

220907 Django 5

1. The Django authentication system

- Django authentication system(인증 시스템)은 인증(Authentication)과 권한(Authorization) 부여를 함께 제공(처리)하며, 이러한 기능을 일반적으로 인증 시스템이라고 함
- Authentication(인증)
 - 신원 확인
 - 사용자가 자신이 누구인지 확인하는 것
- Authorization(권한, 허가)
 - 권한 부여
 - 인증된 사용자가 수행할 수 있는 작업을 결정

1.1 Substituting a custom User model

- Custom User 모델로 대체하기
- 기본 User Model을 필수적으로 Custom User model로 대체하는 이유 이해하기
- Django는 기본적인 인증 시스템과 여러 가지 필드가 포함된 User Model을 제공, 대부분의 개발 환경에서 기본 User Model을 Custom User Model로 대체함
- 개발자들이 작성하는 일부 프로젝트에서는 django에서 제공하는 built-in-User model의 기본 인증 요구사항이 적절하지 않을 수 있음
- Django는 현재 프로젝트에서 사용할 User Model을 결정하는 AUTH_USER_MODEL 설정 값으로 Default User Model을 재정의 override(override)할 수 있도록 함

1.1.1 AUTH_USER_MODEL

- 프로젝트에서 User를 나타낼 때 사용하는 모델
- 프로젝트가 진행되는 동안 (모델을 만들고 마이그레이션 한 후) 변경할 수 없음
- 프로젝트 시작 시 설정하기 위한 것이며, 참조하는 모델은 첫 번째 마이그레이션에서 사용할 수 있어야 함
 - 즉, 첫번째 마이그레이션 전에 확정 지어야 하는 값

- 다음과 같은 기본값을 가지고 있음

```
# settings.py
AUTH_USER_MODEL = 'auth.User'
```

1.2 How to substituting a custom User model

1.2.1 Custom User model로 대체하기

- AbstractUser를 상속받는 커스텀 User 클래스 작성
 - 기존 User 클래스도 AbstractUser를 상속받기 때문에 커스텀 User 클래스도 완전히 같은 모습을 가짐

```
# accounts/models.py
from django.contrib.auth.models import AbstractUser

class User(AbstractUser):
    pass
```

- Django 프로젝트에서 User를 나타내는데 사용하는 모델을 방금 생성한 커스텀 User 모델로 저장

```
# settings.py

AUTH_USER_MODEL = 'accounts.User'
```

- admin.py에 커스텀 User 모델을 등록
 - 기본 User 모델이 아니기 때문에 등록하지 않으면 admin site에 출력되지 않음
 - `admin.site.register` 시험!!!

```
# accounts/admin.py

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

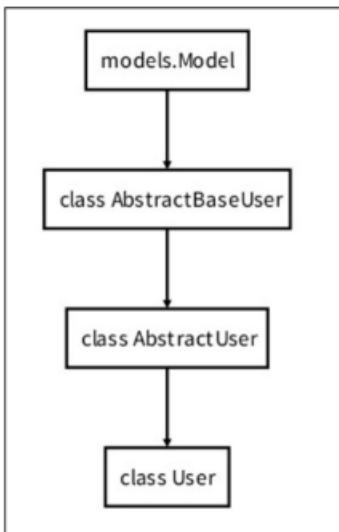
admin.site.register(User, UserAdmin)
```

1.2.2 데이터베이스 초기화

- 수업 진행을 위한 데이터베이스 초기화 후 마이그레이션(프로젝트 중간일 경우)
- migrations 파일 삭제
 - migrations 폴더 및 __init__.py는 삭제하지 않음
 - 번호가 붙은 파일만 삭제
- db.sqlite3 삭제
- migrate 실행
 - makemigrations
 - migrate

[참고]

- User 모델 상속 관계



- **AbstractUser**
 - 관리자 권한과 함께 완전한 기능을 가지고 있는 User model을 구현하는 추상 기본 클래스
 - Abstract base classes (추상 기본 클래스)
 - 몇 가지 공통 정보를 여러 다른 모델에 넣을 때 사용하는 클래스
 - 데이터베이스 테이블을 만드는데 사용되지 않으며, 대신 다른 모델의 기본 클래스로 사용되는 경우 해당 필드가 하위 클래스의 필드에 추가됨
- 프로젝트 중간에 AUTH_USER_MODEL 변경하기
 - 모델 관계에 영향을 미치기 때문에 훨씬 더 어려운 작업이 필요하여 권장하지 않음
 - 프로젝트 처음에 진행하기

- 반드시 User 모델을 대체해야 할까? 시험!!

