8기 파이썬반 PJT - 8 (3)

Created	@2022년 11월 2일 오전 11:28
Last Edited Time	@2022년 11월 2일 오후 4:41
Type	
Status	
Created By	
Last Edited By	
Stakeholders	

3. movies

먼저, `moives` 앱의 URL 은 다음과 같이 제시되었다.

HTTP verb	URL 패턴	설명
GET	/	전체 영화 목록 조회 페이지
GET	<movie_pk>/	단일 영화 상세 조회 페이지
GET	recommended/	추천 영화 조회 페이지

이를 기반으로 `movies/urls.py` 를 작성해보면 다음과 같다.

```
from django.urls import path
from . import views

app_name = 'movies'

urlpatterns = [
    path('', views.index, name='index'),
    path('<int:movie_pk>/', views.detail, name='detail'),
    path('recommended/', views.recommended, name='recommended'),
]
```

`views.py` 도 한번에 작성해보자.

```
from django.shortcuts import get_object_or_404, render
from django.views.decorators.http import require_safe
from .models import Movie, Genre


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

```
@require_safe
def detail(request, movie_pk):
    movie = get_object_or_404(Movie, pk=movie_pk)
    context = {
        'movie': movie,
    }
    return render(request, 'movies/detail.html', context)


@require_safe
def recommended(request):
    pass
```

일단, 추천 기능인 `recommended()` 는 작성하지 않았다. 학생 개개인이 알고리즘을 설계해 어떤 영화를 추천할지 결정하면 될 것이다.

우선, `index()` 즉 `/movies` 로 접속할 경우, 모든 영화 정보를 가져와서 `movies/index.html` 로 넘겨주는 역할을 한다.

그리고 개별 영화에 대한 정보는 마찬가지 방식으로 `detail()` 에서 처리하는 것을 볼 수 있다.

먼저 `movies/index.html` 는 다음과 같다.

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

{% block content %}
  <h1>Movies</h1>
  {% for movie in movies %}
    <h3>{{ movie.title }}</h3>
    <p>
      {% for genre in movie.genres.all %}
        <span>{{ genre.name }}</span>
      {% endfor %}
    </p>
    {% if movie.overview %}
      <p>{{ movie.overview|truncatechars:60 }}</p>
    {% else %}
      <p>줄거리 없음</p>
    {% endif %}
    <a href="{% url 'movies:detail' movie.pk %}">[detail]</a>
    <hr>
  {% endfor %}

{% endblock %}
```

여기선, `|truncatechars` 이외엔 어려울 게 없다. 문자열 길이가 60 이상이 되면, `...` 으로 보여줄 것이다.

그리고, `movies/etail.html` 은 다음과 같다. 어려운 구문은 없다.

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

{% block content %}
  <h1>{{ movie.title }}</h1>
  <p>
    {% for genre in movie.genres.all %}
    <span>genre : {{ genre.name }}</span>
```

```
    {% endfor %}
  </p>
  <p>popularity : {{ movie.popularity }}</p>
  <p>vote average : {{ movie.vote_average }}</p>
  <p>overview : {{ movie.overview }}</p>
  <img src="{{ movie.poster_path }}" alt="">

