Authentication

우리의 메모로그에 아무나 글을 작성하게 할 수는 없습니다! 이제부터는 웹 페이지를 회원제로 변경해보겠습니다. 회원가입과 로그인 기능은 장고에서 제공해 주는 기능이기 때문에 기타 다른 설치없이 바로 코드로 진행해보도록 하겠습니다.

1. Application 세팅

먼저 회원가입을 관리할 앱을 만들겠습니다. Django는 기능별로 앱을 만드는 것을 권장하기 때문에 새로운 앱을 생성해주겠습니다!

```
python3 manage.py startapp accounts
```

앱을 만들었으니 세팅해주어야 할 것들이 있었죠? settings.py에서 INSTALLED_APPS에 'accounts'를 추가해줍니다. 다음으로는 urls.py에 세팅을 해줍니다. 앱이 2개 생겼으니 urls.py를 분리하여 관리해보겠습니다. 먼저 accounts 앱 안에 urls.py를 만들어 세팅합니다.

`accounts/urls.py`

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

urlpatterns = [
    path('signup/', views.signup, name='signup'),
    path('login/', views.login, name='login'),
]
```

그리고 프로젝트의 앱 안에도 세팅을 해줍니다.

`urls.py`

```
from django.contrib import admin
from django.urls import path, include # 앱에 작성한 urls.py를 가져오기 위한 include입니다.
import memolog.views
import accounts.views # 새로운 앱이 생겼으니 그 안의 views.py도 여기서 관리해야겠죠?

urlpatterns = [
    path('admin/', admin.site.urls),

              ...

    path('accounts/', include('accounts.urls')),
]
```

앱과 프로젝트 안에 모두 url을 설정해줬으면 다음으로는 템플릿을 만들어 보도록 하겠습니다. `accounts/templates/accounts`에 login.html과 signup.html을 만들어 주세요.

회원가입을 할 때는 이메일과 비밀번호를 받게 됩니다. 그에 맞는 input form을 만들어 회원가입을 구현해봅시다.

2. Template 구현

페이지부터 띄우고 기능 구현에 들어갑시다. 먼저 회원가입 페이지입니다. username 필드가 필수적으로 들어가야 합니다. username이라고 하면 조금 모호하기 때문에 우리는 여기에 이메일을 받도록 하겠습니다.

- `(참고)` usermodel

`templates/accounts/signup.html`

```
<form>
  <h1>회원가입 페이지입니다.</h1>
    <label for="inputEmail">이메일</label>
    <input type="email" id="inputEmail" name="username" placeholder="이메일을 입력해주세요."> <!--username 필드가 꼭 들어가야 하기 때문에 na
    <br>
    <label for="inputPassword">비밀번호</label>
    <input type="password" id="inputPassword" name="password1" placeholder="비밀번호를 입력해주세요">
    <br>
    <label for="inputPassword">비밀번호 확인</label>
    <input type="password" id="inputPassword" name="password2" placeholder="비밀번호를 입력해주세요">
    <br>

    <input type="submit" value="회원가입하기">
</form>
```

다음으로 로그인 페이지입니다.

`templates/accounts/login.html`

```
<form>
  <h1>로그인 페이지 입니다.</h1>
    <label for="inputEmail">이메일</label>
    <input type="email" id="inputEmail" name="username" placeholder="이메일을 입력해주세요.">
    <br>
    <label for="inputPassword">비밀번호</label>
    <input type="password" id="inputPassword" name="password" placeholder="비밀번호를 입력해주세요">
    <br>

    <input type="submit" value="로그인하기">
</form>
```

두 페이지를 모두 작성했다면 views.py에서 함수를 작성해보겠습니다.

`accounts/views.py`

```
def signup(request):
    return render(request, 'accounts/signup.html')

def login(request):
    return render(request, 'accounts/login.html')
```

회원가입과 로그인으로 들어갈 링크도 index 페이지에 달아줍니다.

templates/blog/index.html

```
<a href="{% url 'signup' %}">sign up</a>
<a href="{% url 'login' %}">log in</a>
```

여기까지 잘 따라 오셨나요? 다음은 기능 구현입니다!

3. sign up 구현

먼저 signup.html을 수정해보겠습니다.

templates/accounts/signup.html

```
{% if error %}
{{ error }}
{% endif %}

<form method="POST" action="{% url 'signup' %}">
  {% csrf_token %}
  <h1>회원 가입 페이지 입니다.</h1>
    <label for="inputEmail">이메일</label>
    <input type="email" id="inputEmail" name="username" placeholder="이메일을 입력해주세요.">
    <br>
    <label for="inputPassword">비밀번호</label>
    <input type="password" id="inputPassword" name="password1" placeholder="비밀번호를 입력해주세요">
    <br>
    <label for="inputPassword">비밀번호 확인</label>
    <input type="password" id="inputPassword" name="password2" placeholder="비밀번호를 입력해주세요">
    <br>

