Day8 Django Auth1 (로그인 로그아웃)

자료	Django
구분	Django
과목	

[시작 스켈레톤 코드는 저번시간에 완성했던 부분 부터 이어시 진행하겠습니다.]

test33_skeleton.zip

HTTP 프로토콜을 이용해서 로그인 하는 과정을 살펴볼 것이다.

(우리는 앞에서 csrf 토큰에 대해서 학습할 때 쿠키와 세션에 대한 공부를 각자 했었다고 가정하겠다.)

쿠키란? - 사이트 방문할때 브라우저에 저장되는 내용들을 말한다.

과정을 살펴보자.
1. 클라이언트가 서버로 요청을 보낸다 (url을 통해서 사이트 접근을 요청한다.)
2. 서버는 웹문서와 함께 [쿠키]를 생성하여 같이 응답한다.
3. 클라이언트가 해당 사이트의 다른 웹페이지를 보기위해 request 요청을 다시 보낼 때면
   요청과 함께 아까 받았던 쿠키도 같이 서버로 전송한다.
4. 그러면 서버는 쿠키를 통해서 해당 클라이언트의 아이디자동완성 / 장바구니 또는
   팝업창 / 오늘하루 안보기 등의 클라이언트 기록 정보를 확인 할 수 있다.

세션이란? - 클라이언트가 아닌!! 서버가 가지고 있는 기록정보 data를 말한다.
-> 비밀번호 또는 카드정보 와 같은 중요한 정보들은 (기업이 책임져야 할 정보들)
클라이언트가 아니라 서버에 저장한다.

1. 클라이언트가 처음으로 로그인을 하면 서버가 session 데이터를 생성 후 저장을 한다.
2. 생성된 세션 데이터에 인증을 할 수 있는 session id 를 발급한다.
3. 이 session id를 클라이언트에게 전달하고
4. 클라이언트는 전달받은 session id를 쿠키에 저장을 한다.
5. 클라이언트의 브라우저에는 서버로부터 받은 쿠키와 session id를 같이 잘 가지고 있다가
   다음부터 서버로 요청을 보낼때 마다 로그인 사실을 입증하는 쿠키와 함께 session id 데이터를
   같이 요청을 보내는 구조다.

그러면 서버에서는 세션을 통해서 클라이언트의 로그인 상태유지를 확인할 수 있고
클라이언트의 로그인 여부에 따라 권한과 인증을 부여할 수 있다.

```
# 쿠키와 세션의 차이점 !

* 쿠키 - 사이트 방문할때 브라우저에 저장되는 내용들
        -> 서버가 아닌!! 클라이언트가 기록정보 data를 가지고 있음
          - 쿠키내용을 내가 수정할 수 있고 남이 훔쳐볼수도 있음
        -> 예시> 로그인정보 또는 장바구니 처럼 보안이 중요하지 않는경우 /
        -> 예시> 팝업창 오늘하루안보기 등

* 세션 - 클라이언트가 아닌!! 서버가 기록정보 data를 가지고 있음
        -> 비번 카드정보 처럼 기업이 책임져야 할 것들은 서버에 저장

    1. 사이트 접속시 서버가 기한이 짧은 "임시키(session id)"를 클라이언트에게 발급한다.
       -> 발급된 임시키가 클라이언트의 브라우저에 저장됨
    2. 그 다음부터 브라우저가 해상 웹사이트를 접속하면 http요청에 (플러스) 그전에
       발급받았던 임시키 까지 더해서 요청메시지 전송 (쿠키+임시키)
    3. 서버에서는 요청이 들어오면 키를 확인 후 해당 클라이언트에게 정보를 제공
       (즉, 세션은 서버가 클라이언트가 누구인지를 식별, 알아보는 수단)
    4. 그러나 세션을 남발하면 접속자가 많을 시 서버에 부하가 걸릴 수 있음
       -> 그래서 '토큰' 방식으로 로그인 하기도 함!
       서버로 부터 토큰을 발급받은 클라이언트가 이를 쿠키로 저장해 두고
       필요할 때마다 서버에 보여줌(마패처럼 보여줌)으로써 서버의 부하를 줄일 수 있음
```

지금까지 우리가 학습한 것을 리마인드 해보자.

유저가 사용할 수 있는 게시판 기능을 구현 했다. (우리는 TodoList를 생성하는 게시판 만들었음)

static file을 통해서 게시판에 css도 적용해서 꾸밀 수 있고

media file을 통해서 유저가 게시판에 이미지 파일도 업로드 할 수 있도록 구현을 해 보았다.

오늘은 '권한'과 '인증'을 통해서 [로그인 로그아웃] 기능을 구현할 것이다.