{% endblock %}
```

테스트해보자.

Hello, sylvie123

내 프로필

Logout

Movies

가브리엘의 지옥 파트 2

로맨스

줄거리 없음

[detail]

가브리엘의 지옥

로맨스

줄거리 없음

[detail]

Dedicada a mi ex

드라마 코미디

줄거리 없음

[detail]

각각의 디테일을 클릭하면?

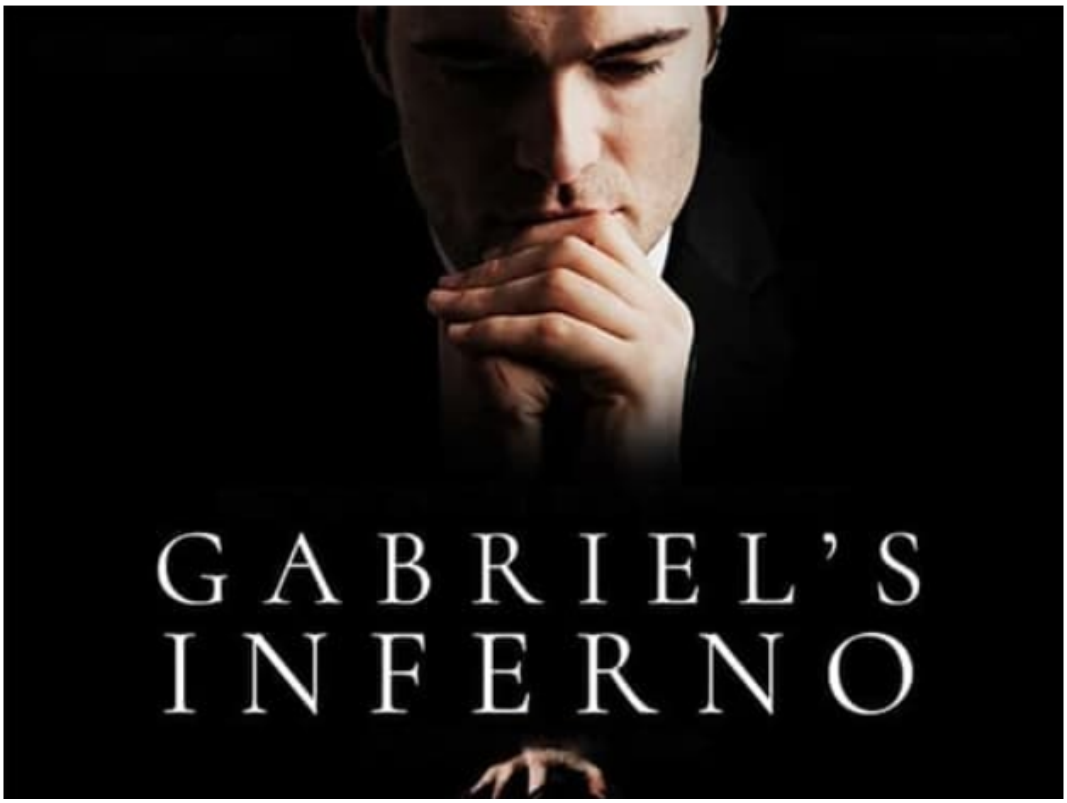
가브리엘의 지옥

genre : 로맨스

popularity : 12.701

vote average : 8.9

overview :



영화 이미지까지 볼 수 있는 멋있는 웹앱이 되었다.

4. community

`community` 앱의 URL 은 다음과 같다.

HTTP verb	URL 패턴	설명
GET & POST	create/	Form 표시 및 신규 리뷰 생성
GET	/	전체 리뷰 목록 조회 페이지
GET	<review_pk>	단일 리뷰 조회 상세 페이지

POST	<review_pk>/comments/create/	댓글 생
POST	<review_pk>/like/	좋아요 기

이를 기반으로 만들어본 `community/urls.py` 는 다음과 같다.

```
from django.urls import path
from . import views

app_name = 'community'

urlpatterns = [
    path('', views.index, name='index'),
    path('create/', views.create, name='create'),
    path('<int:review_pk>/', views.detail, name='detail'),
    path('<int:review_pk>/comments/create/', views.create_comment, name='create_comment'),
    path('<int:review_pk>/like/', views.like, name='like'),
]
```

`views.py` 의 각각의 함수들을 미리 작성해두자.

```
from django.shortcuts import render
from django.views.decorators.http import require_GET, require_POST, require_http_methods

@require_GET
def index(request):
    return render(request, 'community/index.html')


@require_http_methods(['GET', 'POST'])
def create(request):
    pass


@require_GET
def detail(request, review_pk):
    pass


@require_POST
def create_comment(request, review_pk):
    pass


@require_POST
def like(request, review_pk):
    pass
```

각각의 HTTP Methods 와, 파라미터까지 모두 정의를 해두었다.

이 중, 먼저, `index()` 를 먼저 작성해보자.

