하다. 비밀번호가 기존 비밀번호와 일치해야 변경 가능한 경우와 다르게, 나이는 데이터만 유효하다면(숫자이고 사전에 설정한 범위 내부일 때) 곧바로 갱신된다. 그리고 이를 통해 회원 정보 게시판에서 영화 등급에 따른 관람 가능 목록을 자동으로 변경해준다(전체관람가, 12세, 15세 등).