Django 활용한 웹페이지 구현 연습 1

자료	DB
구분	DB
과목	

1. 기초 설정

바탕화면에 폴더하나 만들고 vscode를 이용해서 열어보자.

맨 처음, 가상환경부터 생성하자.

```
$ python -m venv venv
$ source ./venv/Scripts/activate
```

그리고 django와 autopep8 을 설치 후 requirememts.txt 파일을 만들자.

```
$ pip install django
$ pip install autopep8
```

우리는 영화에 관련된 게시글을 포스팅이 가능하고

누군가가 게시글을 포스팅 하면 해당 게시글에 댓글을 작성할 수 있는

서비스를 구현하고자 한다.

프로젝트와 필요한 앱 생성한다.

- 회원정보를 관리하는 accounts app 그리고
- 영화관련 게시글을 올릴 수 있는 movies app 을 만들자.

```
$ django-admin startproject project .
$ python manage.py startapp accounts
$ python manage.py startapp movies
```

일단, `settings.py` 부터 수정하자.

```
# settings.py
```

```
INSTALLED_APPS = [
    'accounts',
    'movies',
]

TEMPLATES = [
    {
        'DIRS': [BASE_DIR / 'templates'],
    }
]

LANGUAGE_CODE = 'ko-kr'

TIME_ZONE = 'Asia/Seoul'

AUTH_USER_MODEL = 'accounts.User'
```

다음은, 전역 `urls.py` 설정하자.

```
# urls.py

from django.contrib import admin
from django.urls import path, include
from accounts import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('movies/', include('movies.urls')),
    path('accounts/', include('accounts.urls')),
    #path('<int:user_pk>/password/', views.change_password, name='change_password'),
]
```

#path('<int:user_pk>/password/', views.change_password, name='change_password'),

이 부분은 주석 처리를 해 놓자.

나중에 회원정보 수정을 구현 할 때 다시 주석을 풀 것이다.

이제 `accounts` 앱 설정을 하자.

`accounts/models.py` 는 다음과 같다.

```
# accounts/models.py

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

```
class User(AbstractUser):
    pass
```

`auth.models.AbstractUser` 모듈의 AbstractUser 클래스를 상속받아서 `User` 모델을 정의한다.

`pass` 라고 작성했지만, AbstractUser 클래스로 부터 상속된 내용 모두가 들어가 있을 것이며,

이는 커스텀유저 모델이 된다.

`accounts/admin.py` 는 다음과 같다.

```
# accounts/admin.py

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


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

admin에 커스텀유저 모델을 등록했다.

마지막으로, `accounts/urls.py` 는 다음과 같이 작성한다.

```
# accounts/urls.py

from django.urls import path
from . import views

app_name = 'accounts'

urlpatterns = []
```