```
from .models import Review

@require_GET
def index(request):
    reviews = Review.objects.order_by('-pk')
```

```
    context = {
        'reviews': reviews,
    }
    return render(request, 'community/index.html', context)
```

리뷰는 최신순부터 출력되는것이 가장 좋으므로, 역순으로 가져와서 `index.html` 로 넘긴다.

다음, `community/index.html` 을 작성해보면 다음과 같다.

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

{% block content %}
  <h1>Community</h1>
  <hr>
  {% for review in reviews %}
    <p>작성자 : <a href="{% url 'accounts:profile' review.user.username %}">{{ review.user }}</a></p>
    <p>글 번호: {{ review.pk }}</p>
    <p>글 제목: {{ review.title }}</p>
    <p>글 내용: {{ review.content }}</p>
    <a href="{% url 'community:detail' review.pk %}">[detail]</a>
    <hr>
  {% endfor %}
{% endblock %}
```

눈여겨볼만한점은 작성자부분이다. 작성자는 링크로 설정되었고, 해당 작성자 이름을 클릭할 경우, 작성자의 프로필페이지로 넘어갈 수 있다. 즉, 팔로우/언팔로우 를 할 수 있는 페이지로 이동할 수 있다.

`admin` 에 관리자로 로그인한 다음, `review` 더미 데이터를 몇 개 생성한 후 테스트해보자.

Community

작성자 : sylvie123

글 번호: 3

글 제목: 무엇을 상상하든 그 이상...

글 내용: 내가 본 인생 띵작.... 진짜 이런 명작이 왜 이제 나왔는지...

[detail]

작성자 : nana123

글 번호: 2

글 제목: 해리포터의 전설을 잇는...

글 내용: 끝났는줄알았던 해리포터시리즈를 다시 볼 수 있다니! 감동이 밀려옴

[detail]

작성자 : jony123

글 번호: 1

글 제목: 정말 재밌는 영화

이제 영화 상세 정보를 구현하기 위해 `community/views.py` 의 `detail()` 을 작성해보자.

```
from django.shortcuts import render, get_object_or_404

@require_GET
def detail(request, review_pk):
    review = get_object_or_404(Review, pk=review_pk)
    comments = review.comment_set.all()
    context = {
        'review': review,
        'comments': comments,
    }
    return render(request, 'community/detail.html', context)
```

~~크게 어려운 건 없는 코드다. 해당 리뷰와, 댓글 목록을 받아 `community/detail.html` 로 넘겨준다.~~

`community/detail.html` 은 다음과 같다.

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

{% block content %}
  <h2 class="text-center">DETAIL</h2>
  <h3>{{ review.pk }} 번째 글</h3>
  <hr>
  <p>제목: {{ review.title }}</p>
  <p>영화 제목: {{ review.movie_title }}</p>
  <p>내용: {{ review.content }}</p>
  <p>평점: {{ review.rank }}</p>
  <p>작성 시각: {{ review.created_at }}</p>
  <p>수정 시각: {{ review.updated_at }}</p>

  <hr>
  <h4>댓글 목록</h4>
  {% if comments|length %}
    <p><b>{{ comments|length }}개의 댓글이 있습니다.</b></p>
  {% endif %}
  {% for comment in comments %}
    <div>
      {{ comment.user }} - {{ comment.content }}
    </div>
  {% empty %}
    <p><b>댓글이 없어요..</b></p>
  {% endfor %}
  <hr>
  <a href="{% url 'community:index' %}">[back]</a>
{% endblock  %}
```

`admin` 에 관리자로 접속해, 댓글 몇 개를 생성한 다음 결과를 테스트해보자.

댓글이 없을 경우는 다음과 같다.

DETAIL

3 번째 글

제목: 무엇을 상상하든 그 이상...

영화 제목: 인셉션

내용: 내가 본 인생 띵작.... 진짜 이런 명작이 왜 이제 나왔는지...

평점: 5

작성 시각: 2022년 11월 2일 3:05 오후

수정 시각: 2022년 11월 2일 3:05 오후

댓글 목록

댓글이 없어요..

[back]

댓글이 있을 경우는 다음과 같다.

댓글 목록

2개의 댓글이 있습니다.

nana123 - ㅋㅋㅋㅋㅋㅋㅋㅋ꿀잼?
sylvie123 - 진짜 재밌나봄...

[back]

이제, 댓글 생성 기능을 만들어야한다. 먼저 `forms.py` 부터 만들자.

