8기 파이썬반 PJT - 8 (2)

Created	@2022년 10월 11일 오후 11:54
Last Edited Time	@2022년 11월 2일 오후 2:46
Type	
Status	
Created By	
Last Edited By	
Stakeholders	

2. accounts

미리 `urls.py` 를 정의해두자.

다음 조건에 맞춰 작성한다.

모든 URL 패턴은 `accounts/` 로 시작한다.

HTTP verb	URL 패턴	설명
GET & POST	signup/	Form 표시 및 신규 사용자 생성 (회원가입)
GET & POST	login/	Form 표시 및 기존 사용자 인증 (로그인)
GET & POST	logout/	인증된 사용자 인증 해제 (로그아웃)
GET	<username>/	사용자 상세 조회 페이지
POST	<user_pk>/follow/	팔로우 추가, 팔로우 취소

```
# accounts/urls.py

from django.urls import path
from . import views

app_name = 'accounts'

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

여기서, username 은 타입을 따로 정하지 않았음에 유의해보자. 실제론 `str` 으로 받을 것이지만, 원래 설정하지 않으면 `str` 이 기본값이다.

`views.py` 작성 전, `forms.py` 를 만들겠다.

```
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth import get_user_model


class CustomUserCreationForm(UserCreationForm):

    class Meta:
        model = get_user_model()
        fields = ('username', 'email',)
```

`UserCreationForm` 을, `CustomUserCreationForm` 으로 상속받아 오버라이딩할것이다.

`get_user_model()` 을 사용해 Django 에서 제공하는 유저 모델을 그대로 가져다쓰고, 사용하고자 하는 필드, `username` 과 `email` 만 가져다쓰겠다.

`signup` 함수를 `views.py` 에 작성해보자.

```
# accounts/views.py

from django.shortcuts import render, redirect
from django.contrib.auth import login as auth_login
from django.views.decorators.http import require_http_methods
from .forms import CustomUserCreationForm

# Create your views here.

@require_http_methods(['GET', 'POST'])
def signup(request):
    if request.user.is_authenticated:
        return redirect('community:index')

    if request.method == 'POST':
        form = CustomUserCreationForm(request.POST)
        if form.is_valid():
            user = form.save()
            auth_login(request, user)
            return redirect('community:index')
    else:
        form = CustomUserCreationForm()
    context = {
        'form': form,
    }
    return render(request, 'accounts/signup.html', context)
```

```
def login(request):
    pass

def logout(request):
    pass

def profile(request, username):
    pass

def follow(request, user_pk):
    pass
```

`urls.py` 를 미리 작성했기에, 에러 방지를 위해 미리 함수들을 만들어두었다.

사용자가 로그인한 상태라면 회원가입 페이지에 들어오지 못하고 `community:index` 로 향한다.

로그인 안 한 상태일 경우라면 POST 일 경우와 GET 일 경우로 달라진다.

POST 는 `signup.html` 에서만 보낼 수 있다. 사용자가 폼을 다 작성 후에 `submit` 버튼을 눌러야만 동작하는 부분이다.

사용자가 입력한 폼이 양식에 맞는지(`is_valid`) 판단한 후, 맞으면 회원가입 후 로그인하고, `community:index` 로 리다이렉트한다.

만약, 양식에 맞지 않다면 if 문은 실행되지 않고 곧바로 `context` 생성 부분으로 넘어가게 되어, 다시 `signup.html` 페이지가 보여지고, 사용자가 입력한 내용이 그대로 유지되도록 한다.

GET 은 회원가입 링크를 클릭했을 때 동작하는데, 코드 상에선 `else` 부분이다. 회원가입 폼을 받아서, `context` 를 만든 다음 사용자에게 보여주게된다.

회원가입 페이지를 보여주기 위해, 먼저 `base.html` 을 수정한다.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <nav>
      {% if user.is_authenticated %}
```

```
        <h3>Hello, {{ user.username }}</h3>
      {% else %}
        <a href="{% url 'accounts:signup' %}">Signup</a>
      {% endif %}
    </nav>
    <hr>
    <div>
      {% block content %}
      {% endblock content %}
    </div>
  </body>
</html>
```

`<nav>` 태그 안에, 만약 사용자가 로그인한 상태라면 이름이 출력되고, 그렇지 않다면 회원가입 링크를 클릭할 수 있도록 만들었다.

