Django _ Movies,Comments 구현

3. movies and comments

3. movies and comments

`movies/urls.py` 는 다음 제시된 설계에 맞춰 작성되어야 한다.

URL 패턴	역할
/movies/	전체 영화 목록 페이지 조회
/movies/create/	새로운 영화 생성 페이지 조회 & 단일 영화 데이터 저장
/movies/<pk>/	단일 영화 상세 페이지 조회
/movies/<pk>/update/	기존 영화 수정 페이지 조회 & 단일 영화 데이터 수정
/movies/<pk>/delete/	단일 영화 데이터 삭제
/movies/<pk>/comments/	단일 댓글 데이터 저장
/movies/<movie_pk>/comments/<comment_pk>/delete/	단일 댓글 데이터 삭제

이를 통해 구현한 `movies/urls.py` 는 다음과 같다.

```
# movies/urls.py

from django.urls import path
from . import views


app_name = 'movies'

urlpatterns = [
    path('', views.index, name='index'),
    path('create/', views.create, name='create'),
    path('<int:pk>/', views.detail, name='detail'),
    path('<int:pk>/delete/', views.delete, name='delete'),
    path('<int:pk>/update/', views.update, name='update'),
    path('<int:pk>/comments/', views.comments_create, name='comments_create'),
    path('<int:movie_pk>/comments/<int:comment_pk>/delete/', views.comments_delete, name='comments_delete'),
]
```

중요한 건, FK 가 포함된 댓글쪽이다. 댓글을 생성할 때 어떤 영화의 댓글을 생성할지 알아야 하기 때문에, 해당 영화 PK 가 필요하다. 또한, 삭제를 할 땐, 어떤 영화의, 어떤 댓글인지 알아야하기에 url 에 PK 는 총 두 개 필요하다.

에러 방지를 위해, `movies/views.py` 에 함수들을 미리 정의해두자.

```
# movies/views.py

from django.shortcuts import render

# Create your views here.
def index(request):
    return render(request, 'movies/index.html')

def create(request):
    pass

def detail(request):
    pass
```

```
def update(request):
    pass

def delete(request):
    pass

def comments_create(request):
    pass

def comments_delete(request):
    pass
```

관리자 페이지를 활용해 영화 데이터를 샘플로 5개 정도만 만들어본 후, 사용자가 `index` 에 접속하면 저장된 모든 목록이 나오도록 해보자.

```
# movies/views.py

from django.views.decorators.http import require_safe
from .models import Movie

@require_safe
def index(request):
    movies = Movie.objects.all()
    context = {
        'movies': movies,
    }
    return render(request, 'movies/index.html', context)
```

`require_safe` 는, 사용자가 해당 페이지에 접속할 때 정보 확인 외에 다른 행동 (이를테면 데이터 작성) 등을 못하게 방지하는 데코레이터이다. 사용자는 인덱스 페이지에 접속할 때 오로지 영화 목록만 확인하기 때문에 해당 데코레이터를 사용했다.

Movies 의 모든 데이터를 가져와 `index.html` 으로 보내는 간단한 ORM 구문이다.

`index.html` 을 다음과 같이 작성하자.

```
# movies/index.html

{% extends 'base.html' %}

{% block content %}
  <h1>INDEX</h1>
  <a href="{% url 'movies:create' %}">[CREATE]</a>
  <hr>
  {% for movie in movies %}
    <a href="{% url 'movies:detail' movie.pk %}"><p>{{ movie.title }}</p></a>
    <hr>
  {% endfor %}
{% endblock %}
```

사용자가 영화를 생성할 수 있도록, CREATE 링크를 하나 걸었다.

Django 의 for 반복을 사용해 각각의 `title` 을 찍고, 각각을 클릭했을 때 해당 영화의 디테일 페이지가 보이도록 만들었다.

