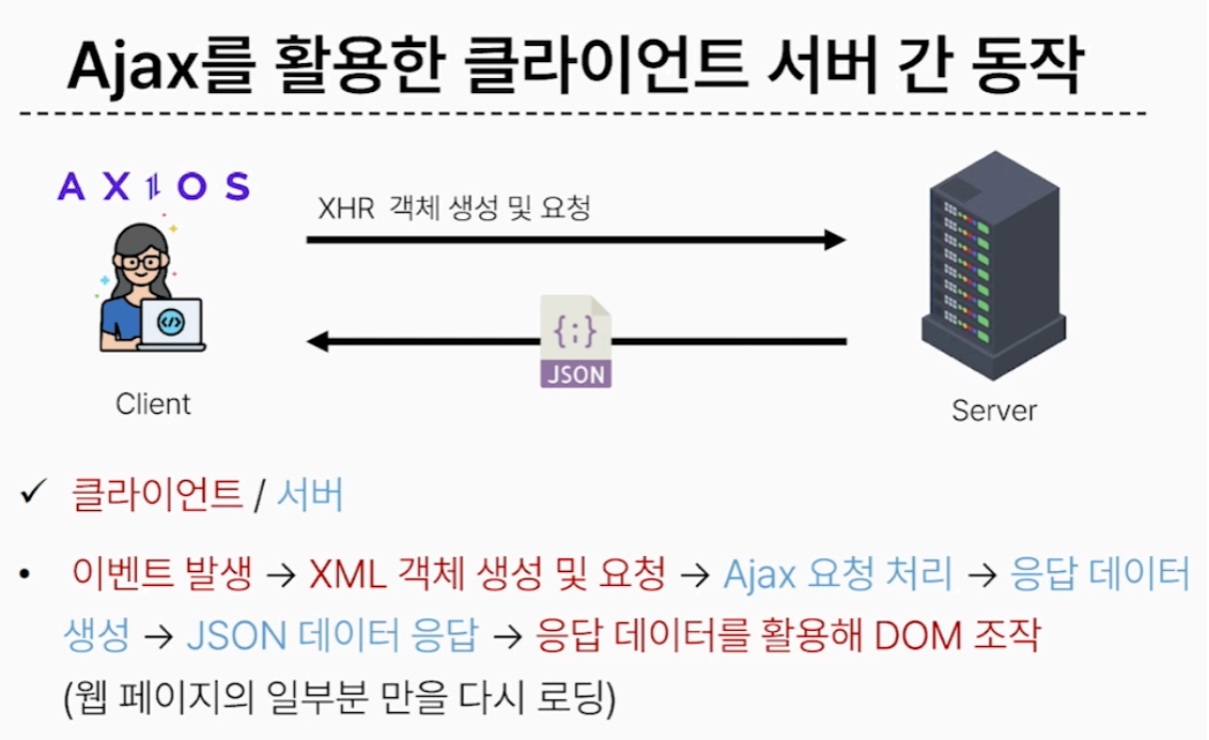
Ajax를 이용한 좋아요 팔로우 구현

자료	Javascrtip
구분	Javascript
과목	

follow_like_async(skeleton).zip

AJAX 를 활용 한 javascript의 dom 조작 과정



1. 이벤트 발생 (클릭) → 콜백함수 동작
2. 콜백함수는 서버에 요청을 보내는 axios 코드가 실행되면
3. 브라우저에서 axios가 XHR (Xml Http Request) 요청을 서버로 보낸다. 여기서 XHR 객체는 javascript가 서버에 데이터를 요청 보내는 양식을 의미한다.
4. 요청을 서버에서 처리하고 응답을 JSON 데이터 응답을 한다.
5. 응답받은 클라이언트는 응답받은 data를 활용해서 DOM을 조작한다.
6. .then 또는 catch를 이용해서 HTML문서를 조작하게 된다.

비동기 적용하기 - 팔로우(Follow)

axios 요청 준비

팔로우 기능을 비동기로 개선해보자.

먼저, `base.html` 을 다음과 같이 수정한다.