```
from django import forms
from .models import Review, Comment


class ReviewForm(forms.ModelForm):

    class Meta:
        model = Review
        fields = ['title', 'movie_title', 'rank', 'content']


class CommentForm(forms.ModelForm):

    class Meta:
        model = Comment
        exclude = ['review', 'user']
```

`ReviewForm` 의 경우, 사용자가 필요한 `field` 는 총 네 가지, "글 제목", "영화 이름", "별점", "리뷰 내용" 이 네 가지만 작성하면 된다.

`CommentForm` 의 경우, "리뷰" 와 "유저" 는 제외 즉 `exclude` 해야 하는데, 해당 디테일에 접속했을 땐 어떤 리뷰에 해당하는 댓글인지 이미 `pk` 를 가지고 있을 것이고, 로그인한 상태이기 때문에 "유저" 역시도 `form` 에 보여주면 안된다.

이걸, 우선 `community/views.py` 로 가져온다.

```
 from .forms import ReviewForm, CommentForm
```

우선, 댓글 쪽을 만들다가 말았으니, `detail()` 에 `ReviewForm` 을 `context` 에 추가하자.

```
@require_GET
def detail(request, review_pk):
    review = get_object_or_404(Review, pk=review_pk)
    comments = review.comment_set.all()
    comment_form = CommentForm()
    context = {
        'review': review,
        'comment_form': comment_form,
        'comments': comments,
    }
    return render(request, 'community/detail.html', context)
```

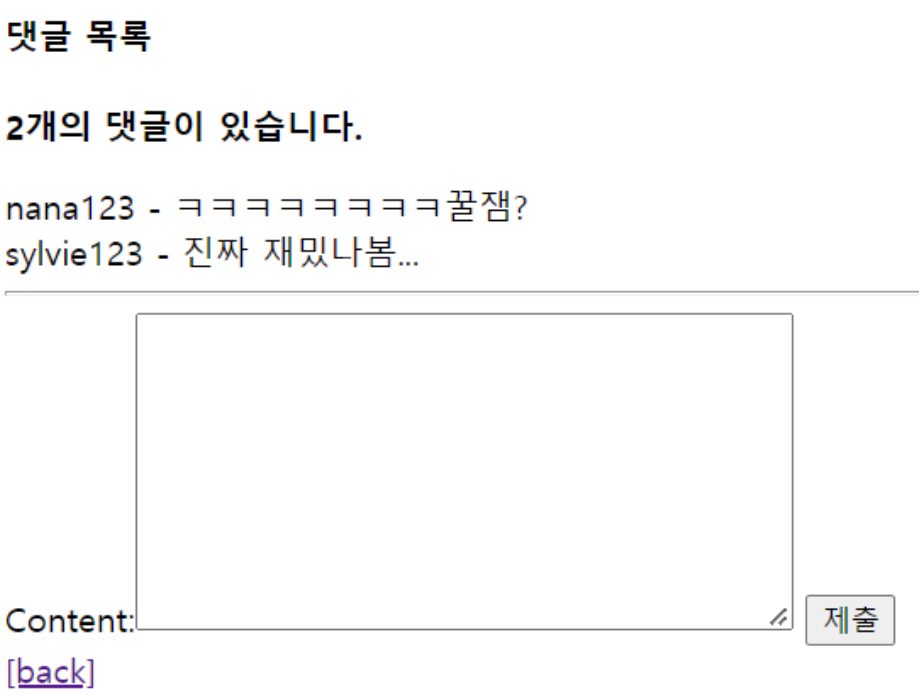
그리고, `community/detail.html` 에서 댓글 작성할 수 있는 `comment_form` 을 출력해보자.