이제 영화 생성, `create` 를 구현해보자. 먼저, `movies/forms.py` 를 만들자.

```
# movies/forms.py

from django import forms
from .models import Movie, Comment


class MovieForm(forms.ModelForm):
    class Meta:
        model = Movie
        fields = ('title', 'description',)


class CommentForm(forms.ModelForm):
    class Meta:
        model = Comment
        exclude = ('movie', 'user',)
```

각각, `MovieForm` 은 `title`, `description` 필드를 작성할 수 있도록 한다. 왜? `__all__` 로 둘 경우, 유저를 선택할 수 있게 되는데, 이는 잘못된 것이다. 영화 작성 시 반드시 현재 접속된 사용자가 해당 영화의 작성자가 되어야 하기 때문이다.

`CommentForm` 은 `movie` 와 `user` 를 제외해서 작성할 수 있도록 하겠다. 왜? `__all__` 로 둘 경우, 어떤 영화이며, 어떤 유저가 썼는지를 사용자가 결정할 수 있는데, 이는 잘못된 것이다. 댓글 작성 시 반드시 현재 접속된 사용자가 해당 영화에 대한 댓글을 써야하기 때문이다.

다음, `views.py` 에서 영화 작성을 할 수 있는 `create` 함수를 작성해보자.

```
# movies/views.py

from django.shortcuts import render, redirect
from django.views.decorators.http import require_safe, require_http_methods
from django.contrib.auth.decorators import login_required
from .forms import MovieForm, CommentForm

@login_required
@require_http_methods(['GET', 'POST'])
def create(request):
    if request.method == 'POST':
        form = MovieForm(request.POST)
        if form.is_valid():
            movie = form.save(commit=False)
            movie.user = request.user
            movie.save()
            return redirect('movies:index')
    else:
        form = MovieForm()
    context = {
        'form': form,
    }
    return render(request, 'movies/create.html', context)
```

반드시 사용자는 로그인된 상태여야 하기에 `@login_required` 를 사용했다.

요청이 들어왔을 경우, `accounts` 에서와 마찬가지로 GET 과 POST 두 경우로 나뉜다. GET 이라면 단순히 화면에 폼을 보여주는 역할만 하고, POST 라면 사용자가 작성한 영화를 저장하는 역할을 하며, 요구사항과 일치하지 않을 경우엔, 사용자가 입력한 정보들을 그대로 담아 다시 영화 생성 페이지를 보여준다.

여기서, `commit=False` 가 쓰였다. 당장 폼 내용을 적용하지 않고, 추가적인 정보를 넣고 싶을 때 사용하는 옵션이다. 여기선 반드시 필요한데, “어떤 유저” 가 영화를 작성하는지에 대한 정보가 빠져있기 때문이다. 따라서, 유저에 대한 정보를 넣은 다음에야 `save` 함수를 호출하고, 인덱스 페이지로 리다이렉트하게 된다.

`movies/create.html` 은 다음과 같다.

```
# movies/create.html

{% extends 'base.html' %}

{% block content %}
  <h1>CREATE</h1>
  <hr>
  <form action="{% url 'movies:create' %}" method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Submit</button>
  </form>
  <hr>
  <a href="{% url 'movies:index' %}">BACK</a>
{% endblock %}
```

상세한 설명은 생략하겠다. 테스트해보도록 하자.

로그인하지 않은 상태라면, 로그인 페이지로 리다이렉트된다. 왜? `@login_required` 가 해당 역할을 하기 때문이다.

현재 jony123 이라는 아이디로 로그인한 상태에서 제출하면, 인덱스 페이지로 향할 것이다. 데이터가 잘 들어갔는지 관리자 페이지에서 확인해보자.

다음은 영화 상세 페이지다. `movies/views.py` 의 `detail` 함수를 다음과 같이 작성한다.

```
# movies/views.py

@require_safe
def detail(request, pk):
    movie = Movie.objects.get(pk=pk)
    comment_form = CommentForm()
    comments = movie.comment_set.all()
    context = {
        'movie': movie,
        'comment_form': comment_form,
```

```
        'comments': comments,
    }
    return render(request, 'movies/detail.html', context)
```

역시 `@require_safe` 를 사용했는데, 해당 페이지는 단순히 특정 영화의 세부 정보를 확인하는 용도이기 때문이다.