['인증'과 '권한']

인증은 무엇이고 권한은 무엇일까?

- 인증 - 로그인 했지? 우리 사이트 회원 맞아? 확인하는 절차를 인증이라 하고
- 권한 - 이 게시판에 새 글을 작성할 권한있어? 게시글을 삭제할 권한이 있어? 등 특정 기능을 수행할 때의 권한을 의미한다.

인증 그리고 권한기능을 부여하기 위해서 인증과 권한 관련된 소스코드를 우리가 직접 모든 소스코드를 작성 할 필요가 없다. 이러한 기능들은 장고가 다 제공을 해준다. ("Django Auth")

자, 사전설정이다.

그러나 앞서 우리는 Django에서 제공해주는 form을 실습하기 위해

accounts 앱을 미리 만들어 놓았다.

그런데 만약에 다른 파일을 가지고 본 실습을 하는데 account 앱을 아직 만들지 않았다면

아래 과정을 통해 account 앱을 하나 생성하자.

```
$ python manage.py startapp accounts
```

settings.py 가서 app 추가하기

```
INSTALLED_APPS = [
    'accounts',
```

accounts app 에서 urls.py 파일 만들기

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

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

그 다음에는 모델을 만들어 보자.

기본적으로 Django는 `django.contrib.auth` 모듈에서 `models.User` 라는 모델을 제공해서

회원가입 / 로그인 / 로그아웃 그리고 비밀번호 관리를 해준다.

하지만, 우리는 우리의 서비스에 맞게 유저모델을 커스텀 해서 사용 할 것이다.

그 이유는 Django의 기본 `User` 모델은 `username`을 로그인 ID로 사용하기 때문이다.

하지만

- `username` 대신 이메일을 로그인 ID로 사용할 경우
- 프로필 사진, 닉네임, 전화번호 등의 추가 필드를 저장할 경우
- 인증 방식을 변경할 경우 (예: 소셜 로그인 추가)

에는 Django에서 제공해주는 기본 User 모델을 직접 수정하면 안 되고,

`AbstractUser` 를 상속받아 새로운 필드를 추가 하거나 `AbstractBaseUser` 모듈을 사용해서 완전히 새로운 사용자 모델을 만들 수 있다.

설명만 들어서는 크게 와닫지 않을 수 있다. 직접 코드로 작성 해 보자.

django에서 기본으로 제공해 주는 user모델을 상속 받은 후 커스텀을 진행 할 것이다.

```
# accounts / models.py
```

```
from django.db import models
from django.contrib.auth.models import AbstractUser

class User(AbstractUser):
    pass
```

Django에서 제공해 주는 User Model 를 커스터마이즈 해서 개발자가 원하는 User 모델을 만들어서 사용 할 것이라고 했다. 이때 settings.py 들어가서 장고에게 우리는 유저 모델을 커스텀 해서 사용할 것이라고 알려줘야 한다.

settings.py들어가서 추가하자.

settings.py 파일 최하단에 적어 보겠다.

```
AUTH_USER_MODEL = 'accounts.User'
```

[앱이름.클래스명]으로 적어 주었다.

그 다음 관리자 페이지에서도 User모델을 조작 할 수 있도록

admin.py에 커스텀 user model을 등록하자.