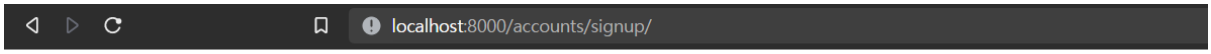
다음, `accounts/templates/accounts/signup.html` 을 다음과 같이 작성한다.

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

{% block content %}
  <h1>Signup</h1>
  <form action="{% url 'accounts:signup' %}" method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Submit</button>
  </form>
{% endblock content %}
```

`forms.py` 에서 만든 폼을 `<p>` 태그 형식으로 사용할 것이다. POST 방식으로 보낼 것이기에 `csrf_token` 을 빼먹지 말자.

로그인 하지 않은 상태에선 다음과 같이 나온다.



Signup

Signup

사용자 이름: 150자 이하 문자, 숫자 그리고 @/./+/-/_만 가능합니다.

이메일 주소:

비밀번호:

- 다른 개인 정보와 유사한 비밀번호는 사용할 수 없습니다.
- 비밀번호는 최소 8자 이상이어야 합니다.
- 통상적으로 자주 사용되는 비밀번호는 사용할 수 없습니다.
- 숫자로만 이루어진 비밀번호는 사용할 수 없습니다.

비밀번호 확인: 확인을 위해 이전과 동일한 비밀번호를 입력하세요.

Submit

기본 제공되는 폼이 확인된다. 먼저 유저를 1명 생성해보겠다.

페이지가 리다이렉트 되면서, 회원가입 링크는 더이상 보이지 않고, “Hello, 유저이름” 이 잘 나타남이 확인된다.

이 사용자를 로그아웃시켜야하므로, 로그아웃 기능을 제작해보자.

먼저, `base.html` 의 `<nav>` 부분만 다음과 같이 수정한다.

```
<nav>
  {% if user.is_authenticated %}
    <h3>Hello, {{ user.username }}</h3>
    <form action="{% url 'accounts:logout' %}" method="POST">
      {% csrf_token %}
      <input type="submit" value="Logout" />
    </form>
  {% else %}
    <a href="{% url 'accounts:signup' %}">Signup</a>
  {% endif %}
</nav>
```

자세히보면, 로그아웃이 추가되었다. 당연히, 로그인이 된 상태에서만 보여야한다.

`views.py` 에 `logout` 함수를 다음과 같이 작성한다.

```
# accounts/views.py
from django.contrib.auth import logout as auth_logout
from django.views.decorators.http import require_http_methods, require_POST

@require_POST
def logout(request):
    if request.user.is_authenticated:
        auth_logout(request)
    return redirect('community:index')
```

만약 사용자가 로그인된 상태라면 로그아웃 진행하고 `community:index` 로 리다이렉트, 로그인 안한 상태라면 그냥 리다이렉트이다.

로그아웃 버튼이 생겼고,

클릭하면 정상적으로 로그아웃된다.

유저를 두 명 더 만들어보자.

이제 로그인을 만들어보자. `base.html` 은 다음과 같이 수정한다.

```
<nav>
  {% if user.is_authenticated %}
    <h3>Hello, {{ user.username }}</h3>
    <form action="{% url 'accounts:logout' %}" method="POST">
      {% csrf_token %}
      <input type="submit" value="Logout" />
    </form>
  {% else %}
    <a href="{% url 'accounts:login' %}">Login</a>
    <a href="{% url 'accounts:signup' %}">Signup</a>
  {% endif %}
</nav>
```

`else` 구문에서, 회원가입 바로 위에 로그인 링크를 걸었다.

`accounts/views.py` 에서 `login` 함수를 작성해보자.

