Django 활용한 웹페이지 구현 연습 2

자료	DB
구분	DB
과목	

3. Movies

이번에는 영화관련 게시글을 포스팅 할 수 있는 게시판을 구현 할 것이다.

그리고 게시판의 게시글에는 댓글을 작성할 수 있는 기능까지 추가 할 것이다.

`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'),
]
```

여기서 눈 여겨서 봐야 할 것은 댓글(comments) 부분이다.

댓글을(comments) 생성할 때에는 어떤 게시글에 속할 comments인지 알아야 한다. 따라서

해당 게시글의 PK 가 필요하다.

또한, 삭제를 할 때에도 특정 게시글에 속한 comments 중 어떤 댓글을 삭제할 것인지 명확하게 지정해 주기 위해서 url 에 들어가는 PK 는 총 두 개가 되겠다. (게시판의 PK + 댓글의 PK)

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

```
# movies/views.py

from django.shortcuts import render


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
```

`index` 페이지 부터 만들어 보자.

관리자 페이지를 (admin) 활용해서 영화 데이터를 샘플을 5개 정도만 만들어본 후,

사용자가 `index` 에 접속하면 저장된 모든 영화 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)
```

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

`@require_safe` 는, 사용자가 해당 페이지에 접속할 때 정보 확인 외에 다른 행동 (이를테면 데이터 작성) 등을 못하게 방지하는 데코레이터이다. 유저는 인덱스 페이지에 접속할 때 오로지 영화 목록만 확인하기 때문에 해당 데코레이터를 사용했다. 사실 기능상 GET 요청에 대해서는 `@require_GET` 데코레이터를 사용해도 무방하다.

[참고] requrire_safe 데코레이터란?
- view함수가 GET 또는 HEAD method만 허용하도록 요구하는 데코레이터
- django는 require_GET 대신 require_safe를 사용하는 것을 권장하기 때문에 사용했다.

`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 셋팅을 했다.

참고로 아직 detail을 구현하지 않았기 때문에 클릭해도 아직은 뭐가 나오지는 않을 것이다.

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` 필드만 작성할 수 있도록 했다.

그 이유는 `__all__` 로 둘 경우, 게시글을 작성 할 유저를 선택 할 수 있게 되는데, 이는 잘못된 것이다.

게시글 작성은 반드시 “현재 접속 된 유저”가 해당 게시글의 “작성자”가 되어야 하기 때문이다.

`CommentForm` 은 `movie` 와 `user` 를 제외해서 작성할 수 있도록 하겠다.
그 이유는 `__all__` 로 둘 경우, 댓글을 작성시, 해당 게시글을 선택을 하게 될 뿐더러 어떤 유저가 작성하는 댓글인지를 사용자가 결정 할 수 있게 된다. 이는 말도 안되는 기능이다.

댓글 작성 시 반드시 현재 접속된 사용자 (로그인 되어있는 유저)가 댓글의 작성자가 되어야 할 것이며,
어떤 게시글에 대한 댓글인지는 유저가 현재 열람 중인 게시글에 대한 댓글이 되어야 한다.

다시말해 A라는 유저가 댓글을 작성하는데 작성자를 B라고 바꿀 수 있는 기능이 있으면 안될 것이며
명량영화에 관한 게시글을 보다가 다는 댓글은 당연히 명량 영화에 관한 댓글이어야 한다는 것이다.

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)
```

게시글을 작성시 로그인이 된 유저만 게시글을 작성 할 수 있도록 `@login_required` 를 사용했다.
HTTP 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 할 것이다. 유저가 유효하지 않은 데이터를 입력을 했을 경우에는 유저가 입력한 정보들을
그대로 담아 다시 영화 생성 페이지를 보여준다.

유저가 작성한 데이터가 유효하다면 DB에 저장을 하면 되는데

여기서 아래 두 구문을 살펴보자.