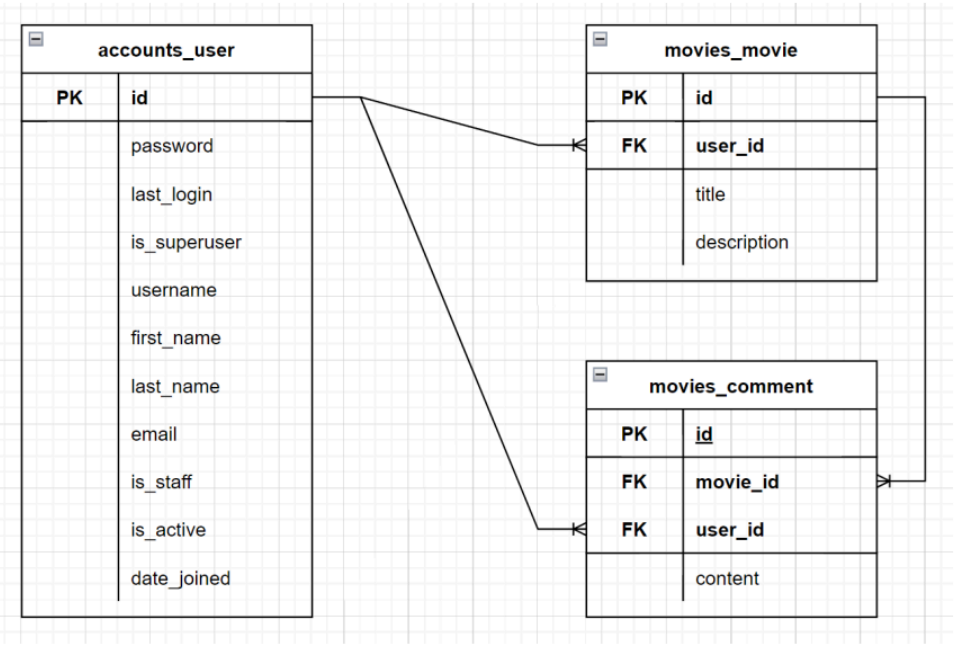
경로는 나중에 적어줄 것이다. 프로젝트의 원활한 진행을 위해

1. 커스텀유저 모델 등록만 먼저 해 놓고
2. movies 앱을 먼저 구현을 한 후에
3. accounts 앱도 구현을 하면서 이 프로젝트를 마무리 할 것이다.

커스텀유저 모델을 개발 초기 설정 단계에서 해 주는 것이 개발을 진행함에 있어 유연성 확장성 측면에서 바람직하다고 볼 수 있겠다.

자, 이제부터 `movies` 앱을 설정하자.

우리가 구현할 ERD 는 다음과 같은 형태이다.



이 중, `accounts_user` 모델은 복잡해 보이지만 하나도 신경 쓸 것이 없다.
모델 안에 있는 필드들은 직접 구현하는 게 아니라,
이미 Django 에서 제공하는 커스텀유저의 필드들이기 때문이다.

다만, 우리가 눈 여겨 봐야 할 것은 `accounts_user`, `movies_movie`, `movies_comments`
DB 테이블의 관계다. 모두 1 대 N 관계이다. ERD의 PK 와 FK 의 위치를 잘 살펴보자.
각 테이블의 PK는 accounts user의 ID가 될 것이고 movies_movie 테이블에서의 ID
그리고 movies_comment모델에도 ID 라고 할 수 있겠다.

`accounts_user` 의 PK (회원정보) 는
`movies_movie`, [영화관련 게시판] 에 게시글을 포스팅 한 사람이 누구인지? 와 연관되어 있다.
`movies_comments` , 만약 특정 게시글에 [“댓글”]이 있다면 그 댓글을 누가 작성 하였는지도
연관되어 있다고 유추할 수 있을 것이다.

추가적으로,
`movies_comments` [게시글의 “댓글”]이 있다면 해당 댓글이
어떠한 게시글에 대한 댓글인지 알 수 있도록 `movies_movie` ID와 연관되어 있다는 것을
확인 할 수 있을 것이다.
그리고 ERD에서 테이블 간의 관계를 표현 할 때에는 까마귀발 **”Crow's Foot” Notation** 을 사용해서
테이블 간의 관계를 표현을 해 준다.

`movies/models.py` 는 다음과 같다.

```
# movies/models.py

from django.db import models
from django.conf import settings


class Movie(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
                             on_delete=models.CASCADE)
    title = models.CharField(max_length=20)
    description = models.TextField()


class Comment(models.Model):
    movie = models.ForeignKey(Movie, on_delete=models.CASCADE)
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
                             on_delete=models.CASCADE)
    content = models.CharField(max_length=100)
```

먼저, `from django.conf import settings` 을 import 해왔다.

그 이유는 게시글을 작성한 작성자를 의미하는 Movie 클래스에서의 user 속성은
우리가 커스텀한 Accounts 앱의 User 모델을 참조해야 한다.
이때 커스텀 User 모델을 사용 하기 위해서 `settings.AUTH_USER_MODEL` 을 참고해야 하기 때문이다.

`settings.AUTH_USER_MODEL` 에 대해서 이야기 해보자.
User 모델을 참조할 때 `settings.AUTH_USER_MODEL` 을 사용한다고 했다.

사실, Django에서 Accounts 앱의 User 모델을 참조하는 방법은 2가지가 있다.

1. `get_user_model()` 이라는 함수를 이용해서 User 모델을 참조할 수 도 있고
2. `settings.AUTH_USER_MODEL` 을 이용해서 User 모델을 참조하는 방법이 있다.

두 방식의 차이점이 있다.

`get_user_model()` 을 동적 참조라고 해서 객체의 형태로 User모델을 참조 할 때 사용한다.
실제로 view.py / forms.py 그리고 우리가 아직은 배우지 않았지만, 시리얼라이저(serializer)에서
User모델의 객체를 생성하거나 직접 참조를 할 때 사용하는 방식이다.

`settings.AUTH_USER_MODEL` 은 정적 참조 방식이라 하며

방금과 같은 Models.py 에서 모델을 정의할 때 사용하는 방식이다.