각각의 템플릿에 script 코드를 작성하기 위한 block tag 영역 작성해 준다.

body의 닫힘태그 바로 위에 작성해 준다.

```
<!-- base.html -->

<body>
...

    {% block script %}
    {% endblock script %}
</body>
</html>
```

맨 아래에, `script` 를 넣을 수 있는 영역을 추가했다.

다음, `accounts/profile.html` 을 아래와 같이 수정한다.

axios CDN 작성할 것이다.

profile.html 파일 안에 {% endblock %} 밑에 {% block script %} 을 넣고 CDN을 추가하자.

{구글에 axios 검색 후 공식 홈에서 CDN 복붙 해 주세요)

```
<!-- accounts/profile.html -->

{% block script %}
    <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

    <script>

    </script>
{% endblock script %}
```

우리는 프로필 페이지에서 팔로우 기능을 구현하고 있다.

여기서 우리는 팔로우 또는 언팔로우 상태(value)를

form태그의 action 그리고 mathod 속성을 이용해서

views.py에 follow함수로 값을 전달했다.

하지만 우리는 요청을 보낼 때 form 태그의

action 그리고 method 속성을 이용하는 것이 아니라 axios로 대체를 할 것이다.

- 먼저 form 태그를 수정해보자.

불필요해진 action과 method 속성은 삭제할 것이다. 우리는 axios로 대체할 것이기 때문이다.

```
<!-- accounts/profile.html -->

//<form action="{% url 'accounts:follow' person.pk %}" method="POST">  삭제!

<form id="follow-form">
  {% csrf_token %}
  {% if request.user in person.followers.all %}
    <input type="submit" value="Unfollow" />
  {% else %}
    <input type="submit" value="Follow" />
  {% endif %}
</form>
```

- 가장 먼저, axios가 동작 시키기 위해서 해당 form요소를 선택해 준다.

```
<!-- accounts/profile.html -->

{% block script %}
    <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

    <script>
      const form = document.querySelector("#follow-form")
    </script>
{% endblock script %}
```

- form 요소에 이벤트 핸들러 작성 및 submit 이벤트 취소를 하겠다.

`form` 태그의 기본 기능은 사용자가 폼을 제출(submit)하면 페이지를 새로고침하면서 데이터를 서버로 전송하는 것이다. 그러나 우리는 새로고침을 일으키지 않으려 한다. 따라서 submit 이벤트를 취소했다.

```
<!-- accounts/profile.html -->

<script>
    const form = document.querySelector("#follow-form")

    form.addEventListener('submit', function (event) {
```

```
    event.preventDefault() // form 태그의 기본 기능 막음
  })
</script>
```

form.addEventListener('submit', function (event) 의 의미는

1. submit 이벤트가 발생하면,
2. 서버로 follow 요청을 보내는 콜백함수가 실행 되도록 했다.

한번 더 말하자면,

form 태그의 submit 이 작동하면 브라우저에서는 새로고침이 작동 되는 것이 기본 디폴트값 이다. 따라서브라우저에서는 자동으로 작동하는 새로고침을 막기 위해 preventDefault() 함수를 추가함으로써

브라우저의 기본 동작(새로고침)을 막았다.

- axios 요청 준비

```
<script>
  const form = document.querySelector("#follow-form")

  form.addEventListener('submit', function (event) {
    event.preventDefault() // form 태그의 기본 기능 막음
    axios({
      method: "post",
      url: `/accounts/${???}/follow/`,
    })
  })

  </script>
```

그리고 URL을 적어 줘야 하는데 예전에 form에 적어 줬던 방식인,

<form action="{% url 'accounts:follow' person.pk %}" 의 형식은

Django Template Language 에서의 문자열 이다.

실제 우리가 요청을 보내는 URL을 확인 해 보면, (urls.py 를 통해 확인이 가능하다.)

url: / accounts / 팔로우할 상대방 pk / follow

이러한 형식이 될 것이다.

따라서 JS에서 사용하는 형식에 맞도록 javascript template literal (템플릿 문자열)로

바꿔줘야 한다.

(참고 : https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Template_literals)

url: `/accounts/${???}/follow/`,
이러한 모양이 될 것이다.

??? 에 들어갈 부분은
내가 팔로우 할 상대방의 pk가 될 것이다.

url에 작성할 user pk 가져오기

- url에 작성할 user pk 가져오기

  우리는 "HTML 데이터 속성"을 이용해서 -> JavaScript 에서 user pk 값을 가져올 것이다.