```
movie = form.save(commit=False)
movie.user = request.user
```

여기서, `commit=False` 가 쓰였다. 당장 폼 내용을 DB에 적용하지 않고, 추가적인 정보를 더 넣고 싶을 때 사용하는 옵션이다. DB에 저장요청을 보내지 않고 우리가 적어준 movie 객체에 인스턴스만 반환 하는 기능이다.

여기에서는 `commit=False` 옵션이 반드시 필요한 이유는

지금 form안에는 "어떤 유저" 가 해당 게시글을 작성 하는지 에 대한 정보가 빠져있기 때문이다.

따라서, `movie.user=request.user` 를 통해서 해당 게시글을 작성한 유저에 대한 정보를 객체에 넣은 다음에야 `save` 함수를 호출하고, 인덱스 페이지로 리다이렉트 하게 된다.

`movies/create.html` 은 다음과 같다.

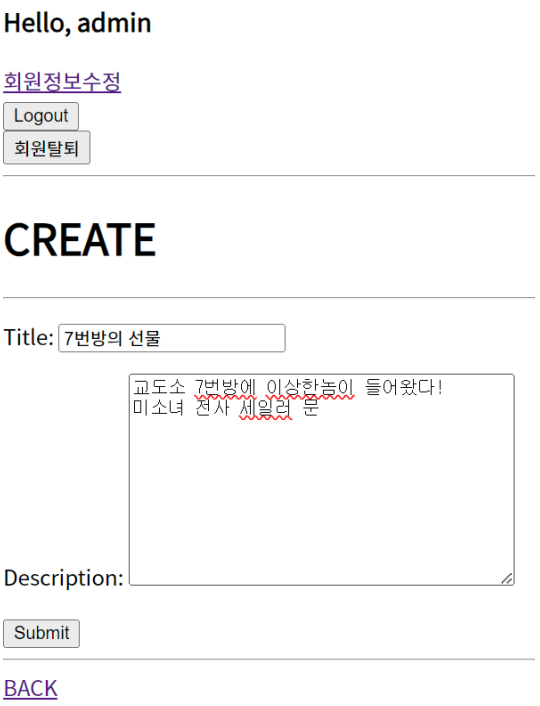
```
# movies/create.html

{% 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 %}
```

반복되는 설명은 생략하겠다. 테스트해보도록 하자.

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



현재 admin 으로 로그인한 상태에서 제출하면, 인덱스페이지로 향할 것이다.

데이터가 잘 들어갔는지 관리자 페이지의 MOVIES에 들어가서 확인을 한번 해봐도 좋겠다.

지금까지 영화 게시글 생성 (Create)파트를 마쳤다.

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)
```

위 코드를 분석해보자.

`@require_safe` 를 사용했는데, 해당 페이지는 단순히 특정 영화의 세부 정보를 확인하는 용도이기 때문에 GET 요청만 허용하면 되기 때문이다.

이번에는 url 파라미터로 pk 를 줬기 때문에, 전부 다가 아니라 (all 이 아니라) 해당 영화만 가져오는 ORM 구문을 사용했다. (단일 데이터 가져오기 Movie.objects.get(pk=pk))

댓글을 작성하기 위한 `CommentForm` 도 가져온다.

해당 게시글의 모든 댓글을 DB에서 가져오기 위해서 `comments = movie.comment_set.all()` 을 사용했다. 이를 "역.참.조" 라고 한다.

역참조는 1:N 관계에서 1 (게시글) 이 N (댓글들)을 참조하거나 조회하는 경우를 말한다.

댓글은 models.py에서 comment 정의를 할때 movie라는 속성을 이용해서 외래키를 가지고 있다.

하지만 게시글 입장에서 N (댓글들)에 대한 참조할 외래키는 존재하지 않지만

역참조 키워드를 이용해서 참조가 가능한 것이다.