```
from django.contrib.auth.forms import AuthenticationForm

@require_http_methods(['GET', 'POST'])
def login(request):
    if request.user.is_authenticated:
        return redirect('community:index')

    if request.method == 'POST':
        form = AuthenticationForm(request, request.POST)
        if form.is_valid():
            auth_login(request, form.get_user())
            return redirect('community:index')
    else:
        form = AuthenticationForm()
    context = {
        'form': form,
```

```
    }
    return render(request, 'accounts/login.html', context)
```

`signup` 함수와 형태가 매우 비슷하니 자세한 설명은 생략한다.

다만, 여기선 로그인 폼을 사용하기 위해 Django 에서 기존 제공하는 `AuthenticationForm` 을 사용했고, `get_user` 함수를 사용해 로그인할 유저가 누구인지 가져왔다.

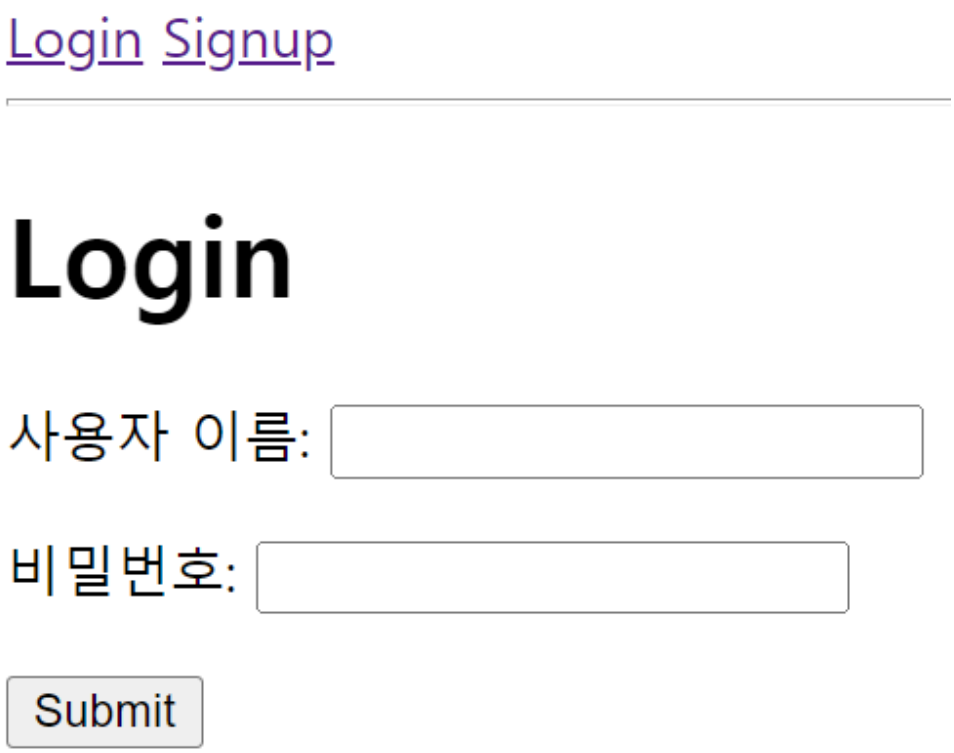
`accounts/login.html` 을 만들어보자.

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

{% block content %}
  <h1>Login</h1>
  <form action="{% url 'accounts:login' %}" method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Submit</button>
  </form>
{% endblock content %}
```

`signup.html` 과 구조상 큰 차이는 없다.

로그인이 생겼고,



로그인 페이지가 생겼다.

로그인 성공 시 화면.

다른 두 명의 유저로도 테스트해보자.