- Django는 새 프로젝트를 시작하는 경우 비록 기본 User 모델이 충분 하더라도 커스텀 User 모델을 설정하는 것을 강력하게 권장(highly recommended)
- 커스텀 User 모델은 기본 User 모델과 동일하게 작동 하면서도 필요한 경우 나중에 맞춤 설정할 수 있기 때문
 - 단, User 모델 대체 작업은 프로젝트의 모든 migrations 혹은 첫 migrate를 실행하기 전에 이 작업을 마쳐야 함

2. HTTP Cookies

2.1 HTTP

- Hyper Text Transfer Protocol
- HTML 문서와 같은 리소스들을 가져올 수 있도록 해주는 프로토콜(규칙, 규약)
- 웹(WWW)에서 이루어지는 모든 데이터 교환의 기초
- 클라이언트-서버 프로토콜이라고도 부름

2.1.1 요청과 응답

- 요청 (request)
 - 클라이언트(브라우저)에 의해 전송되는 메시지
- 응답 (response)
 - 서버에서 응답으로 전송되는 메시지

2.1.2 HTTP 특징

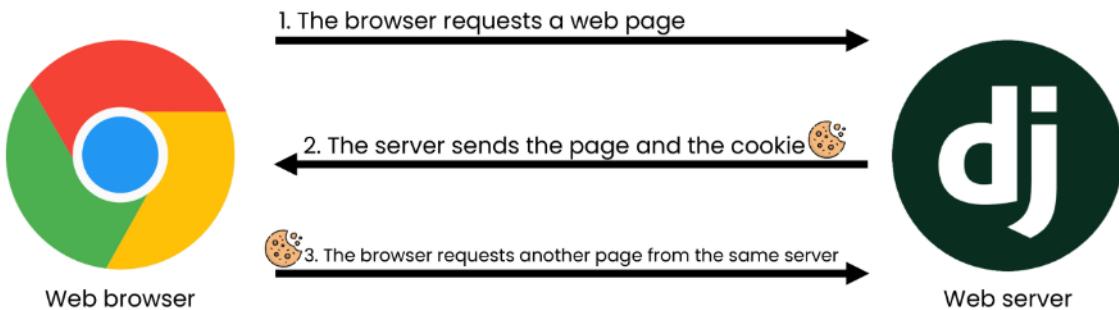
- 비 연결 지향 (connectionless)
 - 서버는 요청에 대한 응답을 보낸 후 연결을 끊음
- 무상태 (stateless)
 - 연결을 끊는 순간 클라이언트와 서버 간의 통신이 끝나며 상태 정보가 유지 X
 - 클라이언트와 서버가 주고받는 메시지들은 서로 완전히 독립적

2.1.3 어떻게 로그인 상태를 유지할까?

- 서버와 클라이언트 간 지속적인 상태 유지를 위해 "쿠키와 세션" 이 존재

2.2 쿠키 (Cookie)

- HTTP 쿠키는 상태가 있는 세션을 만들도록 해줌
- 서버가 사용자의 웹 브라우저에 전송하는 작은 데이터 조각
- 사용자가 웹사이트에 방문할 경우 해당 웹사이트의 서버를 통해 사용자의 컴퓨터에 설치되는 작은 기록 정보 파일
 - 브라우저(클라이언트)는 쿠키를 로컬에 KEY-VALUE의 데이터 형식으로 저장
 - 쿠키를 저장해 놓았다가, 동일한 서버에 재요청 시 저장된 쿠키를 함께 전송
- 쿠키는 두 요청이 동일한 브라우저에서 들어왔는지 아닌지를 판단할 때 주로 사용됨
 - 이를 이용해 사용자의 로그인 상태를 유지할 수 있음
 - 상태가 없는(stateless) HTTP 프로토콜에서 상태 정보를 기억 시켜 주기 때문
- 즉, 웹페이지에 접속하면 웹 페이지를 응답한 서버로부터 쿠키를 받아 브라우저에 저장하고, 클라이언트가 같은 서버에 재요청 시마다 요청과 함께 저장해 두었던 쿠키도 함께 전송