[참고] 우리가 정의 했던 모델

```
# movies/models.py
class Movie(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
                             on_delete=models.CASCADE)
    title = models.CharField(max_length=20)
    description = models.TextField()


class Comment(models.Model):
    movie = models.ForeignKey(Movie, on_delete=models.CASCADE)
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    content = models.CharField(max_length=100)
```

[참고] 역참조를 사용한 코드

```
# movies/views.py

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

역참조 `movie.comment_set.all()` 를 할 때에는

`인스턴스이름 . foreignkey가지고 있는 모델이름(소문자)_set.all()` 로 역참조를 하면 된다.

`movie.comment_set.all()` 에서

- movie가 의미 하는 것은 detail 함수에서 만든 Movie 클래스의 객체 movie를 의미한다.
- comment_set에서 comment가 의미하는 것은 models.py 에 movie를 foreignkey 필드를 가진 클래스 이름을 소문자로 작성한 것이다.
- comment_set은 Django에서 제공하는 매니저 이름으로 1:N (일대다) 또는 N:M (다대다) 관계형

DB에서 역참조 할 때 사용하는 Django매니저 이름이다.

- .all() 이라는 QuerySet Api를 이용해서 모든 데이터를 조회하는 것이다.

> [중요]
>
> 나중에 헷갈리지 않도록 중간 정리를 하고 넘어가자.
>
> 1. 게시글이 있는데 해당 게시글에 달린 모든 댓글을 지칭할 때에는?
>
> movie.comment_set.all() 이 될 것이며 (역참조)
>
> ---
>
> 2. 어딘가에 댓글이 하나 달려 있는데
>
> 해당 댓글이 어떤 게시글을 참조하는지 보려면?
>
> comment.movie 라고 작성하면 되겠다.
>
> 그 이유는 우리는 models.py에서 comment 클래스를 정의할 때,
>
> movie라는 속성으로 외래키(FK)를 이미 지정해 놓았기 때문이다.

다음, `detail.html` 을 생성하자.
작성 할 코드는 다음과 같다.

```
# movies/detail.html

{% extends 'base.html' %}

{% block content %}
  <h1>DETAIL</h1>
  <hr />
  <div>
    <h5>{{ movie.title }}</h5>
    <p>{{ movie.description }}</p>

    {% comment %}게시글 업데이트 링크 및 삭제 버튼 구현{% endcomment %}

    {% 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 />

  {% comment %}해당 게시글의 댓글 Read 그리고 Delete 파트 구현{% endcomment %}
  <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
            {% csrf_token %}
            <input type="submit" value="DELETE" />
          </form>
        {% endif %}
      </li>
    {% endfor %}
  </ul>
  <hr />