  데이터 속성이란?
  HTML에는 표준속성이라고 id class src 등등.. 이 있는데
  HTML에서 표준속성이 아닌, 개별적인 속성을
  HTML요소에 추가적으로 정보를 저장 할 수 있는 기능을 데이터 속성이라고 한다.
  이 데이터 속성은 JavaScript에서 HTML 표준속성 이 아닌 속성값들을 가져올 때 사용한다.

  사용 방법으로는 data-* 의 형식으로
  data- (데이터 그다음 하이픈을) 먼저 써주고 * 에는 아무 이름이나 쓸 수 있다.
  공식문서를 반드시 확인해 보자.
  https://developer.mozilla.org/ko/docs/Learn/HTML/Howto/Use_data_attributes
  공식문서에 들어가서 HTML 문법 그리고 Javascript에서 접근하기 부분에
  적혀있는 코드를 꼭 살펴보자.

```
<!-- accounts/profile.html -->

// user-id 개별속성을 form 태그에 추가
<form id="follow-form" data-user-id="{{ person.pk }}">
...
</form>
```

```
<!-- accounts/profile.html -->

<script>
  const form = document.querySelector('#follow-form')
```

```
    form.addEventListener('submit', function (event) {
        event.preventDefault()
        const userId = event.target.dataset.userId
        ...

    })
    </script>
```

따라서 우리는 url에 작성할 pk를 가져오기 위해서

HTML 데이터 속성을 이용하기 위해 `data-user-id` 라고 user-id 개별속성을 form 태그에

추가 해 주었고

Javascript 코드 (profile.html) 에서는 공식문서에 적혀 있는 문법에 의거하여

`.dataset.userId` 으로 접근 하였다.

`(user-id` 하이픈으로 써 준 것은 javascript에서는 Carmel case의 형태로 자동으로 바꿔준다.)

- url 작성 마치기

```
<!-- accounts/profile.html -->

<script>
    const form = document.querySelector("#follow-form")
    form.addEventListener('submit', function (event) {
      event.preventDefault() // form의 기본 기능 막음
      const userId = event.target.dataset.userId
      axios({
        method: "post",
              url: `/accounts/${userId}/follow/`,

      })
    })

    </script>
```

[참고] data-* attributes

- 사용자 지정 데이터 특성을 만들어 임의의 데이터를 HTML과 DOM사이에서 교환 할 수 있는 방법이다.

  사용 예시를 살펴보자.

```
<div data-my-id="my-data"></div>
<script>
    const myId = event.target.dataset.myId
</script>
```

- 모든 사용자 지정 데이터는 dataset 속성을 통해 사용할 수 있다.

  이는 돔 조작을 위헤서 html에서 제공해 주는 속성이다.
- HTML 요소에 data-my-id 라는 개별속성에 "my-data" 라는 값을 넣어 줬다.
- Javascript로 데이터 속성에 접근 하기 위해서는 아래와 같이 접근이 가능하다.

```
const myId = event.target.dataset.myId;
```

https://developer.mozilla.org/ko/docs/Web/HTML/Global_attributes/data-*

- 예를 들어 `data-test-value` 라는 이름의 특성을 지정했다면,

  JavaScript에서는 `element.dataset.testValue` 로 접근할 수 있다. 이때 속성명 작성 시 주의사항이 있다.

  속성명 작성 시 주의사항
  - 대소문자 여부에 상관없이 xml로 시작하면 안 됨
  - 세미콜론을 포함해서는 안됨
  - 대문자를 포함해서는 안됨

csrftoken 보내기

- 먼저 hidden 타입으로 숨겨져있는 csrf 값을 가진 input 태그를 선택해야 한다.

- AJAX POST 요청에 사용할 CSRF 토큰을 POST 데이터로 전달하는 방법은

  공식문서를 참고하자.