accounts / admin.py

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from .models import User

admin.site.register(User,UserAdmin)
```

참고로 User 모델의 상속 관계를 보면 다음과 같다.

1. accounts 앱에서 models.py에 정의한 User class는 장고에서 제공해 주는 auth 모듈의 AbstracUser라는 클래스를 상속 받았다. 참고로 AbstractUser 클래스는 관리자의 권한으로 User model을 컨트롤 할 때 사용하는 클래스다.

2. AbstracUser라는 클래스는 ctrl+click 을 해서 확인해 보면 AbstracBaseUser라는 클래스에서 코드를 상속받은 클래스 이며

3. AbstracBaseUser라는 클래스도 ctrl+click 을 해서 확인해 보면 결국 models.Model 클래스에서 상속받아 사용한 것이다.

즉. 클래스 상속의 계보를 따져 보자면

models.Model → class AbstractBaseUser → class AbstractUser → class User

가 된다.

이제 중간 정리를 한번 보자. 우리가 지금 한 것은

1. models.py에서 User 모델 클래스를 하나 정의하고 settings.py에 알려 줬다.
2. 그리고 admin사이트에서 회원정보를 관리 할 수 있도록 admin.py 로 이동해서 User 모델을 등록했다.

[중요] Database 초기화

프로젝트를 진행하는데 있어서 서비스 안에 회원가입 시스템이 필요하다면,

프로젝트를 시작할 때 가장 첫번째 작업으로

방금 위에서 했던 과정을 가장 먼저 진행 하는것을 추천 한다.

Django 공식 문서에서도 새 프로젝트를 시작하는 경우 커스텀 User 모델을

가장 먼저 설정하는 것을 강력하게 권장하고 있다.

만약에 프로젝트 중간에 AUTH_USER_MODEL을 정의 또는 변경하게 되면

DB의 스키마가 꼬이거나 충돌을 일이킬 가능성이 많기 때문이다.

따라서 첫 migrations 혹은 첫 migrate를 실행하기 전에 커스텀 유저모델을 먼저

정의를 하는것이 좋겠다.

==========

하지만 지금과 같이 프로젝트 중간에 AUTH_USER_MODEL을 구현했다면

[데이터 베이스 초기화] 반드시 진행 후 다시 migration을 진행해야 한다는 것을

잊지 말자.

데이터 베이스 초기화는 다음과 같다.

1. 각각의 앱마다 migrations 폴더안에 파일을 지우는데 __init__ 파일을 제외하고 숫자가 적혀있는 파일들을 모두 지운다.

2. db.qulite3 삭제한다.

3. migrations / migrate 를 새로 진행한다.

그런다음 DB를 확인해 보면 그전에 보이던 Auth.User 파일은 보이지 않을 것이며

대신 accounts_user 가 새로 보일 것이다.

4. 그리고 admin 사이트를 사용하기 위한 admin 계정도 새로 만들자.

위 과정을 마쳤다면, 다시 돌아와서 로그인 로그아웃 구현을 이어서 해 볼 것이다.

그런데 이번에는 클라이언트가 서버에 로그인요청을 보내고 처리하기 까지의 과정을 고민해 보자.

1. 가장 먼저 사용자가 아이디랑 비밀번호를 작성할 form이 필요 할 것이다.
2. 그리고 만약에 사용자가 자신의 아이디와 비밀번호를 작성후 전송 버튼을 눌렀다면
    a. (서버에 요청이 들어 왔다면)
3. 서버에서는 사용자가 입력한 정보가 DB에 있는 정보랑 일치하는지 확인 및 유효성 검사를 해야 할 것이고
4. 정보가 유효하다면 해당 정보를 바탕으로 세션을 생성 후 세션테이블에 저장을 할 것이다.
5. 그리고 세션키랑 쿠키를 클라이언트에게 응답을 해줘야 할 것이다.

위 과정들을 우리가 로그인 처리하기 위핸 위 과정들을 직접 구현할 필요가 없다.
Django가 대신 다 해줄 것이다.

urls.py

```
# accounts/urls.py

from django.urls import path
from . import views

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

views.py

```
from django.shortcuts import render
from .forms import LoginForm

def login(request):
    form = LoginForm()
    context = {
        'form': form
```

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

Django Form 실습을 위해서 우리는 이미 위의 코드를 작성 했었다.

우리가 forms.py에서 LoginForm 클래스를 먼저 정의를 해주고

views.py의 login 함수에서 미리 정의 해 둔 form을 가져다가 썼었다.

그러나 앞으로는 로그인 작업은 Django가 제공해 주는 Built in form을 이용할 것이다.
django.contrib.auth.forms 모듈 안에 있는 AuthenticationForm 을 사용하면

로그인 기능을 빠르게 구현이 가능 할 뿐만 아니라 보안, 유효성검사, 세션관리를 해주는 코드가

이미 다 정의가 되어 있기 때문이다.

우선 AuthenticationForm을 먼저 import 하자.

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

views.py

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

def login(request):
    if request.method=='POST':          # 2.POST - 로그인 처리해줌
        pass
    else:                               # 1.GET - 로그인 페이지 제공
        form=AuthenticationForm()
    context={
        'form':form
    }
    return render(request,'accounts/login.html',context)
```

accounts / templates / accounts 폴더를 생성후 login.html 파일 하나 만들자

사용자가 로그인 할때 사용할 html 문서를 하나 만들 것이다.