  {% comment %}댓글 Create 파트 구현{% endcomment %}
  <form action="{% url 'movies:comments_create' movie.pk %}" method="POST">
    {% csrf_token %}
    {{ comment_form }}
    <input type="submit" />
  </form>
{% endblock %}
```

아래 추가 설명을 보기 전에 먼저 위의 코드를 살펴보고 스스로 해석을 먼저 해보자.

- `if user == movie.user` 부분을 통해, 해당 영화를 작성한 사용자만 게시글 `update` 와 `delete` 를 할 수 있도록 했다.
- `update` 와 `delete` 두 기능 모두 `movie.pk` 를 넘겨줘서 어떤 게시글을 수정 또는 삭제 할 것인지 알려 준다는 것을 명심하자.

- FOR 문을 사용해서 영화에 대한 댓글들을 출력해서 사용자에게 보여 준 다음에
- `{% if user == comment.user %}` 를 통해서 댓글을 작성한 사용자만 해당 댓글을 지울 수 있도록 처리 했다.
- 그리고 마지막으로 유저가 직접 댓글을 작성할 수 있는 {{comment_form}} 도 작성해 주었다.

서버를 켜서 확인해 보자.

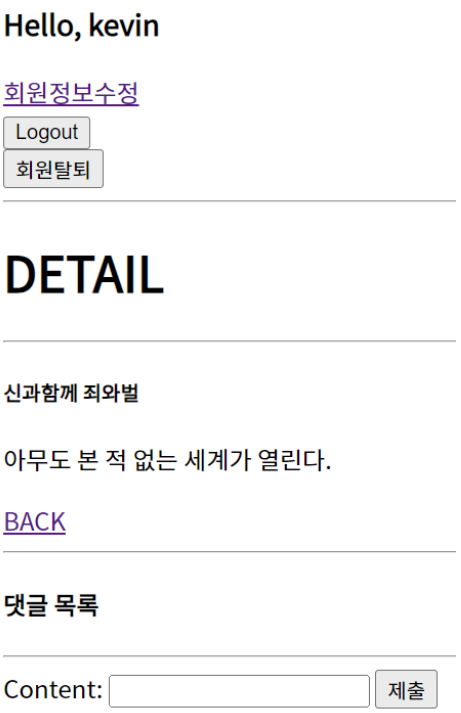
http://127.0.0.1:8000/movies/ 리스트에 아무 영화나 클릭 해 보자.

아직 update와 delete 기능을 구현하지 않았기 때문에 클릭을 해도 아무것도 나오지 않을 것이다.
그렇지만 해당 게시글의 작성자인지 아닌지에 따라 보여지는 페이지가 다른지
확인을 해볼 필요는 있겠다.

> 내가 생성한 영화 게시글 이라면, UPDATE 와 DELETE 가 보이도록 만들었다.
>
> 내가 생성한 영화가 아니라면 UPDATE 와 DELETE 가 보이지 않아야 할 것이다.

kevin이라는 유저가 작성한 명량 게시글을 확인해 보면 다음과 같을 것이다.
update, delete 버튼이 잘 보일 것이다.

하지만 kevin이라는 유저가 작성하지 않은 게시글을 보면 아래와 같을 것이다.
update, delete 버튼은 찾아보기 힘들 것이다.



내가 생성한 영화 게시글 이라면, UPDATE 와 DELETE 가 보이도록 만들었다.
내가 생성한 영화가 아니라면 UPDATE 와 DELETE 가 보이지 않아야 할 것이다.

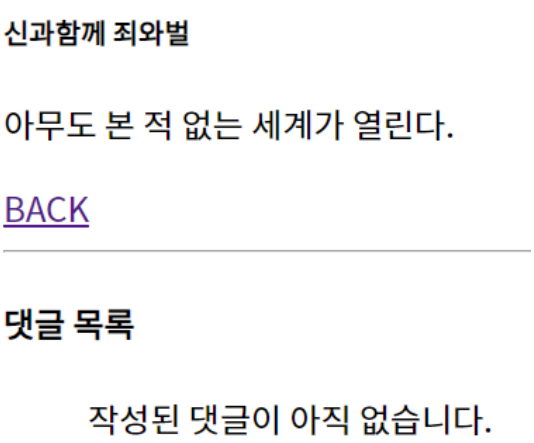
여기서 댓글이 없는 경우 “작성된 댓글이 아직 없습니다.” 가 출력되도록 코드를 조금만 업그레이드 해보자.
댓글이 없는 경우 “작성된 댓글이 아직 없습니다.” 가 출력이 되게 할 것이다.

```
# movies/detail.html

<ul>
 {% for comment in comments %}
  <li>
   {{ comment.content }}

   {% if user == comment.user %}
    <form action="{% url 'movies:comments_delete' movie.pk comment.pk %}" method="POST
     {% csrf_token %}
     <input type="submit" value="DELETE" />
    </form>
   {% endif %}
  </li>
 {% empty %}
  <p>작성된 댓글이 아직 없습니다.</p>
 {% endfor %}
```

```
  </ul>
  <hr />
```



마지막으로, `movies/views.py` 의 `create` 함수의 `redirect` 부분을
`index` 페이지가 아니라 방금 생성한 영화의 디테일페이지로 보내도록 바꿔보자.

redirect('movies:index') 부분을 redirect('movies:detail', movie.pk) 으로 수정하였다.