다음은 사용자 상세 페이지 조회 기능 구현이다.

```
# accounts/views.py
```

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth import get_user_model
from django.contrib.auth.decorators import login_required

@login_required
def profile(request, username):
    person = get_object_or_404(get_user_model(), username=username)
    context = {
        'person': person,
    }
    return render(request, 'accounts/profile.html', context)
```

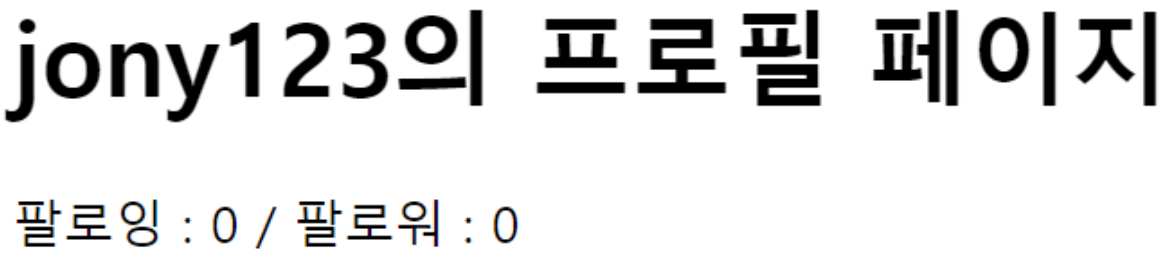
유저 아이디를 파라미터로 받고, `get_user_model()` 을 사용해 유저 모델에 접근하되, `username=username` 을 사용해 파라미터로 받은 유저의 데이터를 받아오도록 하고, 그것을 `person` 으로 받아 `accounts/profile.html` 로 넘겨준다.

우선, `base.html` 에서, 다음과 같이 `profile.html` 페이지로 갈 수 있는 `<a>` 태그 하나를 설정하겠다.

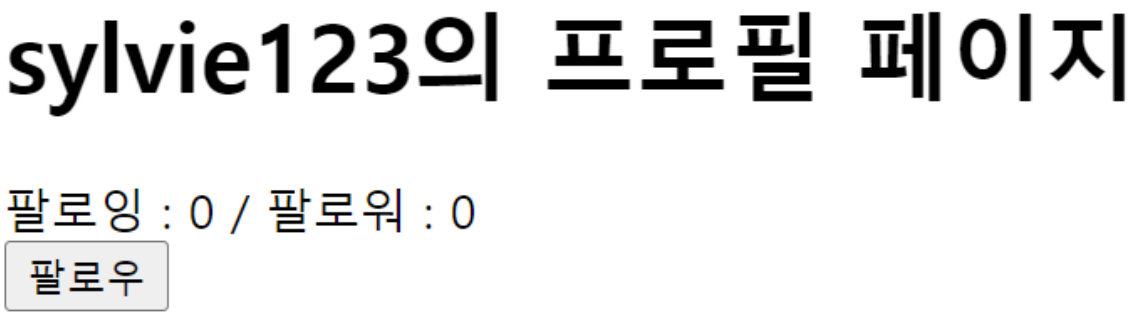
```
<nav>
      {% if user.is_authenticated %}
      <h3>Hello, {{ user.username }}</h3>
      <a href="{% url 'accounts:profile' user %}">내 프로필<a>
      <form action="{% url 'accounts:logout' %}" method="POST">
      ...
```

내 프로필 하이퍼링크를 클릭하면 `profile.html` 로 향할 것이다.

`profile.html` 을 살펴보기 전에, 구현할 화면만 보면 다음과 같다.



현재 팔로잉이 몇 명이고, 팔로워가 몇 명인지를 보여준다.
만약, 로그인한 상태에서 자기 자신의 아이디가 아니라, 다른 사람의 아이디로 접속하면?
즉, `localhost:8000/accounts/anotheruserid` 로 접속하면?



jony123 으로 로그인한 상태에서 `localhost:8000/accounts/sylvie123` 으로 접속한 경우.
즉, 다른 사람의 페이지에 방문했을때는,

- 그 사람을 팔로우했을 경우엔 언팔로우 버튼이,
- 언팔로우했을 경우엔 팔로우 버튼이

나오도록 되어있다.

이것에대한 화면만 구현해보면 다음과 같다.

`accounts/profile.html`

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

{% block content %}
  <h1>{{ person.username }}의 프로필 페이지</h1>
  {% with followings=person.followings.all followers=person.followers.all %}
    <div>
      <div id="follow-count">
        <div>팔로잉 : {{ followings|length }} / 팔로워 : {{ followers|length }}</div>
      </div>
      {% if request.user != person %}
        <div>
          <form id="follow-form" data-user-id="{{ person.pk }}">
            {% csrf_token %}
            {% if request.user in followers %}
              <button>언팔로우</button>
            {% else %}
              <button>팔로우</button>
            {% endif %}
          </form>
        </div>
      {% endif %}
    </div>
  {% endwith %}
{% endblock %}
```