```
  ...
  ...
  ...
  {% if request.user.is_authenticated %}
    <form action="{% url 'community:create_comment' review.pk %}" method="POST">
      {% csrf_token %}
      {{ comment_form }}
      <input type="submit">
    </form>
```

```
  {% else %}
    <a href="{% url 'accounts:login' %}">[댓글을 작성하려면 로그인하세요.]</a>
  {% endif %}
  <a href="{% url 'community:index' %}">[back]</a>
{% endblock  %}
```

로그인한 사용자만 댓글 작성이 허용되며, 댓글 작성을 시도할 시, `community:create_comment` URL 으로, `review.pk` 파라미터와, 사용자가 작성한 데이터가전송된다.

만약 로그인하지 않았다면 로그인 안내 링크를 보여준다.



다음과 같다. 이제 `community/views.py` 의 `create_comment()` 를 작성해보자.

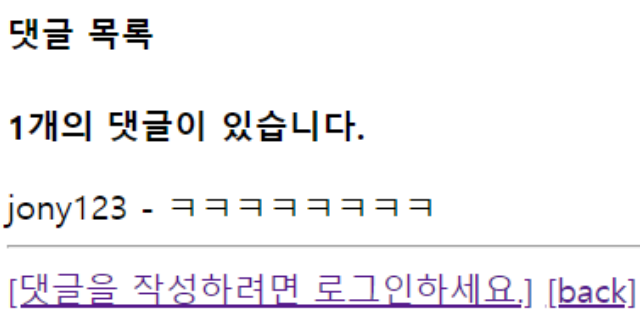
```
from django.shortcuts import render, redirect, get_object_or_404

@require_POST
def create_comment(request, review_pk):
    review = get_object_or_404(Review, pk=review_pk)
    comment_form = CommentForm(request.POST)
    if comment_form.is_valid():
        comment = comment_form.save(commit=False)
        comment.review = review
        comment.user = request.user
        comment.save()
        return redirect('community:detail', review.pk)
    context = {
        'comment_form': comment_form,
        'review': review,
        'comments': review.comment_set.all(),
    }
    return render(request, 'community/detail.html', context)
```

해당 리뷰를 `get_object_or_404()` 로 가져오고, 사용자가 입력한 내용을 `CommentForm(request.POST)` 를 사용해 넣어주어 유효성검사를 한다.

만약, 유효하다면 해당 댓글을 DB 에 저장하고, `detail.html` 로 리다이렉트해준다.

그렇지 않다면, 사용자가 입력한 정보는 그대로 유지시킨 채 `detail.html` 을 다시 렌더링한다.



로그인하지 않은 상태라면 위와 같은 화면이 보인다.

로그인한 상태에서 댓글 작성을 테스트해보자.

다음은 리뷰 작성이다. `community/views.py` 의 `create()` 함수를 작성해보자,

```
@require_http_methods(['GET', 'POST'])
def create(request):
    if request.method == 'POST':
        form = ReviewForm(request.POST)
        if form.is_valid():
            review = form.save(commit=False)
            review.user = request.user
            review.save()
            return redirect('community:detail', review.pk)
    else:
        form = ReviewForm()
    context = {
        'form': form,
    }
    return render(request, 'community/create.html', context)
```

`POST` 에선 사용자 데이터를 넘겨받아 실제로 리뷰 등록이 이루어진다.

`GET` 에선 사용자가 리뷰를 작성할 수 있는 `ReviewForm` 을 넘겨준다.

작성된 `ReviewForm` 의 유효성검사 실패 시, 데이터를 그대로 유지한 채 `create.html` 로 넘겨준다.

이를 위한 `create.html` 은 다음과 같다.

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

{% block content %}
  <h1 class="text-center">CREATE</h1>
  <form action="" method="POST">
    {% csrf_token %}
    {{ form.as_p }}
```

```
    <input type="submit">
  </form>
  <hr>
  <a href="{% url 'community:index' %}">[back]</a>
{% endblock %}
```

간단한 코드이기 때문에 설명은 생략한다.

이제 `base.html` 에, community, movie, 새 리뷰 생성으로 향하는 링크를 만들어주자.

```
    </form>
        <a href="{% url 'movies:index'%}">Movie</a>
        <a href="{% url 'community:index'%}">Community</a>
        <a href="{% url 'community:create'%}">New Review</a>
      {% else %}
        <a href="{% url 'accounts:login' %}">Login</a>
        <a href="{% url 'accounts:signup' %}">Signup</a>
      {% endif %}
    </nav>
```

이후, 리뷰 생성이 잘 되는지 적당한 유저로 로그인한 후 테스트해보자.