모델 클래스에서 user 라는 외래키(FK)를 설정 시 우리는 'accounts.user'라고 문자열의 형태로

User 모델을 참조 한다. User모델의 "객체를 직접 참조"하는 방식이 아닌 "문자열의 형태로 간접적으로 참조"하는 방식이다.

쉽게 결론만 말하자면, models.py에서는 `settings.AUTH_USER_MODEL` 을 사용해서 User모델을 참조하고 models.py 가 아닌 다른 곳에서 User 모델을 참조할 때에는 `get_user_model()` 을 사용하는 것을 Django에서 권장한다는 것을 기억해 두자.

```
class Movie(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
                             on_delete=models.CASCADE)
    title = models.CharField(max_length=20)
    description = models.TextField()


class Comment(models.Model):
    movie = models.ForeignKey(Movie, on_delete=models.CASCADE)
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    content = models.CharField(max_length=100)
```

위에서 이미 작성한 Comment 모델이다. 구문을 하나씩 살펴보자.

- models.ForeignKey() 함수를 통해서 FK 설정을 한다. 그 다음 괄호 안에는
- 내가 참조할 모델을 작성한다. Comment 클래스의 movie 속성은 Comment 모델 위에 정의해 놓은 Movie 클래스를 참조할 것이며
- user 속성은 settings.AUTH_USER_MODEL 을 통해서 Account앱의 user모델을 참조할 것이다.
- 만약에 영화관련 게시글을 작성한 사람이 해당 게시글을 삭제 했을 경우

  게시글에 달린 댓글들도 함께 삭제가 되어야 할 것이다. 그 옵션을 넣어 준 것이

  on_delete=models.CASCADE 옵션이 되겠다.

`movies/admin.py` 는 다음과 같다.

```
# movies/admin.py

from django.contrib import admin
from .models import Movie, Comment
```

```
admin.site.register(Movie)
admin.site.register(Comment)
```

지금까지 한 것들을 정리해 보자.

1. 앱 만들고 settings.py에 등록한다. 그리고 전역 urls.py 에 경로 등록했다.
2. accounts 앱의 models.py / admin.py 에 커스텀유저 모델 생성 및 등록을 하고 urls.py 에 최소한으로 필요한 기본 코드만 작성했다.
3. articles 앱의 models.py 에 Movie 그리고 Comment 모델을 정의 했고 admin.py 에 등록까지 했다.

자, 이제 사용자가 `localhost:8000/movies` 로 접속했을 때 보일 간단한 index 창을 만들자.

`movies/urls.py` 를 생성한 후, 다음과 같이 작성한다.

```
# movies/urls.py

from django.urls import path
from . import views

app_name = 'movies'

urlpatterns = [
    path('', views.index, name='index'),
]
```

`movies/views.py` 는 다음과 같다.

```
# movies/views.py

from django.shortcuts import render

def index(request):
    return render(request, 'movies/index.html')
```

프로젝트 루트 경로 바로 아래에, `templates/base.html` 을 다음과 같이 작성한다.

(전역 templates 폴더 생성 후 base.html 파일 생성)

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

```
    <title>Document</title>
  </head>
  <body>
    <div>
      {% block content %}

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

`movies/templates/movies/index.html` 을 생성하고, 다음과 같이 작성한다.

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

{% block content %}
  <h1>INDEX</h1>
{% endblock %}
```

이제 마이그레이션 생성 및 마이그레이트, 관리자 생성 후, 테스트서버를 구동해보자.

```
$ python manage.py makemigrations
$ python manage.py migrate
$ python manage.py createsuperuser
$ python manage.py runserver
```

서버 켜서 http://127.0.0.1:8000/movies/ 확인 해보면 index 라고 잘 보일 것이다.

admin 관리자 페이지에서 Comments 와 Movies 테이블이 확인 되어야 한다.

2. accounts

accounts 앱을 통해서 회원들의 회원가입 및 탈퇴 / 로그인 로그아웃 / 회원정보 수정까지
먼저 구현 해 보자.

미리 `urls.py` 를 정의해두자.
다음 조건에 맞춰서 작성한다.