이번엔 url 파라미터로 pk 를 줬기 때문에, 전부 다가 아니라 해당 영화를 가져오는 ORM 구문을 사용했다. 댓글을 작성하기 위한 `CommentForm` 도 가져오고, 가져온 영화에 해당하는 댓글은 `comment_set.all()` 으로 가져올 것이다. (역참조)

이후 `context` 를 만들어서 `detail.html` 로 보내버린다.

다음, `detail.html` 은 다음과 같다.

```
# movies/detail.html

{% extends 'base.html' %}

{% block content %}
  <h1>DETAIL</h1>
  <hr>
  <div>
    <h5>{{ movie.title }}</h5>
    <p>{{ movie.description }}</p>
    {% if user == movie.user %}
      <a href="{% url 'movies:update' movie.pk %}">UPDATE</a>
      <form action="{% url 'movies:delete' movie.pk %}" method="POST">
        {% csrf_token %}
        <button type="submit">DELETE</button>
      </form>
    {% endif %}
  </div>
  <a href="{% url 'movies:index' %}">BACK</a>
  <hr>
  <h4>댓글 목록</h4>
  <ul>
    {% for comment in comments %}
      <li>
        {{ comment.content }}
        {% if user == comment.user %}
          <form action="{% url 'movies:comments_delete' movie.pk comment.pk %}" method="POST" style="display: inline;">
            {% csrf_token %}
            <input type="submit" value="DELETE">
          </form>
        {% endif %}
      </li>
    {% endfor %}
  </ul>
  <hr>
  <form action="{% url 'movies:comments_create' movie.pk %}" method="POST">
    {% csrf_token %}
    {{ comment_form }}
    <input type="submit">
  </form>
{% endblock  %}
```

`if user == movie.user` 부분을 통해, 해당 영화를 작성한 사용자만 `update` 와 `delete` 를 할 수 있도록 했다. `delete` 는 별도의 페이지 없이 바로 POST 가 진행되어야 하기 때문에 단순 `<a>` 태그로 처리한 `update` 와는 다르게 `<form>` 태그를 사용했다. 어찌 되었든 둘 다 `movie.pk` 를 넘겨 준다는 것에 유의하자.

또한 영화에 대한 댓글을 출력해서 사용자에게 보여 주도록 하는데, 댓글 역시 오로지 작성한 사용자만 지울 수 있도록 처리했고, 사용자가 댓글을 작성할 수 있는 항목도 마련했다.

DETAIL

어벤져스: 인피니티 워

타노스의 침략으로 멸망의 위기에 처한 세계를 어벤져스는 구할 수 있을까?

BACK

댓글 목록

Content: 제출

내가 생성한 영화가 아니라면 다음과 같이 보여지고,

DETAIL

바람과 함께 사라지다

진짜 바람과 함께 사라졌을까...

UPDATE
DELETE
BACK

댓글 목록

Content: 제출

내가 생성한 영화라면, UPDATE 와 DELETE 가 가능하도록 만들었다. 둘 다 댓글을 쓸 수 있는 칸은 확인된다.

마지막으로, `movies/views.py` 의 `create` 함수의 `redirect` 부분을, `index` 가 아니라 방금 생성한 영화의 디테일로 보내도록 바꿔주자.

```
# movies/views.py -> create

if form.is_valid():
    movie = form.save(commit=False)
    movie.user = request.user
    movie.save()
    return redirect('movies:detail' movie.pk)
```

영화를 생성했을 때, 인덱스 페이지가 아니라, 해당 영화의 디테일 페이지로 가는지 확인해보자.