다음과 같이 작성하고 제출 버튼을 누르면,

DETAIL

4 번째 글

제목: 내 생애 이렇게 멋진 영화는 처음이야

영화 제목: 라라랜드

내용: 정말 환상적인 영화! 뮤지컬을 영화로 본 건 처음이지만 너무 감동적으로 봤다.

평점: 5

작성 시각: 2022년 11월 2일 4:01 오후

수정 시각: 2022년 11월 2일 4:01 오후

댓글 목록

댓글이 없어요..

해당 리뷰의 상세 페이지로 이동하면서 잘 작성된 것을 볼 수 있다.

마지막으로, 구현해 볼 기능이 있다. `community/index.html` 에서, 각 영화에 좋아요를 클릭할 수 있는 기능을 구현할 것인데, 비동기통신을 사용해서 부드러운 사용자 경험을 제공할 것이다.

일단, `community/views.py` 의 `like()` 함수를 작성해보자.

```
from django.http import JsonResponse, HttpResponse

@require_POST
def like(request, review_pk):
    if request.user.is_authenticated:
        review = get_object_or_404(Review, pk=review_pk)
        user = request.user
        if review.like_users.filter(pk=user.pk).exists():
            review.like_users.remove(user)
            liked = False
        else:
            review.like_users.add(user)
            liked = True
        context = {
            'liked': liked,
            'count': review.like_users.count()
        }
        return JsonResponse(context)
    return HttpResponse(status=401)
```

일단, 어떤 리뷰에 들어왔는지 알아야하기 때문에 `review_pk` 를 URL 파라미터로 받았다.

로그인한 사용자가 아니라면, HTTP Status 401, 인증되지 않은 사용자라는 상태를 보낸다.

로그인했다면, 리뷰를 가져와 `review` 에 담고, 현재 로그인한 유저를 `user` 에 담는다.

만약, 이미 좋아요를 눌렀다면 좋아요를 취소하고,

좋아요를 누르지 않았다면 좋아요를 추가하며,

각각에 `liked` 플래그를 같이 설정한다.

그리고 `context` 에, `liked` 플래그와, 몇 명이 좋아요를 눌렀는지를 센 다음, JSON Format 으로 보낸다.

이제 `index.html` 을 수정한다.

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

{% block content %}
  <h1>Community</h1>
  <hr>
  {% for review in reviews %}
    <p>작성자 : <a href="{% url 'accounts:profile' review.user.username %}">{{ review.user }}</a></p>
    <p>글 번호: {{ review.pk }}</p>
    <p>글 제목: {{ review.title }}</p>
    <p>글 내용: {{ review.content }}</p>


    <form class="like-form" data-review-id="{{ review.pk }}">
      {% csrf_token %}
      {% if request.user in review.like_users.all %}
        <button id="like-{{ review.pk }}">좋아요 취소</button>
      {% else %}
        <button id="like-{{ review.pk }}">좋아요</button>
      {% endif %}
    </form>
    <p>
      <span id="like-count-{{ review.pk }}">{{ review.like_users.all|length }}</span> 명이 이 글을 좋아합니다.
    </p>