`{ % with %}` 은, 쓰고자 하는 데이터가 너무 길 때, 변수 지정 역할을 한다. 즉, 이 페이지에선 `person.followings.all` 이 곧 `followings` 가 되고, `person.followers.all` 이 곧 `followers` 가 된다.

현재 팔로잉과 팔로워가 몇 명인지 `|length` 를 통해 계산한 결과를 보여준다.

만약, 현재 로그인한 사용자와 일치하지 않으면, 팔로우 또는 언팔로우 버튼을 보여줄 수 있다.

`data-` 로 시작되는 애트리뷰트는 사용자가 정의할 수 있는데, `event.target.dataset` 에 정의되고, 받을 땐 `userId` 로, camelCase 로 받을 것이다.

이제, 각각의 HTML 파일 내에서 `{ % block script % }` 를 사용하기 위해 `base.html` 로 이동해서 작업해보자.

```
    {% block content %}
    {% endblock %}
  </div>

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

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

맨 아랫쪽에, axios 를 쓰기 위한 CDN 과, `{% block script %}` 를 추가했다.

즉, 각각의 페이지에서 JavaScript 는 `{ % block script % }` 안에서 정의해서 쓰면 되고, 각각의 페이지는 모두 axios 를 사용할 수 있다.

`profile.html` 에서 버튼을 누를 경우, `'<int:user_pk>/follow/'` 로 요청을 주게 될 것이다. 따라서, 해당 url 에 연결될 함수인 `follow` 를 작성해야한다. `accounts/views.py` 로 향하자.

```
from django.http import JsonResponse, HttpResponse

@require_POST
def follow(request, user_pk):
    if request.user.is_authenticated:
        person = get_object_or_404(get_user_model(), pk=user_pk)
        user = request.user
        if person != user:
            if person.followers.filter(pk=user.pk).exists():
                person.followers.remove(user)
                followed = False
            else:
                person.followers.add(user)
                followed = True
            context = {
                'isFollowed': followed,
                'followers_count': person.followers.count(),
                'followings_count': person.followings.count(),
            }
            return JsonResponse(context)
    return HttpResponse(status=401)
```

이 함수는 페이지를 보여주는 역할을 하지 않고, 성공 시 JSON 리턴하는 역할을 한다.

로그인되어있지 않거나, 팔로우하려는 사용자가 본인인 경우엔 HTTP 401 status, 즉 인증되지 않은 사용자라는 신호를 보내준다. 이것은 버튼이 보일 수 있는 가장 기본적인 조건이다.

그렇다면, 여기서 `user_pk` 는 로그인된 사용자 본인인가? 아니다. 팔로우하고자하는 대상이다.

해당 유저의 데이터를 `person` 으로 가져오고, 현재 로그인된 사용자는 `user` 에 담아 둘의 비교부터해서 “불일치” 할 경우에만 팔로우, 언팔로우를 진행한다.

`person.followers.filter(pk=user.pk).exists()` 는 무슨 뜻인가? 현재 팔로우하고자하는 사람의 팔로워정보를 봤을 때, 로그인한 유저가 있는지를 말한다. 이것이 참이라는 것은, 이미 팔로우

하고있다는 뜻이므로 언팔로우를 진행해야한다.

그 반대라면 팔로우를 진행하면 된다. 그리고 비동기통신에 응답할 플래그, `followed` 에 팔로우 결과를 담는다.

`context` 는 비동기통신이 올 때 전달해야할 데이터셋이다. 팔로우/언팔로우 여부, 팔로워는 몇명인지, 팔로잉은 몇명인지를 JSON 리턴한다.

이제 남은 건 하나인데, `accounts/profile.html` 에서 `{% block script %}` 부분을 작성해주는 것이다.

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

{% block content %}
  <h1>{{ person.username }}의 프로필 페이지</h1>
  {% with followings=person.followings.all followers=person.followers.all %}
    <div>
      <div id="follow-count">
        <div>팔로잉 : {{ followings|length }} / 팔로워 : {{ followers|length }}</div>
      </div>
      {% if request.user != person %}
        <div>
          <form id="follow-form" data-user-id="{{ person.pk }}">
            {% csrf_token %}
            {% if request.user in followers %}
              <button>언팔로우</button>
            {% else %}
              <button>팔로우</button>
            {% endif %}
          </form>
        </div>
      {% endif %}
    </div>
  {% endwith %}
{% endblock %}

{% block script %}
<script>
  const form = document.querySelector("#follow-form");
  const csrftoken = document.querySelector("[name=csrfmiddlewaretoken]").value;

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

    axios({
      method: "post",
      url: `/accounts/${userId}/follow/`,
      headers: { "X-CSRFToken": csrftoken },
    }).then((response) => {
      const isFollowed = response.data.isFollowed;
      const followBtn = document.querySelector("#follow-form > button");
      const followers_count = response.data.followers_count;
      const followings_count = response.data.followings_count;
```

```
      const followCountDiv = document.querySelector("#follow-count > div");

      if (isFollowed === true) {
        followBtn.innerText = "언팔로우";
      } else {
        followBtn.innerText = "팔로우";
      }
      followCountDiv.innerText = `팔로잉 : ${followings_count} / 팔로워 : ${followers_count}`;
    });
  });
</script>
{% endblock script %}
```