  아래 링크는 반드시 한번은 눌러서 확인하자.

  https://docs.djangoproject.com/en/4.2/howto/csrf/

- 코드는 아래와 같다.

```
<!-- accounts/profile.html -->

<script>
  const form = document.querySelector('#follow-form')
  form.addEventListener('submit', function(event) {
    event.preventDefault()
```

```
  const userId = event.target.dataset.userId
  const csrftoken = document.querySelector('[name=csrfmiddlewaretoken]').value

  axios({
    method: 'post',
    url: `/accounts/${userId}/follow/`
  })

 })
</script>
```

- Axios 로 csfrtoken 보내기

```
<!-- accounts/profile.html -->

<script>
 const form = document.querySelector('#follow-form')
 form.addEventListener('submit', function(event) {
  event.preventDefault()

  const userId = event.target.dataset.userId
  const csrftoken = document.querySelector('[name=csrfmiddlewaretoken]').value

  axios({
    method: 'post',
    url: `/accounts/${userId}/follow/`,
    headers: {'X-CSRFToken': csrftoken,}
  })

 })
</script>
```

팔로우 버튼 토글하기

views.py 로 넘어가서 follow 함수를 수정해 보자.
팔로우/언팔로우 상황이 바뀔 때 마다 새로운 HTML 페이지를 불러올 것이 아니라,
팔로우/언팔로우 상태가 바뀌면 그 결과를 JSON 형식으로 반환을 하는 기능을 구현할 것이다.

1. 팔로우 버튼을 토글하기 위해서는 우선적으로 현재 팔로우가 된 상태인지 아닌지,
   팔로우 여부 확인이 먼저 필요하다.
2. axios 요청을 통해 받는 response 객체를 활용하여

view 함수를 통해서 팔로우 여부를 파악 할 수 있도록 변수를 만들고,

그 변수에 팔로우 여부를 저장해서 JSON 타입으로 응답할 것이다.

팔로우 여부를 확인하기 위한 is_followed 변수를 새로 작성한 다음에
JSON 응답하는 코드를 추가해 보겠다.

가독성을 위해서 you 그리고 me라는 변수를 이용해서 추가해 보겠다.
me - 현재 로그인한 사용자
you - 팔로우 할 대상

```
# accounts/views.py


@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')



from django.http import JsonResponse

@require_POST
def follow(request, user_pk):
    if request.user.is_authenticated:
        User = get_user_model()
        me = request.user
        you = User.objects.get(pk=user_pk)
        if me != you: # 내가 나를 팔로우 하지는 않은 것이다 ㅋ
            if you.followers.filter(pk=me.pk).exists():
                you.followers.remove(me)
                is_followed = False
            else:
                you.followers.add(me)
                is_followed = True
            context = {
                'is_followed': is_followed,
```

```
            }
            return JsonResponse(context)
        return redirect('accounts:profile', you.username)
    return redirect('accounts:login')
```

현재 사용자가 자신을 팔로우하려고 하지 않는지 확인한 후

- you.followers에서 현재 사용자가 이미 팔로워인 경우, 이를 제거하고 언팔로우 한다.
- 그렇지 않은 경우, 사용자를 팔로워로 추가한다.
- 그리고 난 후, is_followed 상태를 포함한 JSON 응답을 반환한다.

  그러면 클라이언트 측에서 팔로우 상태를 업데이트하는 데 사용된다.

다시 profile.html 파일로 넘어가자.

view 함수에서 응답한 is_followed를 사용해 버튼을 토글할 것이다.

