Day5 M:N 팔로우 구현, Fixtures

구분	DB

Day5 팔로우 구현

project.zip

저번 시간에 “좋아요”를 구현했던 파일에 이어서 진행 하겠다. 오늘은

1. 회원별 프로필 페이지 작성 후
2. 팔로우 언팔로우 기능을 구현 할 것이다. 그리고
3. Fixtures 에 대해서 실습 해 볼 것이다. Fixtures가 무엇인지는 조금 이따가 살펴보자.

1. 팔로우 구현하기

앞에서 “좋아요” 기능을 구현하면서 N:M 관계를
장고에서 제공해주는 ManyToManyField 중개기(Bridge Table) 을 통해서
게시글과 <-> User를 연결시켰다.
팔로우를 구현 하는 것은 비슷하면서도 조금 다르다. 팔로우 기능 역시 M:N 관계라는 점은 앞에서
구현한 “좋아요”와 비슷한 상황이지만 중개기를 통해서 연결 할 대상이 조금 다르다.
“좋아요”에서의 M:N 관계는 게시글과 <-> User 였다면
“팔로우”에서의 M:N 관계는 User <-> User 가 될 것이다.

팔로우 기능을 구현하기 앞서 각 유저들의 프로필을 볼 수 있도록 Profile 페이지를
먼저 구현해 보자.

```
# accounts/urls.py

urlpatterns = [
  ...
  path('profile/<username>/', views.profile, name='profile'),
]
```

view 함수로 이동해서 profile 함수를 정의한다.

```
# accounts/views.py

from django.contrib.auth import get_user_model

def profile(request, username):
    User = get_user_model()
    person = User.objects.get(username=username)
    context = {
        'person': person,
    }
    return render(request, 'accounts/profile.html', context)
```

accounts app에 profile.html 파일을 하나 생성하자

```
# accounts/templates/accounts/profile.html

{% extends 'base.html' %}

{% block content %}
  <h1>{{ person.username }}님의 프로필</h1>

  <hr />
  <h2>{{ person.username }}'s 게시글</h2>
  {% for movie in person.movie_set.all %}
    <div>{{ movie.title }}</div>
  {% endfor %}

  <hr />
  <h2>{{ person.username }}'s 댓글</h2>
  {% for comment in person.comment_set.all %}
    <div>{{ comment.content }}</div>
  {% endfor %}

  <hr />
  <h2>{{ person.username }}님이 좋아요를 누른 게시글</h2>
  {% for movie in person.like_movies.all %}
    <div>{{ movie.title }}</div>
  {% endfor %}

  <hr />

  <a href="{% url 'movies:index' %}">back</a>
{% endblock %}
```

그다음 base.html에 내 프로필을 추가하고

index 페이지에 작성자를 클릭시 작성자의 해당 프로필 페이지로 넘어 갈 수 있도록 수정해 보자.

```
# base.html

<body>
  <nav>
    {% if user.is_authenticated %}
      <h3>Hello, {{ user.username }}</h3>
      <a href="{% url 'accounts:profile' user.username %}">내 프로필</a>
      ...
      <a href="{% url 'accounts:update' %}">회원정보수정</a>
```

그리고 movies app의 index 페이지에서 게시글의 작성자가 보이도록 수정하자.

```
# movies/index.html

<p>작성자 : <a href="{% url 'accounts:profile' movie.user.username %}">{{ movie.user }}</a></b>
```

이렇게 user의 프로필 페이지가 완성 되었다. 서버 켜서 확인해 보자.

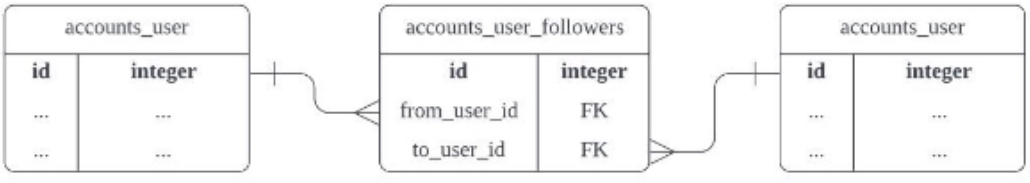
내 프로필 또는 작성자를 누르면 해당 user의 프로필 페이지로 이동 할 것이다.