```
{% extends 'base.html' %} {% block content %}
<h1>로그인 페이지</h1>

<form action="{% url 'accounts:login' %}" method="POST">
 {% csrf_token %}
 {{form.as_p}}
 <input type="submit" value="로그인">
```

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

그리고 이제 사용자가 올바른 ID 그리고 PASSWORD를 기입했을 때
로그인 처리를 해주는 코드를 완성해 보자.

로그인 처리시 고려해야 할 점이 두 세가지 정도가 있다고 앞에서 언급했다.

1. 사용자가 입력한 정보가 유효하다면 서버에서 세션테이블에 세션을 만들고
2. 사용자 정보를 세션에 저장한다. 그리고 쿠키에 세션키를 담아서 응답을 한다.
3. 그러면 사용자는 이후 요청에 인증된 상태로 유지된다.

위 과정을 다 직접 구현 할 필요 없이 "사용자 인증에 관련된 모듈" 에 있는 (django.contrib.auth)
login 이라는 함수를 사용 할 것이다.

login 은 우리가 views.py에 정의할 함수 이름과 같아서 as를(별칭) 이용해서
auth_login 이라는 별칭을 하나 붙여 줄 것이다.
`from django.contrib.auth import login as auth_login`

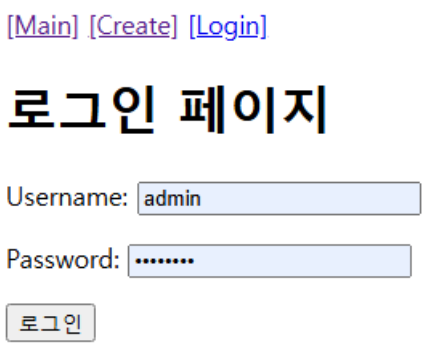
views.py

```
from django.shortcuts import render, redirect
from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth import login as auth_login

def login(request):
    if request.method=='POST':
        form=AuthenticationForm(request, request.POST)
        if form.is_valid():
            auth_login(request, form.get_user())
            return redirect('todos:index')

            # auth_login()을 사용해서 로그인 처리를 하는데
            # 첫번째 매개변수로 request를 받고
            # 두번째 매개변수로는 가져온 form에서 get_user() 라는 메서드를 사용해서
            # 유저 정보만을 가져올 것이다.
                # 그 다음에 index 페이지로 redirect 할 것이다.

    else:
        form=AuthenticationForm()

    context={
```

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

`form.get_user()`는 Django의 `AuthenticationForm` 클래스에 내장된 메서드인데
해당 사용자의 User 인스턴스를 반환 해주는 메서드 이다.

서버켜서 http://127.0.0.1:8000/accounts/login/ 들어간 후에
로그인페이지가 잘 나오는지 확인해 보자.



이번에는 html 파일에서 로그인된 User 정보를 사용해보자.
todos app 에서 templates안에 index.html 파일에 웰컴 인사를 넣어보자.

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

{% block content %}

  <h1>할 일 목록 페이지</h1>
  <h1>{{ user }} 님의 할 일 목록 페이지</h1>

  ... 생략
```

다시 브라우저를 새로고침 후 확인해 보자.

[참고로 한번 읽고 넘어 가도 될 부분]

{{ user }} 를 todos 앱의 template에 작성했더니 어떻게 로그인 한 유저의 ID가 나오지?

로그인한 "유저의 ID"를 user라는 변수로 사용할 수 있는 이유가 뭘까?

user라는 변수로 로그인 한 유저의 ID를 확인할 수 있는 이유는

settings.py에 context processor 설정값 때문에 가능한 것이다.

settings.py에 TEMPATES > OPTIONS 에 있는 context processor 이 부분은 템플릿이 렌더링 될 때

호출 가능한 컨텍스트 데이터 목록을 명시해 주는 부분이다.

명시된 컨텍스트 데이터는 기본적으로 템플릿에서 변수로 활용이 가능하다.

안에 들어가 보면

'django.contrib.auth.context_processors.auth', 라고 있는 코드가 있는데

이 코드로 인해서 로그인 한 user 의 이름을 변수처럼 사용이 가능한 것이다.

이번엔 로그아웃 해보자.

간단하다. 로그아웃은 무엇을 하는 것인가?

바로 서버에 있는 세션 데이터를 삭제하고 그리고

클라이언트의 쿠키 안에 있는 세션을 지우는 과정이다.

urls.py

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

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

views.py

```
from django.contrib.auth import logout as auth_logout

def logout(request):
```

```
    auth_logout(request)
    return redirect('accounts:login')
```

auth_logout(request) 이 한줄로 인해서

request 요청에 있는 쿠키를 열어서 세션아이디가 있으면 그것을 꺼내고,

우리 DB에 있는 세션테이블과 비교해서 세션아이디가 있으면 그것을 지워줘! 가

모두 실행되는 것이다.

templates안에 base.html 파일에 로그아웃 url을 하나 넣어보자.

base.html

```
<body>
  <a href="{% url 'todos:index' %}">[Main]</a>
  <a href="{% url 'todos:create_todo' %}">[Create]</a>
  <a href="{% url 'accounts:login' %}">[Login]</a>
  <a href="{% url 'accounts:logout' %}">[Logout]</a>
```

로그인 그리고 로그아웃이 잘 작동 되는지 서버켜서 확인하자.

수고 많았습니다. <끝>

test33_completed.zip