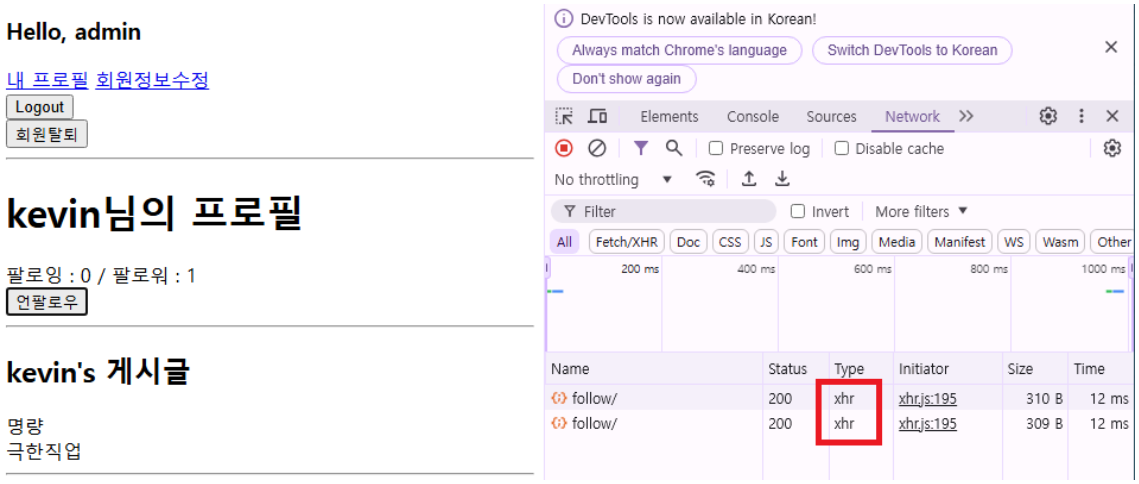
axios 요청에 .then 구문을 추가해 주자

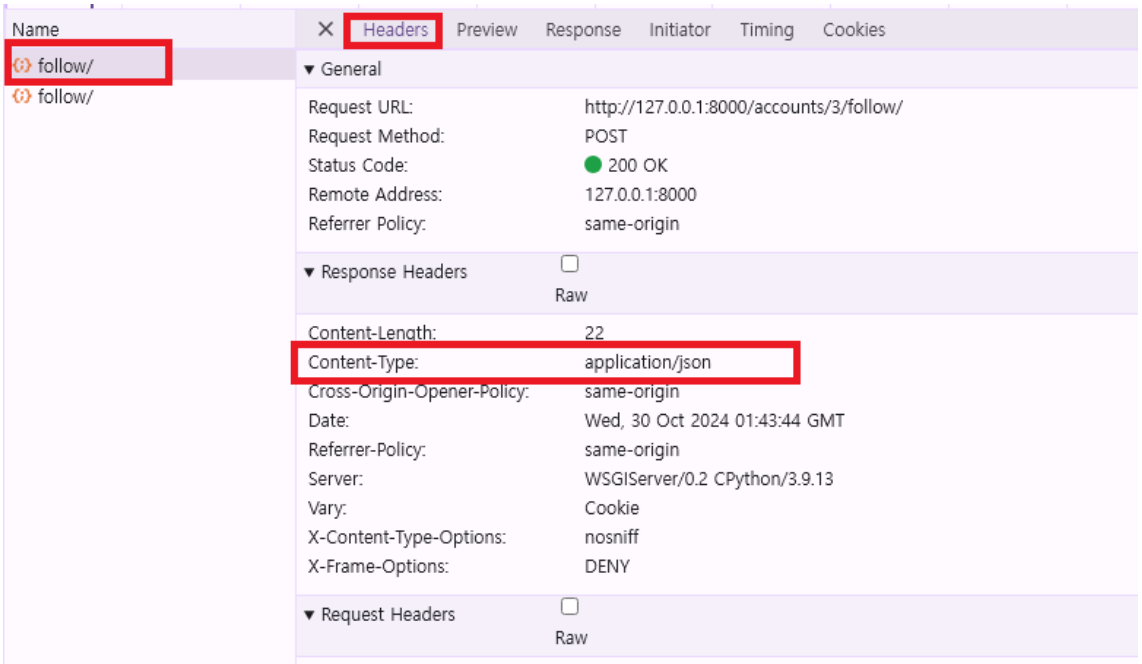
```
<!-- accounts/profile.html -->

<script>
 const form = document.querySelector('#follow-form')
 form.addEventListener('submit', function(event) {
  event.preventDefault()

  const userId = event.target.dataset.userId
  const csrftoken = document.querySelector('[name=csrfmiddlewaretoken]').value

  axios({
   method: 'post',
   url: `/accounts/${userId}/follow/`,
   headers: {'X-CSRFToken': csrftoken,}
  })
   .then((response) => {

    const isFollowed = response.data.is_followed
    const followBtn = document.querySelector('#follow-form > input[type=submit]')

    if (isFollowed === true) {
     // 팔로우 상태일 때는 언팔로우 버튼
     followBtn.value = '언팔로우'
    } else {
     followBtn.value = '팔로우'
    }
   })
```

```
  })
</script>
```

- .then을 통해서 서버에서의 요청이 성공정으로 처리 된 후
- response.data.is_followed 을 통해서 서버가 반환한 is_followed 값을 가져온다. 그다음
- #follow-form > input[type=submit] 을 통해서 버튼을 가리켜서 value 값을 팔로우 또는 언파로우로 변경해준다.