```
# 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, 인덱스페이지가 아니라, 해당 영화의 디테일페이지로 가는지
서버를 켜서 확인 해보자.

지금까지 우리가 무엇을 구현 했는지 중간점검을 하자.

1. 가장 먼저 index 페이지를 만들고
2. 게시글 생성을 (Create) 구현했고
3. 게시글 Detail 페이지도 구현했다.

지금부터 할 것은 게시글 삭제 이다.
`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')
```

- movie = Movie.objects.get(pk=pk) 삭제를 하고자 하는 영화를 가져온 후
- if request.user.is_authenticated: 로그인이 되어 있을 때 만 동작되도록 했다.
- if request.user == movie.user: 삭제 할 게시글의 작성자가 현재 로그인한 유저인지 확인

서버를 켜서 아무 게시글의 상세 페이지로 들어간 후, 삭제가 잘 되는지 테스트해보자.

지금까지 우리가 무엇을 구현 했는지 중간점검을 하자.

1. 가장 먼저 index 페이지를 만들고
2. 게시글 생성을 (Create) 구현했고
3. 게시글 Detail 페이지도 구현했다.
4. 방금 게시글 삭제를 (Delete) 구현했고

지금부터 할 것은 영화 내용 수정이다.

```
# movies/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 업데이트 하려고 하는지 알아야 하기 때문에, url 파라미터로 movie.pk 를 받았다.

또한, 수정하려는 사람은 해당 게시글을 작성한 사용자 이여야만 하기에

`if requres.user == movie.user`로 판단을 한다.

만약, 영화를 등록한 사용자가 아니라면 인덱스 페이지로 리다이렉트 될 것이다.

여기서도 GET 인지 POST 인지에 따라 `update` 함수의 역할이 달라지는데

- GET 일 경우 단순히 작성 폼을 보여주기만 해야 되지만, 중요한 것은 폼에 수정 전 내용을 보여줘야 함으로 `instance=movie` 로 하였다. 즉 사용자가 선택한 단일 영화로 받아 폼을 만들고, `context` 를 구성해 `update.html` 로 넘겨주었다.

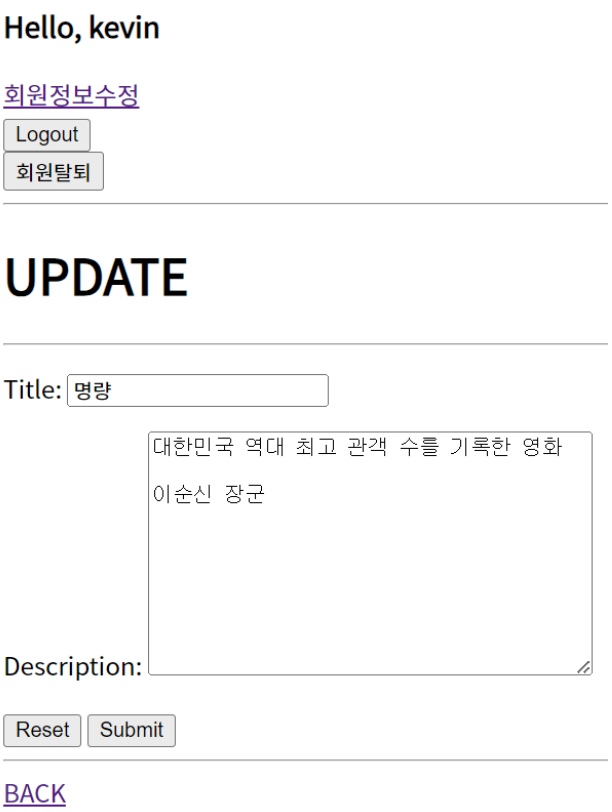
- POST 즉, 사용자가 폼에 기록을 한 내용을 받는다면, 해당 영화에 작성 되어야 함으로 역시 `instance=movie` 를 파라미터로 줘야 하고, 유효한지 체크를 한 다음에 폼을 저장한 후, 수정한 해당 영화의 디테일 페이지로 넘어간다. 만약, 유효성 체크를 통과하지 못한다면, 단순히 그 영화에 대한 폼을 채운 다음 다시 업데이트 페이지로 넘긴다.