다음은 영화 삭제다. `movies/views.py` 의 `delete` 함수를 다음과 같이 작성하자.

```
# movies/views.py

from django.views.decorators.http import require_safe, require_http_methods, require_POST

@require_POST
def delete(request, pk):
    movie = Movie.objects.get(pk=pk)
    if request.user.is_authenticated:
        if request.user == movie.user:
            movie.delete()
    return redirect('movies:index')
```

해당 영화를 가져온 후, 로그인이 되어있을 때만 동작하고 로그인 되지 않은 유저라면 인덱스로 리다이렉트 해버린다.

그리고 조건이 하나 더 있는데, 삭제하려는 해당 영화의 작성자가 현재 로그인한 작성자인지 한번 더 확인한 후에 삭제를 진행하며, 모든 과정이 끝나면 인덱스 페이지로 리다이렉트한다.

특정 영화의 상세 페이지로 넘어간 후, 삭제가 잘 되는지 테스트해보자.

다음은 영화 내용 수정이다.

```
# accounts/views.py

@login_required
@require_http_methods(['GET', 'POST'])
def update(request, pk):
    movie = Movie.objects.get(pk=pk)
    if request.user == movie.user:
        if request.method == 'POST':
            form = MovieForm(request.POST, instance=movie)
            if form.is_valid():
                form.save()
                return redirect('movies:detail', movie.pk)
        else:
            form = MovieForm(instance=movie)
    else:
        return redirect('movies:index')
    context = {
        'movie': movie,
        'form': form,
    }
    return render(request, 'movies/update.html', context)
```

어떤 게시물을 업데이트하는지 알아야 하기 때문에, url 파라미터로 movie.pk 를 받았다. 또한, 수정하려는 사람은 해당 영화를 등록한 사람이어야만 하기에 `if requres.user == movie.user` 로 판단한다. 만약, 아니라면 인덱스 페이지로 리다이렉트될 것이다.

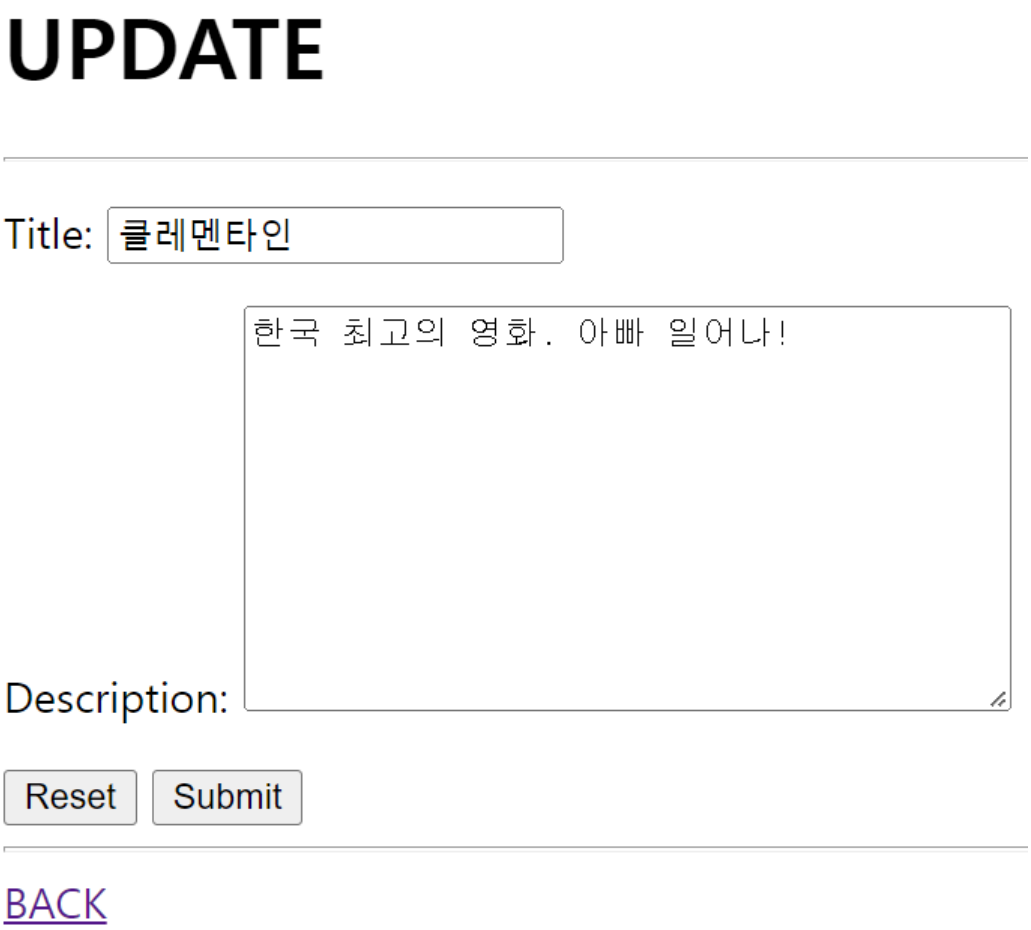
여기서도 GET 인지 POST 인지에 따라 `update` 함수의 역할이 달라지는데, GET 일 경우 단순히 보여주기만 해야 되지만, 중요한 건 “어느 것” 을 보여 줄지 이므로, instance 를 movie 로, 즉 사용자가 선택한 단일 영화로 받아 폼을 만들고, `context` 를 구성해 `update.html` 로 넘겨주었다.