서버 켜서 확인해 보자. 새로고침 없이 팔로우가 작동이 될 것이다.





팔로워 & 팔로잉 수 비동기 적용

팔로우 또는 언팔로우 버튼을 선택 후

팔로잉수 또는 팔로워수가 바뀌는 것도 비동기를 적용시켜 보자.

- 해당 요소를 Dom 조작을 위해 선택할 수 있도록 span 태그와 id 속성을 추가적으로 작성해 준다.

```
<!-- accounts/profile.html -->

{% extends 'base.html' %}

{% block content %}
  <h1>{{ person.username }}님의 프로필</h1>
  <div>
    팔로워 : <span id="followers-count">{{ person.followers.all|length }}</span> /
    팔로잉 : <span id="followings-count">{{ person.followings.all|length }}</span>
  </div>
```

- 직전에 작성한 span 태그를 각각 선택

```
<!-- accounts/profile.html -->

<script>
  const form = document.querySelector('#follow-form')
  form.addEventListener('submit', function(event) {
    event.preventDefault()

    const userId = event.target.dataset.userId
    const csrftoken = document.querySelector('[name=csrfmiddlewaretoken]').value

    axios({
      method: 'post',
      url: `/accounts/${userId}/follow/`,
      headers: {'X-CSRFToken': csrftoken,}
    })
      .then((response) => {

        const isFollowed = response.data.is_followed
        const followBtn = document.querySelector('#follow-form > input[type=submit]')

        const followersCountTag = document.querySelector('#followers-count')
        const followingsCountTag = document.querySelector('#followings-count')

        if (isFollowed === true) {
          followBtn.value = '언팔로우'
        } else {
          followBtn.value = '팔로우'
        }
      })
```

```
  })
</script>
```

- 팔로워, 팔로잉 인원 수 연산은 view 함수에서 진행하고, 그 결과를 응답으로 전달할 것이다.

```
# accounts/views.py

from django.http import JsonResponse

@require_POST
def follow(request, user_pk):
    if request.user.is_authenticated:
        User = get_user_model()
        me = request.user
        you = User.objects.get(pk=user_pk)
        if me != you:
            if you.followers.filter(pk=me.pk).exists():
                you.followers.remove(me)
                is_followed = False
            else:
                you.followers.add(me)
                is_followed = True
            context = {
                'is_followed': is_followed,
                'followers_count': you.followers.count(),
                'followings_count': you.followings.count(),
            }
            return JsonResponse(context)
        return redirect('accounts:profile', you.username)
    return redirect('accounts:login')
```

view 함수에서 응답한 연산 결과를 사용해 각 태그의 인원 수 값 변경하기

```
<!-- accounts/profile.html -->

<script>
  const form = document.querySelector('#follow-form')
  form.addEventListener('submit', function(event) {
    event.preventDefault()

    const userId = event.target.dataset.userId
    const csrftoken = document.querySelector('[name=csrfmiddlewaretoken]').value

    axios({
```

```
    method: 'post',
    url: `/accounts/${userId}/follow/`,
    headers: {'X-CSRFToken': csrftoken,}
  })
    .then((response) => {

      const isFollowed = response.data.is_followed
      const followBtn = document.querySelector('#follow-form > input[type=submit]')

      const followersCountTag = document.querySelector('#followers-count')
      const followingsCountTag = document.querySelector('#followings-count')

      const followersCount = response.data.followers_count
      const followingsCount = response.data.followings_count

      followersCountTag.innerText = followersCount
      followingsCountTag.innerText = followingsCount

      if (isFollowed === true) {
        followBtn.value = '언팔로우'
      } else {
        followBtn.value = '팔로우'
      }
    })

  })
</script>
```

서버켜서 확인해 보면 새로고침 없이 팔로우/언팔로우가 작동되는 동시에

팔로우 숫자도 새로고침 없이 변경이 잘 될것이다.