2.2.1 쿠키 사용 목적

- 세션 관리 (Session management)
 - 로그인, 아이디 자동완성, 공지 하루 안 보기, 팝업 체크, 장바구니 등의 정보 관리
- 개인화 (Personalization)
 - 사용자 선호, 테마 등의 설정
- 트래킹 (Tracking)
 - 사용자 행동을 기록 및 분석

2.2.2 세션

- 사이트와 특정 브라우저 사이의 state(상태)를 유지시키는 것
- 클라이언트가 서버에 접속하면 서버가 특정 session id를 발급하고, 클라이언트는 session id를 쿠키에 저장
 - 클라이언트가 다시 동일한 서버에 접속하면 요청과 함께 쿠키(session id가 저장된)를 서버에 전달
 - 쿠키는 요청 때마다 서버에 함께 전송 되므로 서버에서 session id를 확인해 알맞은 로직을 처리
- session id는 세션을 구별하기 위해 필요하며, 쿠키에는 session id만 저장

2.2.3 쿠키 Lifetime(수명)

- Session cookie
 - 현재 세션이 종료되면 삭제됨
 - 브라우저 종료와 함께 세션이 삭제됨
- Persistent cookies
 - Expires 속성에 지정된 날짜 혹은 Max-Age 속성에 지정된 기간이 지나면 삭제됨

2.2.4 Session id Django

- Django는 database-backed sessions 저장 방식을 기본 값으로 사용
 - session 정보는 Django DB의 django_session 테이블에 저장
 - 설정을 통해 다른 저장방식으로 변경 가능
- Django는 특정 session id를 포함하는 쿠키를 사용해서 각각의 브라우저와 사이트가 연결된 session을 알아냄

3. Authentication in Web requests

3.1 Login

- 로그인은 `Session` 을 `Create` 하는 과정
- `AuthenticationForm`
 - 로그인을 위한 built-in form
 - 로그인하고자 하는 사용자 정보를 입력 받음
 - 기본적으로 username과 password를 받아 데이터가 유효한지 검증
 - request를 첫번째 인자로 취함

3.1.1 로그인 페이지 작성

```
# accounts/urls.py

from django.urls import path
from . import views

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

<!-- accounts/login.html -->
{% extends 'base.html' %}

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

```
# accounts/views.py

from django.contrib.auth.forms import AuthenticationForm

def login(request):
    if request.method == 'POST':
        pass
    else:
        form = AuthenticationForm()
    context = {
        'form': form
    }
    return render(request, 'accounts/login.html', context)
```

59

- `login()`
 - `login(request, user, backend=None)`
 - 인증된 사용자를 로그인 시키는 로직으로 view 함수에서 사용됨
 - 현재 세션에 연결하려는 인증 된 사용자가 있는 경우 사용
 - HttpRequest 객체와 User 객체가 필요

3.1.2 로그인 로직 작성

- view 함수 login과의 충돌을 방지하기 위해 import함 login 함수 이름을 auth_login 으로 변경해서 사용

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

def login(request):
    if request.method == 'POST':
        form = AuthenticationForm(request, request.POST)
        # form = AuthenticationForm(request, data=request.POST)
        if form.is_valid():
            auth_login(request, form.get_user())
            return redirect('articles:index')
    else:
        form = AuthenticationForm()
    context = {
        'form': form
    }
    return render(request, 'accounts/login.html', context)
```

- `get_user()`
 - AuthenticationForm의 인스턴스 메서드
 - 유효성 검사를 통과했을 경우 로그인 한 사용자 객체를 반환

3.1.3 Authentication with User

- 현재 로그인 되어있는 유저 정보 출력하기

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

<body>
    <div class="container">
        <h3>Hello, {{ user }}</h3>
        <a href="{% url 'accounts:login' %}">Login</a>
        <hr>
        {% block content %}
        {% endblock content %}
    </div>
    ...
</body>
</html>
```

- `context processors`

- 템플릿이 렌더링 될 때 호출 가능한 컨텍스트 데이터 목록
- 작성된 컨텍스트 데이터는 기본적으로 템플릿에서 사용 가능한 변수로 포함됨
- django에서 자주 사용하는 데이터 목록을 미리 템플릿에 로드 해 둔 것