`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 작동 하는지 테스트해보자.



자 이제 거의 다 끝났다.

게시글 관련된 것은 모두 끝났다. 이제 남은 것은 댓글 생성과 삭제 이다.

댓글 생성 먼저 보자. 게시글의 디테일페이지에서 댓글 생성 기능을 추가 할 것이다.

먼저 `views.py` 에서 `comment_create` 함수를 작성해 보자.

```
# 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')
```

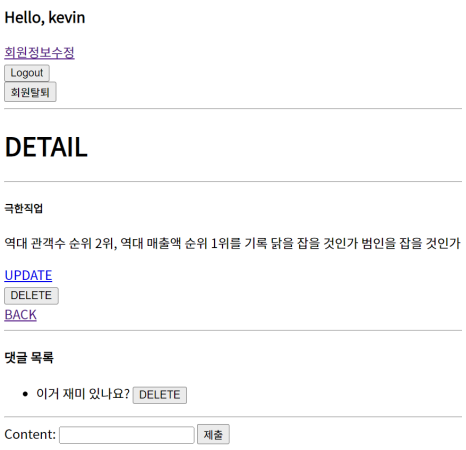
- 우선, `urls.py` 에서 URL 경로를 확인을 해보면, 매개변수 `pk` 는 댓글을 작성할 “게시글”의 PK라는 것을 알 수 있을 것이다.

- 댓글은 로그인한 사람만 작성할 수 있으며, 로그인하지 않았을 경우, 로그인 창으로 리다이렉트 된다.
- comment 작성 시 `commit=False` 로, Database에 form을 저장하기 전에 필요한 정보를 추가할 것이다.

  댓글을 작성하는 대상이 게시글인지, 그리고 어떤 유저가 댓글을 작성 하는지에 대한 정보는 `CommentForm` 에 빠져있기 때문이다. 이는 `forms.py` 에 가면 확실히 알 수 있는데, 해당 댓글을 작성한 유저가 아닌 일반 유저가 해당 댓글을 조작하지 못하도록 `exclude` 처리했기 때문이다.

  comment객체에 작성할 댓글의 주체가 되는 게시글과 작성자 정보까지 모두 comment 객체에 포함 시킨 후
- comment 객체를 저장하고, 디테일 페이지로 보낸다.

댓글을 몇 개 작성해 보자. 그리고 다른 아이디로 로그인 후에 댓글을 지울 수 있는지 확인해 보자.



"이거 재미 있나요?" 라는 댓글을 살펴 봐라.

해당 댓글을 작성하지 않은 사용자라면, 댓글을 지울 권한이 없다.

그리고 로그인을 하지 않은 사람이 댓글을 작성 하려고 한다면 제출 버튼을 누르자 마자

로그인 페이지로 이동할 것이다.

여기서 몇가지 더 업그레이드 해보자.

1. 현재 게시글에 댓글이 몇개 존재하는지도 표기해 보자.
2. 그리고 댓글 작성자가 누구인지도 표기해 보겠다.

```
# movies/detail.html


  <h4>댓글 목록</h4>

  {% if comments %}
    <p>총 {{ comments|length }}개 댓글</p>
    <ul>
      {% for comment in comments %}
```

```
      <li>
        {{ comment.user }} : {{ comment.content }}

        {% if user == comment.user %}
          <form action="{% url 'movies:comments_delete' movie.pk comment.pk %}" method="POS
            {% csrf_token %}
            <input type="submit" value="DELETE" />
          </form>
        {% endif %}
      </li>
    {% endfor %}
  </ul>
{% else %}
  <p>작성된 댓글이 아직 없습니다.</p>
{% endif %}

<hr />
```

IF 문을 통해서 만약에 댓글이 있을 경우 length filter를 이용해서 댓글의 개수를 출력하고

댓글이 없을 경우 else구문이 실행되어 “작성된 댓글이 아직 없습니다.” 가 출력 되도록 수정했다.