만약에 서버켜서 확인 후 잘 작동이 되지 않는다면 아래 한번 더 정리 해 둔

최종 코드를 보면서 버그를 찾아보자.

최종 코드

HTML

```
<!-- accounts/profile.html -->

{% extends 'base.html' %}

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

```
  팔로워 : <span id="followers-count">{{ person.followers.all|length }}</span> /
  팔로잉 : <span id="followings-count">{{ person.followings.all|length }}</span>
</div>

{% if request.user != person %}
<div>
 <form id="follow-form" data-user-id="{{ person.pk }}">
  {% csrf_token %}
  {% if request.user in person.followers.all %}
   <input type="submit" value="언팔로우">
  {% else %}
   <input type="submit" value="팔로우">
  {% endif %}
 </form>
<div>
{% endif %}
...
```

Python

```
# accounts/views.py

from django.http import JsonResponse

@require_POST
def follow(request, user_pk):
    if request.user.is_authenticated:
        User = get_user_model()
        me = request.user
        you = User.objects.get(pk=user_pk)
        if me != you:
            if you.followers.filter(pk=me.pk).exists():
                you.followers.remove(me)
                is_followed = False
            else:
                you.followers.add(me)
                is_followed = True
            context = {
                'is_followed': is_followed,
                'followers_count': you.followers.count(),
                'followings_count': you.followings.count(),
            }
            return JsonResponse(context)
        return redirect('accounts:profile', you.username)
    return redirect('accounts:login')
```

JavaScript

```
<!-- accounts/profile.html -->

{% block script %}
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
<script>
 const form = document.querySelector('#follow-form')
 form.addEventListener('submit', function(event) {
  event.preventDefault()

  const userId = event.target.dataset.userId
  const csrftoken = document.querySelector('[name=csrfmiddlewaretoken]').value

  axios({
   method: 'post',
   url: `/accounts/${userId}/follow/`,
   headers: {'X-CSRFToken': csrftoken,}
  })
   .then((response) => {

    const isFollowed = response.data.is_followed
    const followBtn = document.querySelector('#follow-form > input[type=submit]')

    const followersCountTag = document.querySelector('#followers-count')
    const followingsCountTag = document.querySelector('#followings-count')

    const followersCount = response.data.followers_count
    const followingsCount = response.data.followings_count
    followersCountTag.innerText = followersCount
    followingsCountTag.innerText = followingsCount

    if (isFollowed === true) {
     followBtn.value = '언팔로우'
    } else {
     followBtn.value = '팔로우'
    }
   })

 })
</script>
{% endblock script %}
```

[부록] 부트스트랩 적용하기

- HTML

```
<!-- accounts/profile.html -->

{% extends 'base.html' %}

{% block content %}
 <h1>{{ person.username }}님의 프로필</h1>
 <div>
  팔로워 : <span id="followers-count">{{ person.followers.all|length }}</span> /
  팔로잉 : <span id="followings-count">{{ person.followings.all|length }}</span>
 </div>

 {% if request.user != person %}
 <div>
  <form id="follow-form" data-user-id="{{ person.pk }}">
   {% csrf_token %}
   {% if request.user in person.followers.all %}
    <button type="submit" class="btn btn-secondary">언팔로우</button>
   {% else %}
    <button type="submit" class="btn btn-primary">팔로우</button>
   {% endif %}
  </form>
 <div>
 {% endif %}
```

- JavaScript

```
<!-- accounts/profile.html -->

{% block script %}
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
<script>
 const form = document.querySelector('#follow-form')
 form.addEventListener('submit', function(event) {
  event.preventDefault()

  const userId = event.target.dataset.userId
  const csrftoken = document.querySelector('[name=csrfmiddlewaretoken]').value

  axios({
   method: 'post',
   url: `/accounts/${userId}/follow/`,
   headers: {'X-CSRFToken': csrftoken,}
  })
   .then((response) => {

    const isFollowed = response.data.is_followed
    const followBtn = document.querySelector('#follow-form > button[type=submit]')
```