```
# settings.py

TEMPLATES = [
    {
        ...
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

- `django.contrib.auth.context_processors.auth`
 - 현재 로그인한 사용자를 나타내는 User 클래스의 인스턴스가 템플릿 변수 `{{ user }}`에 저장
 - 클라이언트 로그인하지 않은 경우 AnonymousUser 클래스의 인스턴스로 생성

3.2 Logout

- 로그아웃은 Sessions 을 Delete 하는 과정
- `logout(request)`
- HttpRequest 객체를 인자로 받고 반환 값이 없음
- 사용자가 로그인하지 않은 경우 오류를 발생시키지 않음
 - 현재 요청에 대한 session data를 DB에서 삭제
 - 클라이언트의 쿠키에서도 sessionid를 삭제
 - 이는 다른 사람이 동일한 웹 브라우저를 사용하여 로그인하고, 이전 사용자의 세션 데이터에 액세스하는 것을 방지하기 위함

3.2.1 로그아웃 로직 작성

The diagram illustrates the flow of logic from URLs to views and finally to the template. On the left, `accounts/urls.py` defines two paths: one for login and one for logout. On the right, `accounts/views.py` contains a `logout` function that uses Django's built-in `auth_logout` function and then redirects to the 'articles:index' view. In the center, `base.html` shows the HTML structure for a user profile page. It includes a `Logout` link (`a href="{% url 'accounts:logout' %}"`) and a form for logging out. The form's action is set to the URL for `accounts:logout`, it uses POST method, and it includes a CSRF token input.

```
# accounts/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'),
]

# accounts/views.py
from django.contrib.auth import logout as auth_logout

def logout(request):
    auth_logout(request)
    return redirect('articles:index')


<!-- base.html -->
<body>
<div class="container">
<h3>Hello, {{ user }}</h3>
<a href="{% url 'accounts:login' %}">Login</a>
<form action="{% url 'accounts:logout' %}" method="POST">
    {% csrf_token %}
    <input type="submit" value="Logout">
</form>
<hr>
{%
    block content
    {%
        endblock content
    %}
</div>
</body>
```

4. Authentication with User

- User Object와 User CRUD에 대한 이해
 - 회원가입, 회원탈퇴, 회원정보 수정, 비밀번호 변경

4.1 회원가입

- 회원가입은 User를 Create하는 것이고, UserCreatingForm 을 사용
- `UserCreationForm`
 - 주어진 username과 password로 권한이 없는 새 user를 생성하는 ModelForm
 - 3개의 필드를 가짐
 - username (from the user model)
 - password1
 - password2

4.1.1 회원가입 페이지 작성

```
# accounts/urls.py
app_name = 'accounts'
urlpatterns = [
    ...,
    path('signup/', views.signup, name='signup'),
```

```
<!-- accounts/signup.html -->
{% extends 'base.html' %}

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

```
# accounts/views.py
from django.contrib.auth.forms import AuthenticationForm, UserCreationForm

def signup(request):
    if request.method == 'POST':
        pass
    else:
        form = UserCreationForm()
    context = {
        'form': form,
    }
    return render(request, 'accounts/signup.html', context)
```

84

4.1.2. 회원가입 로직 작성

```
# accounts/views.py

def signup(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('articles:index')
    else:
        form = UserCreationForm()
    context = {
        'form': form,
    }
    return render(request, 'accounts/signup.html', context)
```

- 회원가입에 사용하는 UserCreationForm이 우리가 대체한 커스텀 유저 모델이 아닌 기존 유저 모델로 인해 작성된 클래스라서 오류 발생

4.2 Custom user & Built-in auth forms

- 아래 Form 클래스는 User 모델을 대체하더라도 커스텀 하지 않고 사용 가능
 - AuthenticationForm
 - SetPasswordForm
 - PasswordChangeForm
 - AdminPasswordChangeForm
- 기존 User 모델을 참조하는 Form이 아니기 때문에

- 커스텀 유저 모델을 사용하려면 다시 작성하거나 확장해야 하는 forms
 - `UserCreationForm`
 - `UserChangeForm`
 - 두 form 모두 class Meta: model = User가 등록된 form이기 때문에 반드시 커스텀(확장)해야 함

4.2.1 UserCreationForm() 커스텀 하기