자 이제 팔로우 기능을 구현 할 준비를 마쳤다.

앞서 언급 한 대로 “팔로우” 에서의 M:N 관계는 User <-> User 가 될 것이다.

즉 이것 또한 “좋아요”와 마찬가지로 브릿지 테이블로 `ManyToManyField` 로 구성되어야 한다.

테이블을 쪼개보면 다음과 같아지는데,



즉, 양쪽에 `accounts_user` 가 있고, 하나의 중개테이블이 차지하고 있는 형태가 된다.

다시 말하자면, 자기 자신에게 ManyToMany 관계를 준 것이라고 할 수 있다.

그래서, ManyToManyField 함수를 사용하기 위해 고려해야 할 사항은 다음과 같다.

1. 첫번째로, 유저가 유저에게 ManyToMany 를 한 경우이므로, 첫번째 인자에 자기 자신을 의미하는`'self'` 를 넣어 줘야 한다.
2. 두번째, 대칭인지 비대칭인지 고려해서 symmetrical 을 지정해야 한다.

   중개 테이블을 자세하게 보면, `from_user_id` 에서 `to_user_id` 로, 오로지 한쪽에서만

   다른 유저에게 팔로우를 할 수 있는 것으로 확인되는데, 이는 한쪽이 팔로우를 했다고 해서

   다른 유저가 똑같이 팔로우함을 의미하지는 않는다. 예를들어 A가 B를 팔로우 했다고 해서 B가 A를 팔로우 하는것은 아니다.

   즉, “대칭이 되지 않는다.” 라고 이야기를 하며 이를 “비대칭 재귀 참조”라고도 한다.

   이러한 구조의 테이블을 만들 때에는 `symmetrical=False` 옵션을 지정해주면 된다.

   그렇다면, `symmetrical=True` 가 되는 관계는 “대칭 되는 관계” 라는 뜻인데,

   어떠한 경우가 있을까? 한쪽이 친구 추가를 하면 다른 쪽도 자동으로 친구가 되어서 양쪽에서 친구로 등록되는 경우를 대칭이라고 할 수 있겠다.

이에, `accounts_user_followers` 를 포함한 모델을 만들어보면 다음과 같다.

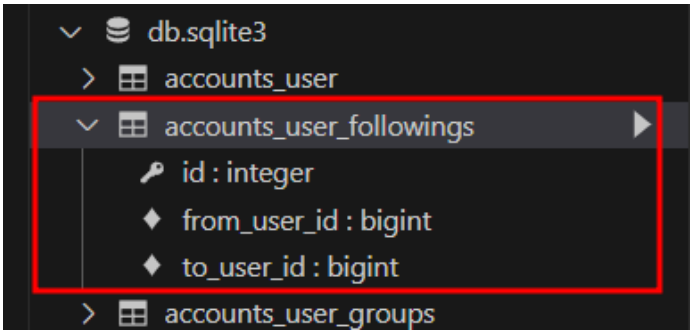
```
from django.db import models
from django.contrib.auth.models import AbstractUser

class User(AbstractUser):
  followings = models.ManyToManyField('self', symmetrical=False, related_name='followers')
```

테이블 이름은 `followings` 이며, 이 안에는 양쪽의 FK 가 담길 것이다. 그리고 접근은 `followers` 로 한다. 다시말해, `related_name='followers'` 로 지정 해 주었기 때문에 해당 테이블에 접근 할 때에는 `followers` 라는 이름으로 접근 하겠다는 것이다.

```
$ python manage.py makemigrations
$ python manage.py migrate
```

SQLITE EXPLORER를 새로고침 후 생성된 중개 테이블을 확인하자.



