Django Auth1

자료	Django
구분	Django

HTTP 프로토콜을 이용해서 로그인 하는 과정을 살펴보자.

쿠키 그리고 세션이 무엇인지 먼저 살펴 보아야 한다.

```
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 같이 응답한다.
3. 클라이언트가 해당 사이트의 다른 웹페이지를 보기위해 request 요청을 다시 보낼 때면
   요청과 함께 쿠키를 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.
          -> 비번 카드정보 처럼 기업이 책임져야 할 것들은
             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
          -> 로그인정보 또는 장바구니 처럼 보안이 중요하지 않는경우 / 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 보여줌으로써 서버의 부하를 줄일 수 있음
```

지금까지 게시판 만드는 것을 해 보았고

static file을 통해서 게시판에 css도 넣어 보았고

media file을 통해서 게시판에 이미지도 올려 보았다.

오늘은 ‘권한’과 ‘인증’을 통해서 [로그인 로그아웃] 기능을 구현해 보겠다.

실습을 시작할 파일이다

09-django-authentication-system.zip

오늘 공부할 내용을 보자.

1. 인증과 권한
2. Custom User Model
3. 쿠키와 세션
4. Django Auth System 순서로 보자

1. 인증과 권한

Django 에서 Auth 라고 하면 인증 그리고 권한 둘다 이야기를 한다. 인증은 무엇이고 권한은 무엇일까?

인증 - 로그인 했지? 우리 사이트 회원 맞아? 확인하는 절차를 인증이라 하고

권한 - 게시판에 글 쓸 권한있어? 게시글을 삭제할 권한이 있어? 등 기능을 작동시킬 때의 권한을 의미한다.

이러한 기능들도 장고가 다 제공을 해준다. (장고 만세)

사전설정

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

settings.py 가서 app 추가하기

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

accounts app 에서 urls.py 파일 만들기

```
from django.urls import path


urlpatterns = [

]
```

그다음에는 모델을 만들어 보자.

Django가 기본 User Model을 제공 해주지만 우리는 우리 서비스에 맞도록 Custom User Model 을 사용 해 볼 것이다. 예를 하나 들어 보자면 장고에서 기본으로 제공해 주는 User모델에서는 username을 User ID 식별값으로 갖는다. 하지만 우리 서비스에서는 username이 필요 없고 username 대신 유저의 e-mail을 ID로 대체하고 싶을 수도 있을 것이다. 그럴 경우에는 커스터마이징이 필요 하겠다.

accounts / models.py

django에서 기본으로 제공해 주는 user모델을 상속 받은 후 커스텀을 진행 해 보자

```
from django.db import models
from django.contrib.auth.models import AbstractUser
# Create your models here.

class User(AbstractUser):
    pass
```

Django에서 제공해 주는 User Model을 커스터 마이징 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.

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

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)
```

[참고] User 모델의 상속 관계를 보면 다음과 같다.

```
1. accounts 앱에서 models.py에 등록한 user class는  장고에서 제공해 주는 auth 모델의
  AbstracUser라는 클래스를 상속 받았다. 참고로 AbstractUser 클래스는 관리자 권한으로
  User model을 컨트롤 할 때 사용하는 클래스 이다.

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

3. AbstracBaseUser라는 클래스도 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 유저모델 하나 정의하고 settings.py에 알려 줬다.
2. 그리고 admin사이트에서 회원정보를 관리 할 수 있도록 admin 페이지에 user 모델을 등록했다.

[중요] Database 초기화

```
프로젝트를 맨 처음 진행할 때 회원가입 시스템이 필요하다면, 프로젝트 처음에

방금 위의 1, 2 과정을 먼저 진행 하는것을 추천 한다.

User model에 필드를 추가 하는것은 나중에 해도 상관이 없지만 프로젝트 구현 초반에

방금 위의 1, 2 과정을 먼저 진행 해야 한다.

Django 공식 문서에서도 새 프로젝트를 시작하는 경우 커스텀 User 모델을 먼저 설정하는 것을

강력하게 권장하고 있다. 즉 migrations 혹은 첫 migrate를 실행하기 전에 위 작업을 실시해야 한다.

그렇지 않고 프로젝트 중간에 AUTH_USER_MODEL을 정의 또는 변경하게 되면

DB의 스키마가 꼬이게 된다. 만약에 지금 우리가 실습하고 있는 파일과 같이

AUTH_USER_MODEL을 구현하지 않고 다른 앱을(accounts) 먼저 구현하였다면

[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 다 지운다.
```

```
2. db.qulite3  삭제한다.
3. migration을 새로 진행하라.

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

accounts.user 가 새로 보일 것이다. 그리고 admin 계정도 새로 만들자.
```

자 다시 로그인 로그아웃 구현을 해보자.