```
# accounts/forms.py

from django.contrib.auth import get_user_model
from django.contrib.auth.forms import UserCreationForm, UserChangeForm

class CustomUserCreationForm(UserCreationForm):
    class Meta(UserCreationForm.Meta):
        model = get_user_model()

class CustomUserChangeForm(UserChangeForm):
    class Meta(UserChangeForm.Meta):
        model = get_user_model()
```

- `get_user_model()`
 - 현재 프로젝트에서 활성화 된 사용자 모델(active user model)을 반환
 - 직접 참조하지 않는 이유
 - 예를 들어 기존 User 모델이 아닌 User 모델을 커스텀 한 상황에서는 커스텀 User 모델을 자동으로 반환해주기 때문

4.2.2 CustomUserCreatingForm() 으로 대체하기

```
# accounts/views.py

from django.contrib.auth.forms import AuthenticationForm, UserCreationForm
from .forms import CustomUserCreationForm, CustomUserChangeForm

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

4.2.3 로그인 진행

```
# accounts/views.py

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

- `UserCreationForm` 의 `save` 메서드

- `user`를 반환하는 것을 확인

```
def save(self, commit=True):
    user = super().save(commit=True)
    user.set_password(self.cleaned_data["password1"])
    if commit:
        user.save()
    return user
```

4.3 회원탈퇴

- 회원 탈퇴하는 것은 DB에서 유저를 Delete하는 것과 같음

4.3.1 회원 탈퇴 로직 작성

```
# accounts/urls.py

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

```
# accounts/views.py

def delete(request):
    request.user.delete()
    return redirect('articles:index')
```

```
<!-- base.html -->
...
<h3>Hello, {{ user }}</h3>
<form action="{% url 'accounts:delete' %}" method="POST">
    {% csrf_token %}
    <input type="submit" value="회원탈퇴">
</form>
```

- 탈퇴하면서 해당 유저의 세션 정보도 함께 지우고 싶은 경우

- `탈퇴 → 로그아웃` 의 순서가 바뀌면 안됨

- 먼저 로그아웃을 해버리면 해당 요청 객체 정보가 없어지기 때문에 탈퇴에 필요한 정보 또한 없어짐

```
# accounts/views.py

def delete(request):
    request.user.delete()
    auth_logout(request)
```

4.4 회원 정보 수정

- 회원 정보 수정은 User를 Update하는 것이며 UserChangeForm을 사용
- **UserChangeForm**
 - 사용자의 정보 및 권한을 변경하기 위해 admin 인터페이스에서 사용되는 ModelForm
 - UserChangeForm 또한 ModelForm이기 때문에 instance 인자로 기존 user 데이터 정보를 받는 구조 또한 동일함
 - 이미 이전에 CustomUserChangeForm으로 확장했기 때문에 CustomUserChangeForm 사용

4.4.1 회원정보 수정 페이지 작성

```
# accounts/urls.py

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

```
<!-- accounts/update.html -->
{% extends 'base.html' %}

{% block content %}
<h1>회원정보수정</h1>
<form action="{% url 'accounts:update' %}" method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit">
</form>
{% endblock content %}
```

```
# accounts/views.py

def update(request):
    if request.method == 'POST':
        pass
    else:
        form = CustomUserChangeForm(instance=request.user)
    context = {
        'form': form,
    }
    return render(request, 'accounts/update.html', context)
```

105

- UserChangeForm 사용 시 문제점
 - 일반 사용자가 접근해서는 안 될 정보들(fields)까지 모두 수정이 가능해짐
 - admin 인터페이스에서 사용되는 ModelForm이기 때문
 - 따라서 UserChangeForm을 상속받아 작성해 두었던 서브 클래스 CustomUserChangeForm에서 접근 가능한 필드를 조정해야 함

- CustomUserChangeForm fields 재정의