그리고, `accounts/urls.py` 는 다음과 같이 코드를 추가한다.

```
urlpatterns = [
        ...
        path('<int:user_pk>/follow/', views.follow, name='follow'),
]
```

`accounts/views.py` 는 다음과 같이 코드를 추가한다.

```
@require_POST
def follow(request, user_pk):
    if request.user.is_authenticated:
        User = get_user_model()
        person = User.objects.get(pk=user_pk)
        if person != request.user:
            if person.followers.filter(pk=request.user.pk).exists():
                person.followers.remove(request.user)
            else:
                person.followers.add(request.user)
        return redirect('accounts:profile', person.username)
    return redirect('accounts:login')
```

”좋아요”를 구현 했을때와 비슷 한 코드 이다. 아무리 자기애가 넘치더라도

자기가 자기 스스로에게 팔로우를 하지 않으므로

if person != request.user:

팔로우의 대상(person) 과 팔로워 (request.user)가 달라야 하고

if 만약에 팔로우를 시전한 유저가 이미 팔로워 라면 remove 하고

else 그게 아니라면 팔로우 add 하겠다는 의미가 되겠다.

다음 user의 프로필에 팔로잉 / 팔로워수 그리고 팔로우 언팔로우 버튼을 만들어 보자.

```
# accounts/profile.html

{% extends 'base.html' %}

{% block content %}
  <h1>{{ person.username }}님의 프로필</h1>

  <div>
    <div>팔로잉 : {{ person.followings.all|length }} / 팔로워 : {{ person.followers.all|length }}</div>
    {% if request.user != person %}
      <div>
        <form action="{% url 'accounts:follow' person.pk %}" method="POST">
          {% csrf_token %}
          {% if request.user in person.followers.all %}
            <input type="submit" value="Unfollow" />
          {% else %}
            <input type="submit" value="Follow" />
          {% endif %}
        </form>
      </div>
    {% endif %}
  </div>
```

서버 켜서 테스트를 해보자. 적어도 두 명의 유저로 로그인을 바꿔 보면서, 팔로잉, 팔로우가 잘 동작이 되는지 확인해보자.

sfminho2 아이디로 로그인 후 kevin의 프로필에 들어가 follow를 하고 로그 아웃을 했다.

kevin 아이디로 로그인 후 프로필을 확인 해 보니 팔로워가 1명 있는것이 확인이 되었다.

이상으로 follow 그리고 unfollow 기능까지 구현을 하였다.

마지막으로 Fixtures 에 대해서 알아보자.

2. Fixtures

fixture란?

```
협업시 협업 파트너에게 내가 작업하던 파일을 넘겨 줄 때가 있는데 내가 작업하면서 생성되는 DB데이터는
상대방에게 넘겨주지 않는다. 그 이유는 협업 상대방은 해당 DB가 필요 없을 수도 있고, 또는
해당 프로젝트를 이어 받는 사람이 DB 관리를 위해 sqlite3를 사용하지 않을 수도 있다.

하지만 협업 파트너 입장에서는 DB 데이터가 하나도 없는 상태로 프로젝트 파일만 넘겨 받다보면
데이터가 없는 빈 프로젝트를 가지고는 넘겨받은 프로젝트를 초기에 빠르게 파악하는데 어려움이
있을 수도 있다. 그래서 상대가 프로젝트를 쉽게 파악하고 간단한 테스트를 할수 있도록
초기 initial data를 제공해 주곤 한다.즉, Fixture란? Django에서 간단한 테스트를 실행하는
환경을 위해 실제 DB의 데이터를 Json 형식 또는 Xml 형식을 이용하여 초기 DB를 셋팅하는 것을
말한다.
```

우리도 초기 데이터로 json 파일을 넣어 봄으로써 현재 프로젝트가 잘 작동되는지 확인하고 자 한다.

1. 위에서 진행한 프로젝트에 이어서 생성한 데이터 들을 추출 (`dump data`) 하는 것을 실습 하고
2. 추출 후에는 DB를 깨끗하게 비운 후
3. 아까 추출한 데이터를 다시 데이터 입력 (`load data`) 하는 것을 실습 해 볼 것이다.

Dump data and Load data 실습

각각의 앱의 모델을 json형식의 파일로 각각 추출할 것이다.