로그인을 처리하기 위한 프로세스 부터 고민해 보자. 가장 먼저 사용자가 아이디랑 비밀번호를 작성할 form이 필요 할 것이다. 그리고 만약에 사용자가 자신의 아이디와 비밀번호를 작성후 전송 버튼을 눌렀다면 (서버에 요청이 들어 왔다면)

서버에서는

1. 사용자가 입력한 정보가 DB에 있는 정보랑 일치하는지 확인 및 유효성 검사를 해야 할 것이고
2. 정보가 유효하다면 해당 정보를 바탕으로 세션을 생성 후 세션테이블에 저장을 할 것이다.

   그리고 세션키랑 쿠키를 클라이언트에게 응답을 해줘야 할 것이다.

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

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

def login(request):
```

```
    if request.method=='POST':
        # 로그인 처리해줌
        pass
    else:
        # 비어있는 로그인 페이지 제공
        pass
```

코드를 마저 완성시켜 보자

그전에 게시판을 만드는 것을 학습할 때에는 특정 게시물 페이지 제공할 경우 템플릿을 만든 후

템플릿 안에 form을 만드는 작업을 직접 다 구현을 했다면

로그인 작업은 Django가 제공해 주는 Built in form을 이용할 것이다.

django가 forms.py에 정의해야 할 것들을 이미 다 가지고 있다. 어디에?

django.contrib.auth.forms 안에 있는 AuthenticationForm 을 가져다 쓸 것이다.

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

def login(request):
    if request.method=='POST':
        # 로그인 처리해줌
        pass
    else:
        # 비어있는 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.

```
<h1>login</h1>

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

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

그리고 이제 로그인에 성공시 로그인 처리를 해주는 코드를 완성해 보자.

로그인 처리시 고려해야 할 점을이 두세가지 있다고 앞에서 언급했다.

1. 입력한 아이디 비번이 유효한지?
2. 유효하다면 세션테이블에 세션 만들고
3. 쿠키에 세션키 담아서 응답하기

위 과정을 다 직접 구현 할 필요 없이 django.contrib.auth 안에 login 이라는 것을 가져다 쓸 것이다.

login 은 함수 이름과 같아서 as를 이용해서 auth_login 이라는 별칭을 하나 붙여 줄 것이다.

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':
        # 유저가 입력한 정보가 채워진 form
        form=AuthenticationForm(request, request.POST)

        # form 안의 정보가 유효 하다면
        if form.is_valid():

            auth_login(request, form.get_user())
            return redirect('articles:index')

            # auth_login을 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:
        # 비어있는 로그인 페이지 제공
        form=AuthenticationForm()
    context={
        'form':form
    }
    return render(request,'accounts/login.html',context)
```

서버켜서 http://127.0.0.1:8000/accounts/login/ 들어간 후에

로그인 되는지 확인해 보자. 잘 될것이다.

마지막으로 템플릿에서 user를 사용해보자.

articles app 에서 templates안에 index.html 파일에 웰컴 인사를 넣어보자.

그리고 편의상 로그인 링크도 하나 추가하자.

```
<body>

  <h1>{{user}}님 반갑습니다.</h1>
  <h1>INDEX</h1>

  <a href="{% url 'articles:create' %}">CREATE</a>
  <a href="{% url 'accounts:login' %}">로그인</a>


  ... 생략
```

다시 새로고침 후 확인해 보자.

[참고] 어떻게 로그인한 유저의 ID를 변수로 사용할 수가 있지?

```
로그인 한 유저의 ID도 확인할 수 있다. 이는 settings.py에

context processor 설정값 때문에 가능한 것이다.

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 있다.

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

이번엔 로그아웃 해보자.

간단하다. 로그아웃은 무엇하는것인가?

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('articles:index')
```

auth_logout(request) 이것 한줄로 인해서

request 요청에 있는 쿠키를 열어서 세션아이디가 있으면 그것을 꺼내고,

우리 DB에 있는 세션테이블과 비교해서 세션아이디가 있으면 그것을 지워줘! 가 모두 실행되는 것이다.

articles app 에서 templates안에 index.html 파일에 로그아웃 url을 하나 넣어보자

index.html

```
<h1>{{user}}님 반갑습니다.</h1>
  <h1>INDEX</h1>

  <a href="{% url 'articles:create' %}">CREATE</a>
```

```
  <a href="{% url 'accounts:login' %}">로그인</a>
  <a href="{% url 'accounts:logout' %}">로그아웃</a>
```

http://127.0.0.1:8000/articles/ 들어가서 로그인 그리고 로그아웃이 잘 작동 되는지 확인하자

수고 많았습니다. <끝>