```
# accounts/forms.py

class CustomUserChangeForm(UserChangeForm):

    class Meta(UserChangeForm.Meta):
        model = get_user_model()
        fields = ('email', 'first_name', 'last_name',)
```

4.4.2 회원정보 수정 로직 작성

```
# accounts/views.py

def update(request):
    if request.method == 'POST':
        form = CustomUserChangeForm(request.POST, instance=request.user)
        # form = CustomUserChangeForm(data=request.POST, instance=request.user)
        if form.is_valid():
            form.save()
            return redirect('articles:index')
    else:
        form = CustomUserChangeForm(instance=request.user)
    context = {
        'form': form,
    }
    return render(request, 'accounts/update.html', context)
```

4.5 비밀번호 변경

- **PasswordChangeForm**
 - 사용자가 비밀번호를 변경할 수 있도록 하는 Form
 - 이전 비밀번호를 입력하여 비밀번호를 변경할 수 있도록 함
 - 이전 비밀번호를 입력하지 않고 비밀번호를 설정할 수 있는 SetPasswordForm을 상속받는 서브 클래스

4.5.1 비밀번호 변경 페이지 작성

<pre># accounts/urls.py app_name = 'accounts' urlpatterns = [..., path('password/', views.change_password, name='change_password'),]</pre>	<pre># accounts/views.py from django.contrib.auth.forms import AuthenticationForm, PasswordChangeForm def change_password(request): if request.method == 'POST': pass else: form = PasswordChangeForm(request.user) context = { 'form': form, } return render(request, 'accounts/change_password.html', context)</pre>
<pre><!-- accounts/change_password.html --> {% extends 'base.html' %} {% block content %} <h1>비밀번호 변경</h1> <form action="{% url 'accounts:change_password' %}" method="POST"> {% csrf_token %} {{ form.as_p }} <input type="submit"> </form> {% endblock content %}</pre>	

4.5.2 비밀번호 변경 로직 작성

```
# accounts/views.py

def change_password(request):
    if request.method == 'POST':
        form = PasswordChangeForm(request.user, request.POST)
        # form = PasswordChangeForm(user=request.user, data=request.POST)
        if form.is_valid():
            form.save()
            return redirect('articles:index')
    else:
        form = PasswordChangeForm(request.user)
    context = {
        'form': form,
    }
    return render(request, 'accounts/change_password.html', context)
```

- 비밀번호 변경 후 로그인 상태가 지속되지 못하는 문제 발생
 - 비밀번호가 변경되면 기존 세션과의 회원 인증 정보가 일치하지 않게 되어버려 로그인 상태가 유지되지 못함
- 암호 변경시 세션 무효화 방지
 - `update_session_auth_hash(request, user)`
 - 현재 요청과 새로운 session data가 파생 될 업데이트 된 사용자 객체를 가져 오고, session data를 적절하게 업데이트 해줌
 - 암호가 변경 되어도 로그아웃 되지 않도록 새로운 password의 session data로 업데이트

```
from django.contrib.auth import update_session_auth_hash

def change_password(request):
    if request.method == 'POST':
        form = PasswordChangeForm(request.user, request.POST)
        # form = PasswordChangeForm(user=request.user, data=request.POST)
        if form.is_valid():
            form.save()
            update_session_auth_hash(request, form.user)
            return redirect('articles:index')
    else:
        form = PasswordChangeForm(request.user)
    context = {
        'form': form,
    }
    return render(request, 'accounts/change_password.html', context)
```

5. Limiting access to logged-in users

- 로그인 사용자에 대한 접근 제한하기
- 로그인 사용자에 대해 접근을 제한하는 2가지 방법
- The raw way
 - `is_authenticated` attribute
- The `login_required` decorator

5.1 `is_authenticated`

- User model의 속성(attributes) 중 하나
- 사용자가 인증 되었는지 여부를 알 수 있는 방법
- 모든 User 인스턴스에 대해 항상 True인 읽기 전용 속성
 - AnonymousUser에 대해서는 항상 False
- 일반적으로 `request.user`에서 이 속성을 사용
(`request.user.is_authenticated`)
- 권한(permission)과는 관련이 없으면, 사용자가 활성화 상태(active)이거나 유효한 세션을 가지고 있는지도 확인하지 않음

5.1.1 `is_authenticated` 적용하기

- 로그인과 비로그인 상태에서 출력되는 링크를 다르게 설정하기

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

{% if request.user.is_authenticated %}
    <n3>Hello, {{ user }}</n3>
    <form action="{% url 'accounts:logout' %}" method="POST">
        {% csrf_token %}
        <input type="submit" value="Logout">
    </form>
    <a href="{% url 'accounts:update' %}">회원정보수정</a>
    <form action="{% url 'accounts:delete' %}" method="POST">
        {% csrf_token %}
        <input type="submit" value="회원탈퇴">
    </form>
{% else %}
    <a href="{% url 'accounts:login' %}">Login</a>
    <a href="{% url 'accounts:signup' %}">Signup</a>
{% endif %}
```