만약 POST 즉 사용자가 폼에 기록을 한 내용을 받는다면, 해당 영화에 작성 되어야 하니 역시 `instance=movie` 를 파라미터로 줘야 하고, 유효한지 체크한 다음 폼을 저장한 후, 수정한 해당 영화의 디테일 페이지로 넘어간다. 만약, 유효성 체크를 통과하지 못한다면, 단순히 그 영화에 대한 폼만 채운 다음 업데이트 페이지로 넘긴다. 폼만 채워 넘길 경우 사용자가 다른 값으로 이미 수정했다 하더라도, 다시 원래의 영화 제목과 내용이 들어갈 것이다.

`movies/update.html` 은 다음과 같다.

```
{% extends 'base.html' %}

{% block content %}
  <h1>UPDATE</h1>
  <hr>
  <form action="{% url 'movies:update' movie.pk %}" method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="reset" value="Reset">
    <button type="submit">Submit</button>
  </form>
  <hr>
  <a href="{% url 'movies:detail' movie.pk %}">BACK</a>
{% endblock %}
```



업데이트가 정상 작동하는지 테스트해보자.

다음, 댓글 생성이다. 영화 상세 페이지에서 댓글 생성 가능하며, 이를 위한 `views.py` 의 `comment_create` 함수는 다음과 같다.