    <a href="{% url 'community:detail' review.pk %}">[detail]</a>
    <hr>
  {% endfor %}
{% endblock %}
```

추가된 부분은 두 줄씩 띄워놓았다.

`review.pk` 는 script 작성 시에 필요하다. `reviewId` 로 받을 예정이다.

만약 현재 유저가 해당 게시글에 좋아요를 이미 눌렀다면 “좋아요 취소” 버튼을,

현재 유저가 해당 게시글에 좋아요를 누른 적 없다면 “좋아요” 버튼을 보여준다.

그러나 이 경우, `{% for review in reviews %}` 를 사용해 여러 개 태그가 이미 존재하는 경우이다. 그렇기에, 접근이 필요한 각각의 태그에는 고유한 `id` 를 부여하는데, `review.pk` 를 사용한다.

그래서, `<form>` 태그에 `id` 가 아니라, `class` 지정을 한 것이다.

남은 건, 이걸 가능하게 하는 `{% block script %}` 를 작성하면 된다.

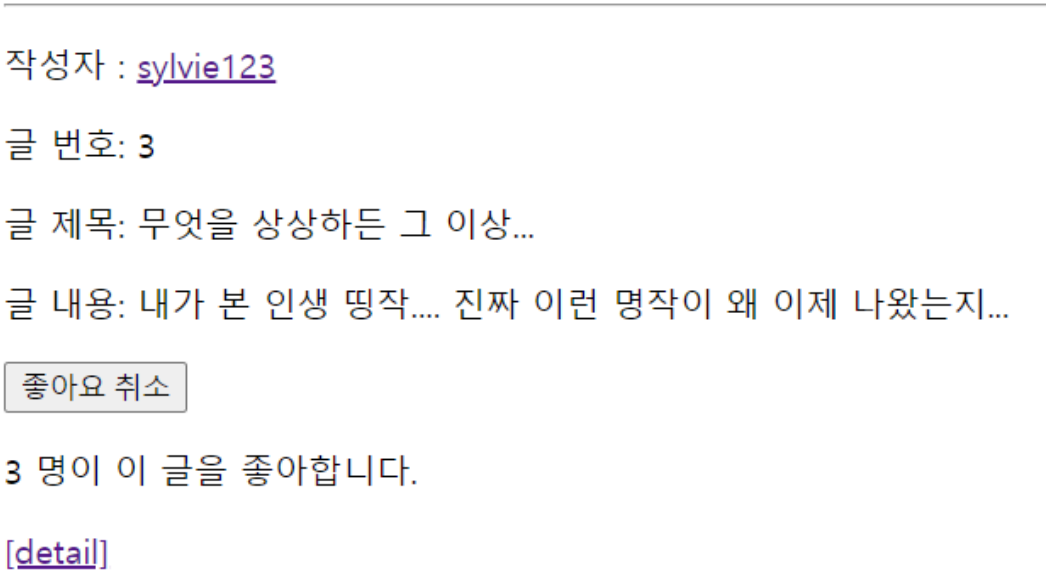
```
{% block script %}
  <script>
    const forms = document.querySelectorAll(".like-form");
    const csrftoken = document.querySelector("[name=csrfmiddlewaretoken]").value;
    forms.forEach((form) => {
      form.addEventListener("submit", function (event) {
        event.preventDefault();
        const reviewId = event.target.dataset.reviewId;
        axios({
          method: "post",
          url: `/community/${reviewId}/like/`,
          headers: { "X-CSRFToken": csrftoken },
        }).then((response) => {
          const liked = response.data.liked;
          const count = response.data.count;
          const likeButton = document.querySelector(`#like-${reviewId}`);
          const likeCount = document.querySelector(`#like-count-${reviewId}`);
          if (liked === true) {
            likeButton.innerText = "좋아요 취소";
          } else {
            likeButton.innerText = "좋아요";
          }
          likeCount.innerText = count;
        });
      });
    });
  </script>
{% endblock script %}
```

아까전과 똑같은 원리긴 한데, 여기선 클래스를 가져온다. 즉, 배열이 될 것이다.

그래서, `forEach` 를 사용해 배열인 `form` 을 순회하며 작업하는 것이다. 즉, 현재 DOM 에 존재하는 모든 버튼에 이벤트리스너를 부여한다고 보면 된다.

접근하고자하는 `좋아요/좋아요 취소` 버튼을 정확하게 알아야 하기 때문에, `querySelector(`#like-${reviewId}`)` 로, 백틱을 사용해 접근했고, `likeCount` 도 마찬가지다.

테스트해보면 다음과 같다.



해당 리뷰에 좋아요 버튼을 눌렀을 때, `좋아요` 에서 `좋아요 취소` 로 버튼이 바뀌었고, 2명에서 3명으로 좋아하는 사람의 숫자가 성공적으로 바뀌었음을 확인할 수 있다.

- 도전: 영화 추천 알고리즘 작성

```
# movies/views.py

@require_safe
def recommended(request):
    pass
```

영화 추천 알고리즘은 구현되지 않은 상태이다. 창의성을 발휘해서, 영화 추천 기능을 구현해보자.

```
영화 추천 기능
1. 사용자가 인증되어 있다면, 적절한 알고리즘을 활용하여 10개의 영화를 추천하여 제공합니다.
2. 영화를 추천하는 알고리즘은 자유롭게 구상합니다.
3. 구현한 알고리즘에 대해 README.md에 상세히 작성합니다.
```