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 개를 넘겨 받았다.

`urls.py` 에서 경로를 주의깊게 살펴보자. 영화의 PK 도 필요하고, 댓글의 PK 도 필요함을 알 수 있다.

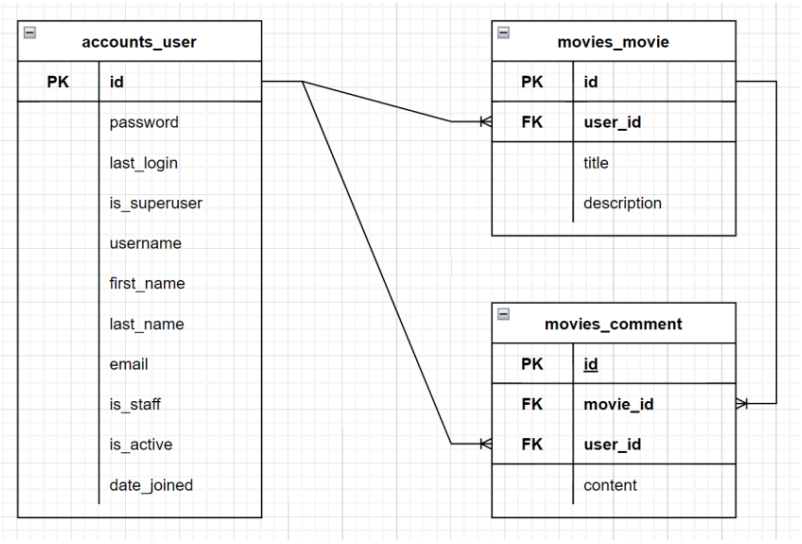
즉, 어떤 영화의 댓글을 삭제할 것인지, 그리고 댓글 중에서 어떤 댓글을 지울 것인지 파악하기 위해서

영화의 PK 그리고 댓글의 PK 모두 알아야 한다. 댓글을 삭제 한 후에는 디테일 페이지로 리다이렉트 하도록 하였다.

서버켜서 잘 지워지는지 확인해 보자.

자, 진짜 마지막으로, ERD 에 대해서 살펴보자.

ERD는



이거다.

ERD는 “Entity-Relationship Diagram” 을 줄인 말로

데이터베이스의 구조를 시각적으로 표현한 다이아그램이다.

ERD는

Entity / Attribute / Relation으로 구성이 되는데

- Entity - 여기서 회원정보(User) 게시글(Movie) 댓글(Comment) 들을 의미한다.
  즉, 엔티티는 DB에 저장될 데이터의 유형을 의미한다.
- Attribute - 회원이름 비밀번호는 User의 속성이 되는 것이고
  게시글의 ID, 제목 그리고 내용등은 게시글의 속성이라고 할 수 있겠다.
- Relation - 엔티티 간의 연관성을 의미한다. 여기서는 1:N (일대다) 관계가 존재한다.
  1:N 또는 N:M 등 엔티티간의 관계를 데이터베이스 에서는 Cardinality 라고 표현 하기도 한다.
  간단하다. Relation은 단순히 엔티티간의 연결여부를 의미하고
  Cardinality 연결의 상태, 수량적 특성(1:N,N:M)을 의미한다고 이해하면 된디.

- ERD 제작 사이트는 여러가지가 있다 마음에 드는 것을 이용해서 직접 ERD를 만들어 보자.
  - https://www.lucidchart.com/pages/?
  - https://app.diagrams.net/
  - https://www.erdcloud.com/
  - https://creately.com/

진짜 끝! 화이팅 !

django_test (1).zip