    <input type="submit" value="회원가입하기">
</form>
```

복잡해보여도 저번에 배웠던 http method 중 POST를 잠깐 떠올려보면 어렵지 않습니다. post 방식을 통해서 서버에 보내는 정보가 노출되는 것을 방지하고 사이트 요청 위조를 막기 위해서 `{% csrf_token %}`을 사용했습니다. 이제는 views.py를 수정해보겠습니다.

accounts/views.py

```
from django.shortcuts import render, redirect
from django.contrib.auth.models import User # 장고가 제공해주는 User 모델을 사용하기 위해서 import 해줬습니다.
from django.contrib import auth # 그리고 인증 절차를 위해서도 가져왔습니다.
# Create your views here.


def signup(request):
    if request.method == 'POST': # 앞서 form에서 post 방식으로 요청을 보내왔을 경우, 즉 회원가입 버튼을 눌렀을 경우 이 다음 과정이 실행이 됩니다.
        if request.POST['password1'] == request.POST['password2']: # 비밀번호가 일치하는지 확인합니다.
            user = User.objects.create_user(
                request.POST['username'], password=request.POST['password1']) # 폼에서 받은 아이디와 패스워드를 가진 유저를 생성합니다.
            auth.login(request, user) # 회원가입 후 바로 로그인이 되게 하기 위해 작성해주었습니다.
            return redirect('index') # 회원가입이 완료되면 index 페이지로 보냅니다.
    return render(request, 'accounts/signup.html')
```

지금은 아주 간단한 방식으로 유저를 생성하였습니다. 중복되는 유저 아이디를 걸러주는 등 회원가입을 해야할 때 필수적으로 거쳐야 하는 것들에 대해서는 지금 다루지 않겠습니다. 구글링이나 공식 문서를 통해서 더 자세하게 구현해보세요!

4. login 구현

로그인도 똑같이 post 방식을 이용해서 정보를 보내주어야 합니다. html 페이지는 회원가입과 유사하게 만들면 됩니다.

`templates/accounts/login.html`

```
{% if error %} <!--에러가 나면 에러를 출력해 준다는 간단한 문구입니다. 잠시 후 views.py에서 에러도 구현할 거에요! -->
{{ error }}
{% endif %}

<form method="POST" action="{% url 'login' %}">
  {% csrf_token %}
  <h1>로그인 페이지 입니다.</h1>
    <label for="inputEmail">이메일</label>
    <input type="email" id="inputEmail" name="username" placeholder="이메일을 입력해주세요.">
    <br>
    <label for="inputPassword">비밀번호</label>
    <input type="password" id="inputPassword" name="password" placeholder="비밀번호를 입력해주세요">
    <br>

    <input type="submit" value="로그인하기">
</form>
```

남은건 views.py입니다!

`accounts/views.py`

```
def login(request):
    if request.method == 'POST': # 회원가입과 똑같죠?
        username = request.POST['username']
        password = request.POST['password']
        user = auth.authenticate(request, username=username, password=password) # 로그인 페이지에서 보낸 데이터가 유저들이 들어있는 데이터베이
        if user is not None:
            auth.login(request, user) # 성공했다면 로그인!
            return redirect('index')
        else:
            return render(request, 'accounts/login.html', {'error': 'username or password is incorrect.'}) # 성공하지 못했다면 에러를
    else:
        return render(request, 'accounts/login.html')
```

이제 회원가입과 로그인이 모두 가능한 페이지를 만들었습니다!

5. logout 구현

마지막으로 로그아웃이 남았습니다. 로그아웃을 하려면 유저가 로그인 된 상태인지를 알아야 합니다. 생각만 해도 복잡해지지만 다행스럽게도 장고가 기능을 제공해줍니다! 아래 코드를 이용해서 로그아웃을 구현해보겠습니다.

`templates/accounts/login.html`

```
{% if user.is_authenticated %}
  <form id="logout" method="POST" action="{% url 'logout' %}">
    {% csrf_token %} <input type="hidden" />
  </form>
{% else %}
  <a href="{% url 'login' %}">log in</a>
{% endif %}
```

이해가 되시나요? 유저가 로그인 한 상태에서는 로그아웃 버튼만 보이고 로그인이 되어 있지 않다면 로그인 버튼을 보여주게 해주었습니다. 이제 로그아웃 기능을 만들어 봅시다.

`accounts/views.py`

```
def logout(request):
    if request.method == 'POST':
        auth.logout(request)
        return redirect('index')
    return render(request, 'accounts/signup.html')
```

urls.py도 마저 세팅해줍니다.

`accounts/urls.py`

```
urlpatterns = [
    path('signup/', views.signup, name='signup'),
    path('login/', views.login, name='login'),
    path('logout/', views.logout, name='logout'),
]
```

다 작성했다면 서버를 돌려 정상적으로 작동하는지 확인해주세요! 추가로 `{% if user.is_authenticated %}` 태그를 활용하면 글을 작성한 사람만 수정하게 하거나 로그인을 한 유저에게만 보여주어야 하는 버튼 등을 구현할 수 있습니다!