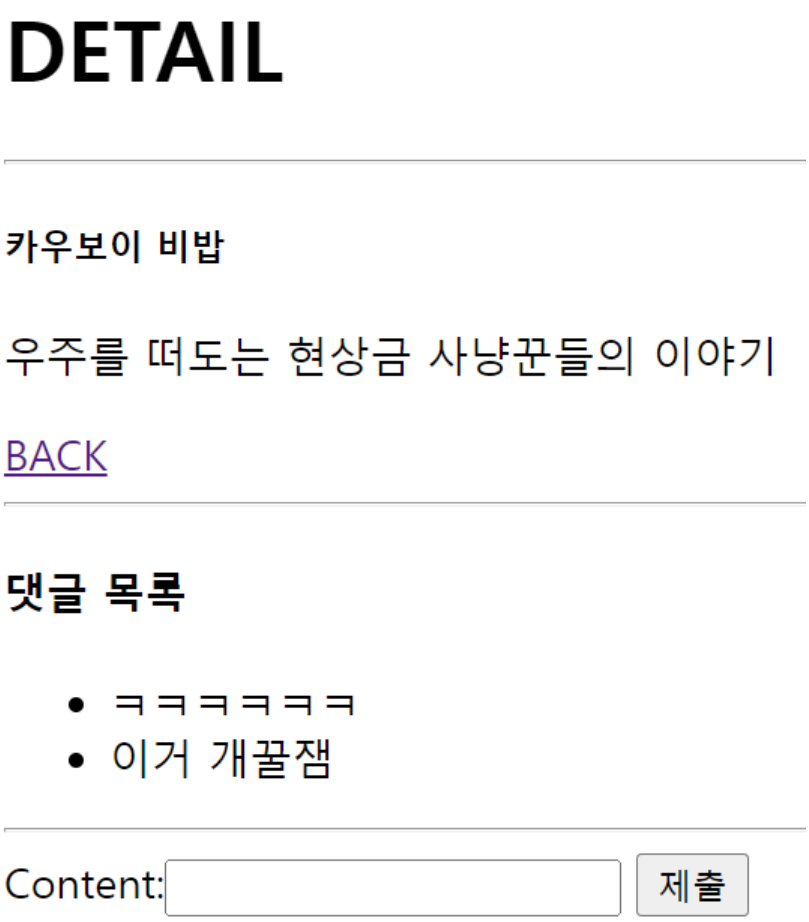
```
# movies/views.py

@require_POST
def comments_create(request, pk):
    if request.user.is_authenticated:
        movie = Movie.objects.get(pk=pk)
        comment_form = CommentForm(request.POST)
        if comment_form.is_valid():
            comment = comment_form.save(commit=False)
            comment.movie = movie
            comment.user = request.user
            comment.save()
        return redirect('movies:detail', movie.pk)
    return redirect('accounts:login')
```

우선, `pk` 는 `urls.py` 를 확인해봤을 때, “영화” 의 `pk` 를 의미한다.

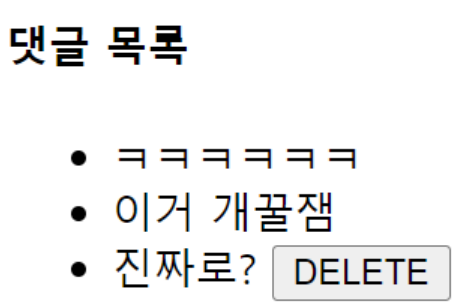
댓글은 로그인한 사람만 작성할 수 있으며, 로그인하지 않았을 경우, 로그인 창으로 리다이렉트된다.

`commit=False` 로, 필요한 정보를 추가할 것이다. 댓글의 대상이 어떤 영화인지 그리고 어떤 유저인지가 `CommentForm` 에 빠져있기 때문이다. 이는 `forms.py` 에 가면 확실히 알 수 있는데, 일반 사용자가 조작하지 못하도록 `exclude` 처리했기 때문이다. 해당 정보까지 포함한 후, 저장하고, 생성된 해당 영화의 디테일페이지로 보낸다.



로그인을 하지 않은 사람이 댓글을 작성 하려하면 로그인 창으로 이동한다.

해당 댓글을 작성하지 않은 사용자라면, 댓글을 지울 권한이 없다.



만약 댓글 작성자라면, 해당 댓글을 지울 수 있는 권한이 생긴다.

마지막으로, 댓글 삭제를 구현해보자.

`movies/views.py` 의 `comments_delete` 는 다음과 같다.

```
# movies/views.py

from .models import Movie, Comment
```

```
@require_POST
def comments_delete(request, movie_pk, comment_pk):
    if request.user.is_authenticated:
        comment = Comment.objects.get(pk=comment_pk)
        if request.user == comment.user:
            comment.delete()
    return redirect('movies:detail', movie_pk)
```

파라미터를 총 두 개 넘겨 받는다. `urls.py` 를 참고하면, 영화의 PK 도 필요하고, 댓글의 PK 도 필요함을 알 수 있다. 즉, 어떤 영화의 어떤 댓글을 지울 것인지 둘 다 알아야 한다.

삭제한 후, 상세 페이지로 리다이렉트한다.

이상으로 1편에서는 회원가입 (accounts) 기능을 구현하고 이번 2편에서는 게시글(movies) 그리고 댓글(comments) 을 작성하는 기능을 구현 하였다. <끝>