새로 쓰인 부분은 `{ % block script % }` 뿐이다. 나머지 부분은 동일하나, 굳이 모든 코드를 다 써준 이유는 `script` 는 항상 html 즉 DOM 을 참고하면서 작성해야하기 때문이다.

이 코드를 하나하나 세부적으로 분석해보겠다.

일단, 제어해야할 부분은 이 부분이다.

```
<form id="follow-form" data-user-id="{{ person.pk }}">
  {% csrf_token %}
  {% if request.user in followers %}
    <button>언팔로우</button>
  {% else %}
    <button>팔로우</button>
  {% endif %}
</form>
```

즉, `<form>` 태그이다. 지금까지는, 이 `<form>` 태그에서 `submit` 시엔 적당한 경로를 설정해 결과를 받아오면 리다이렉트하는 식으로 처리했다. 지금까지 배운 지식으로 팔로잉 기능을 구현 못하는게 절대 아니다.

다만, axios 를 사용한 비동기통신을 할 경우, “새로고침이 발생하지 않는다.” 이것은 곧 무엇을 의미하나? 사용자가 버튼을 누르면, 다른 페이지로 요청을 보냈다가, 화면 전체를 변경하지 않고 변경이 일어난 부분만 다시 그리는 것을 의미한다. 즉, 사용자 입장에선 버튼 하나 눌렀다고 불친절하게 새로고침이 되는 게 아니라, 아무 일 없었다는듯 팔로우/팔로잉 숫자만 바뀌는, 마치 네이티브 앱처럼 동작하는 부드러운 웹을 사용할 수 있게 된다.

그래서, 이 `<form>` 태그의 DOM 정보를 일단 가져온다. 아이디인 `follow-form` 에 접근할 것이다.

```
const form = document.querySelector('#follow-form');
const csrftoken = document.querySelector('[name=csrfmiddlewaretoken]').value;
```

`csrftoken` 은 장고에서 제공하는 `csrfmiddlewaretoken` 의 값을 가져와서, `csrf` 에러가 일어나지 않도록 미리 받아두는 용도다.