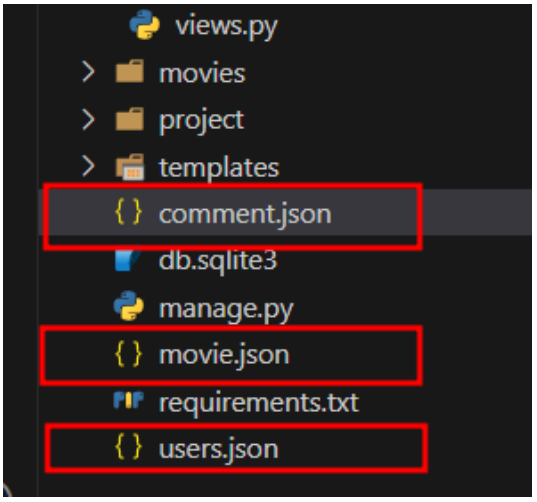
```
$ python -Xutf8 manage.py dumpdata --indent 4 accounts.User > users.json

$ python -Xutf8 manage.py dumpdata --indent 4 movies.Movie > movie.json

$ python -Xutf8 manage.py dumpdata --indent 4 movies.Comment > comment.json
```

라이브 교재와 다르게 -Xutf8 를 적어 주었는데 이는 한글깨짐 UTF-8 인코딩 문제를 사전에 방지하기 위해서 적어 두었다.

[참고] python - Django dumpdata fails on special characters - Stack Overflow



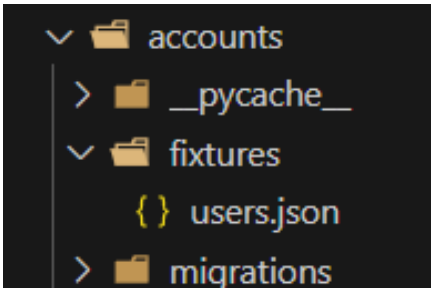
자 이렇게 json 형식으로 data를 추출에 성공 했다면

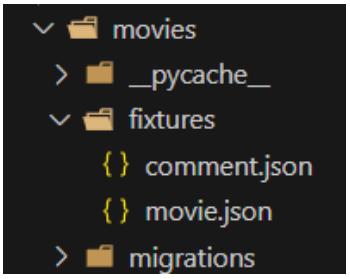
이번에는 `load data`를 해보자. load 를 실습 하기 전에

각각의 앱에 (accounts and movies) fixture 디렉터리 생성 후 json 파일을 배치 할 것이다.

/movies/fixtures/ 폴더를 생성 후 movie.json 파일과 comment.json 파일을 위치 시키고

/accounts/fixtures/ 폴더를 생성 후 users.json 파일을 위치 시킬 것이다.



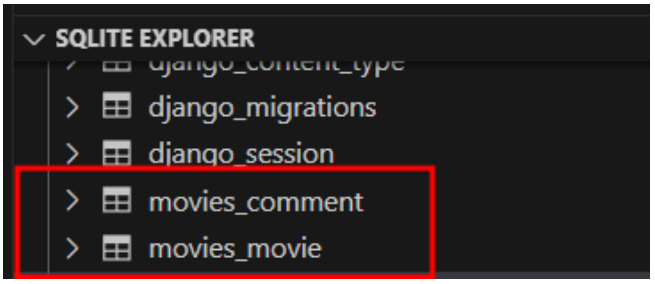


그리고 data들을 load하기 전에 DB를 깨끗하게 정리를 하자.

각각의 앱 (account 그리고 movies) 의 migrations 폴더에서 init을 제외하고 0001 0002 등의 파일은 모두 지운 후, db.sqlite3 도 지워버리자.

```
$ python manage.py makemigrations
$ python manage.py migrate
```

explorer를 통해서 데이터가 모두 날아간 것을 확인 해 보자.



그리고 아래의 명령어를 통해서 fixtures 전체 load 해 볼 것이다.

```
$ python manage.py loaddata movie.json comment.json users.json

$ python manage.py migrate
```

그리고 SQLite 를 다시 열어, 새로고침 후 movie 쪽의 테이블들을 확인해보자. 그러면 다시 data들이 모두 들어와 있을 것이다. <끝>

project.zip