- 인증된 사용자만 게시글 작성 링크를 볼 수 있도록 처리하기

- 하지만 아직 비 로그인 상태로도 URL을 직접 입력하면 게시글 작성 페이지로 갈 수 있음

```
<!-- articles/index.html -->
{% extends 'base.html' %}

{% block content %}
    <h1>Articles</h1>
    {% if request.user.is_authenticated %}
        <a href="{% url 'articles:create' %}">CREATE</a>
    {% else %}
        <a href="{% url 'accounts:login' %}">새 글을 작성하려면 로그인하세요</a>
    {% endif %}
    ...
{% endblock content %}
```

- 인증된 사용자라면 로그인 로직을 수행할 수 없도록 처리

```
# accounts/views.py

def login(request):
    if request.user.is_authenticated:
        return redirect('articles:index')
    ...
```

5.2 login_required

- login_required decorator
- 사용자가 로그인 되어 있으면 정상적으로 view 함수를 실행
- 로그인 하지 않은 사용자의 경우 settings.py의 LOGIN_URL 문자열 주소로 redirect
 - LOGIN_URL의 기본 값은 /accounts/login
 - 두번째 app 이름을 accounts 로 했던 이유 중 하나

5.2.1 login_required 적용

- 로그인 상태에서만 글을 작성/수정/삭제 할 수 있도록 변경

```
from django.contrib.auth.decorators import login_required

@login_required
@require_http_methods(['GET', 'POST'])
def create(request):
    pass

@login_required
@require_POST
def delete(request, pk):
    pass

@login_required
@require_http_methods(['GET', 'POST'])
def update(request, pk):
    pass
```

5.2.2 적용 확인하기

- /articles/create/ 로 강제접속 시도해보기
- 인증 성공 시 사용자가 redirect 되어야하는 경로는 'next'라는 쿼리 문자열 매개 변수에 저장됨
- /accounts/login/? next=/articles/create/
- 'next' query string parameter
 - 로그인이 정상적으로 진행되면 이전에 요청했던 주소로 redirect 하기 위해 Django가 제공해주는 쿼리 스트링 파라미터
 - 해당 값을 처리할지 말지는 자유이며 별도로 처리 해주지 않으면 view에 설정한 redirect 경로로 이동하게 됨

5.3 두 데코레이터로 인해 발생하는 구조적 문제

- 문제 확인
 - 비로그인 상태로 detail 페이지에서 게시글 삭제 시도
 - delete view 함수의 `@login_required`로 인해 로그인 페이지로 리다이렉트
 - redirect로 이동한 로그인 페이지에서 로그인 진행
 - delete view 함수의 `@require_POST`로 인해 405 상태 코드를 받게됨
 - 로그인 성공 이후 GET method로 next 파라미터 주소에 리다이렉트 되어서
- 두 가지의 문제 발생
 - redirect 과정에서 POST 요청 데이터의 손상
 - redirect로 인한 요청은 GET 요청 메서드로만 요청됨
- 해결 방안
 - `@login_required`는 GET request method를 처리할 수 있는 view 함수에서만 사용해야 함
 - POST method만 허용하는 delete 같은 함수 내부에서는 `is_authenticated` 속성 값을 사용해서 처리

```
# articles/views.py

@login_required
def delete(request, pk):
    if request.user.is_authenticated:
        article = Article.objects.get(pk=pk)
        article.delete()
    return redirect('articles:index')
```