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

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

여기서, `submit` 에 해당하는 이벤트 발생 시 작동할 이벤트리스너 추가를 위해 `addEventListener` 를 사용해 콜백함수를 등록한다. 그런데, 우린 `submit` 을 준 적이 없고 그냥 버튼 두 개만 달지 않았는가? 상관없다. 해당 버튼이 있다면 그것을 `submit` 으로 인식할 것이다.

`event.preventDefault()` 는 다른 화면으로 전환되는것을 방지한다. `<form>` 태그는 `submit` 이 이루어지는순간 필연적으로 목적 페이지로의 전환이 이루어지는데, 비동기통신을 할 것이므로 그렇게 되면 안되므로 해당 함수로 방지해주었다.

여기서, `event.target.dataset.userId` 는 아까 말했듯, `person.pk` 를 의미한다. 로그인한 사용자가 아니라, 팔로우하고자하는 대상이다.

그리고 `axios` 를 사용해 비동기통신을 할 것인데, http method 는 `post` 이며, 접속하고자하는 url은 `/accounts/${userId}/follow/` 이고, POST 통신 때 필요한, 아까 변수로 받아둔 `csrftoken` 을 넣어준다.

그런데, 실제 `accounts/urls.py` 에 작성된 url 은 무엇인가? `<int:user_pk>/follow/` 이다. 왜 일치하지 않는가? axios 로 통신할 땐, 현재 서버가 동작하고 있는 `localhost:8000` 에서부터 시작해야하기 때문이다. axios 는 JavaScript 패키지이기 때문에, 장고의 상황을 모른다고 봐야한다. 차라리, JavaScript 로 쓰여진 영역은 Python 으로 쓰여진 영역과는 아무 상관없다고 생각하는게 쉽다.

에러처리구문은 따로 넣진 않았는데, 통신 성공했을 때, `.then()` 이 호출된다. 성공했다는 뜻은 JSON 을 받았다는 뜻인데, 어떤 형태로 받았는지 기억하는가?

```
context = {
    'isFollowed': followed,
    'followers_count': person.followers.count(),
    'followings_count': person.followings.count(),
}
```

즉, 팔로우했는지 여부, 팔로워/팔로잉 수

이 세가지를 리턴한다.

이제 `.then()` 의 구현부를 보자.

```
const isFollowed = response.data.isFollowed;
const followBtn = document.querySelector("#follow-form > button");
const followers_count = response.data.followers_count;
const followings_count = response.data.followings_count;
const followCountDiv = document.querySelector("#follow-count > div");

if (isFollowed === true) {
  followBtn.innerText = "언팔로우";
} else {
  followBtn.innerText = "팔로우";
}
followCountDiv.innerText = `팔로잉 : ${followings_count} / 팔로워 : ${followers_count}`;
```

`response.data` 로 성공적으로 받아온 데이터는 각각 `isFollowed`, `followers_count`, `followings_count` 로, 동일한 이름으로 받는데, 더 이상 Python이 아니라 JavaScript 임을 기억하자.

그리고, `<button>` 태그를 `followBtn` 으로 가져오는데, if 문으로 분기해놔서 그렇지 사실 버튼은 하나만 존재한다.

또한, 실제 몇 명의 팔로잉, 팔로워가 있는지 나타내기위해 그 텍스트영역도 `followCountDiv` 로 가져온다.

그래서, 버튼에 보여줄 텍스트를, 팔로우한 경우엔 `언팔로우` 버튼이 보이도록 하고, 팔로우하지 않은 경우엔 `팔로우` 버튼이 보일 수 있도록, `innerText` 를 사용해 상황에 따라 바꿔준다.

마지막으로 `followCountDiv` 의 텍스트에 들어갈 변수를 `followings_count` 와 `followers_count` 로 지정해주고 텍스트를 바꾸면된다.

테스트해보자. 적어도 두 명의 유저로 로그인을 바꿔보면서, 팔로잉, 팔로워가 어떻게 동작되는지 확인해보자.

Hello, sylvie123

내 프로필

Logout

---

jony123의 프로필 페이지

팔로잉 : 1 / 팔로워 : 0

팔로우