```
      const followersCountTag = document.querySelector('#followers-count')
      const followingsCountTag = document.querySelector('#followings-count')

      followBtn.classList.toggle('btn-secondary')
      followBtn.classList.toggle('btn-primary')

      const followersCount = response.data.followers_count
      const followingsCount = response.data.followings_count
      followersCountTag.innerText = followersCount
      followingsCountTag.innerText = followingsCount

      if (isFollowed === true) {
        followBtn.innerText = '언팔로우'
      } else {
        followBtn.innerText = '팔로우'
      }
    })

  })
</script>
{% endblock script %}
```

비동기 적용하기 - 좋아요 (like)

- 좋아요 버튼에 비동기 적용은
  - "팔로우와 동일한 흐름 + `forEach()` & `querySelectorAll()`" 만 해주면 된다.

    그 이유는 index 페이지의 각 게시글마다 좋아요 버튼을 생성 할 것이기 때문이다.
  - 물론 버블링을 이용한 구현도 있지만 나는 개인적으로 버블링을 좋아하지 않는다
    - 프로필 페이지 처럼 하나의 버튼만 있는 경우에는 버블링을 이용한 방법도 좋겠지만,

      만약에 index 페이지 처럼 여러 개시글에 여러개의 좋아요 버튼이 존재 하고 독립적으로 동작을 해야 한다면 forEach()방식이 유지보수 및 디버깅이 쉽기 때문이다.

accouts 앱이 아니라 이제는 movis앱으로 이동하자.

HTML

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

    {% comment %} 여기서 부터 추가 작성  {% endcomment %}

    <p><span id="liked-count-{{movie.pk}}"></span></p>
      <div>
        <form class="like-forms" data-movie-id="{{ movie.pk }}">
          {% csrf_token %}
          {% if request.user in movie.like_users.all %}
            <input type="submit" value="좋아요 취소" id="like-{{ movie.pk }}">
          {% else %}
            <input type="submit" value="좋아요" id="like-{{ movie.pk }}">
          {% endif %}
        </form>
      </div>
      <a href="{% url 'movies:detail' movie.pk %}">상세 페이지</a>
      <hr>
  {% endfor %}
{% endblock %}
```

그다음 views.py로 넘어가자.

Python

```
# movies/views.py

from django.http import JsonResponse

@require_POST
def likes(request, movie_pk):
    if request.user.is_authenticated:
        movie = Movie.objects.get(pk=movie_pk)

        if movie.like_users.filter(pk=request.user.pk).exists():
            movie.like_users.remove(request.user)
            is_liked = False
        else:
            movie.like_users.add(request.user)
            is_liked = True
        context = {
            'is_liked': is_liked,
            'liked_count': movie.like_users.count(),
```

```
        }
        return JsonResponse(context)
    return redirect('accounts:login')
```

그 다음은 새로고침 없이 좋아요가 작동 할 수 있도록

위에 정의한 view함수에 요청을 보내자.

JavaScript

```
# movies/views.py

{% block content %}

{% endblock %}


{% block script %}
 <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
 <script>
  const forms = document.querySelectorAll('.like-forms')
  const csrftoken = document.querySelector('[name=csrfmiddlewaretoken]').value

    forms.forEach((form) => {
   form.addEventListener('submit', function (event) {
    event.preventDefault()  // form의 기본 기능 막아야 새로고침이 되지 않음

    const movieId = event.target.dataset.movieId

    axios({
     method: 'post',
     url: `http://127.0.0.1:8000/movies/${movieId}/likes/`,
     headers: {'X-CSRFToken': csrftoken},
    })
    .then((response) => {

     const isLiked = response.data.is_liked
     const likeBtn = document.querySelector(`#like-${movieId}`)
     if (isLiked === true) {
      likeBtn.value = '좋아요 취소'
     } else {
      likeBtn.value = '좋아요'
     }

     const likedCount=response.data.liked_count
     const likedCountTag=document.querySelector(`#like-count-${movieId}`)
     likedCountTag.textContent = likedCount
```

```
      })
      .catch((error) => {
        console.log(error.response)
      })
    })
  })
</script>
{% endblock script %}
```

새로고침 없이 "좋아요"가 잘 작동되는지 확인해 보자.

<끝>

follow_like_async(completed).zip