URL 패턴	역할
/accounts/login/	로그인 페이지 조회 & 세션 데이터 생성 및 저장 (로그인)
/accounts/logout/	세션 데이터 삭제 (로그아웃)
/accounts/signup/	회원 생성 페이지 조회 & 단일 회원 데이터 생성 (회원가입)
/accounts/delete/	단일 회원 데이터 삭제 (회원탈퇴)
/accounts/update/	회원 수정 페이지 조회 & 단일 회원 데이터 수정 (회원정보수정)
/accounts/password/	비밀번호 수정 페이지 조회 & 단일 비밀번호 데이터 수정 (비밀번호변경)

```
# 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'),
    path('signup/', views.signup, name='signup'),
    path('delete/', views.delete, name='delete'),
    path('update/', views.update, name='update'),
]
```

`views.py` 작성 전, `forms.py` 를 만들겠다.

```
# accounts/forms.py

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


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


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

`UserCreationForm` 과 `UserChangeForm` 을 각각 `CustomUserCreationForm` , `CustomUserChangeForm`으로 상속받아 오버라이딩할 것이다.

`get_user_model()` 을 사용해 Django 에서 제공하는 유저 모델을 그대로 가져온 후,
사용하고자 하는 필드만 적어준다.

유저 생성에서는 `username` 과 `email` 만 가져다가 사용하겠다. 그리고 유저 수정은 `email` 만 바꾸도록 허용 하도록 해보겠다.
비밀번호 변경 폼은 Django 에서 기본적으로 제공하니 지금은 신경 쓰지 않아도 좋다.

`signup` 함수를 `views.py` 에 작성해보자.

```python
# 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.


def login(request):
    pass


def logout(request):
    pass


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

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


def delete(request):
    pass


def update(request):
    pass


def change_password(request):
    pass
```

우리는 `urls.py` 에 이 프로젝트에서 사용 할 URL들을 미리 작성했기에, 에러 방지를 위해
미리 함수들을 만들어 두고 pass를 넣어 두었다.

복습해보자.
사용자가 로그인한 상태라면 회원가입 페이지에 들어오지 못하고 `movie:index` 로 향한다.
로그인 안 한 상태일 경우라면 POST 일 경우와 GET 일 경우로 달라진다.

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

POST 는 `signup.html` 에서만 보낼 수 있도록 할 것이다.
사용자가 폼을 다 작성 후에 `submit` 버튼을 눌러야만 동작하는 부분이 되겠다.

사용자가 입력한 폼이 양식에 맞는지(`is_valid`) 판단한 후, 맞으면 회원가입 후 로그인하고, `movies: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.

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

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <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 %}
```

```
    </div>
  </body>
</html>
```

`<nav>` 태그 안에, 만약 사용자가 로그인한 상태라면 (인증된 사용자라면) 이름이 출력 되고,
그렇지 않다면 회원가입 링크를 클릭할 수 있도록 만들었다.

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

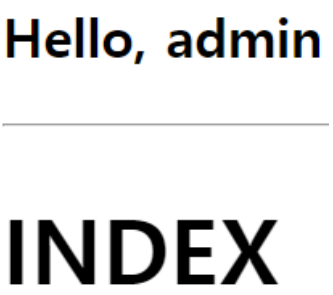
```
# accounts/signup.html

{% 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 잊지 말고 꼭 넣어주자.

서버를 다시 켜서 http://127.0.0.1:8000/movies/ 를 확인해 보자.
admin으로 로그인이 되어 있다면 다음과 같이 보일 것이다.



관리자 페이지에서 다시 로그아웃을 한 후에 확인을 해보면

위와 같이 보일 것이다.

그리고 signup을 클릭을 해서 url을 확인해 보고 http://127.0.0.1:8000/accounts/signup/
버그 없이 화면이 잘 렌더링 되는지 확인해 보자.

그리고 회원가입을 통해서 유저를 하나 생성해 놓자.
나는 kevin 이라는 유저를 하나 생성했다.

이제 사용자를 로그아웃 시킬 수 있도록 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>
```

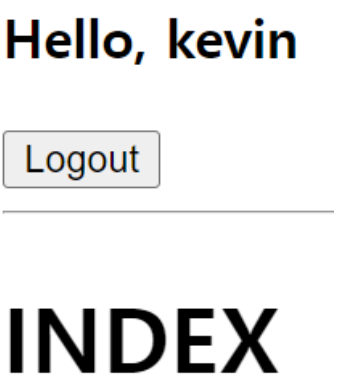
```
  {% endif %}
</nav>
```

코드를 살펴보면, 로그아웃 부분이 추가되었다. 당연히, 로그인이 된 상태에서만 로그아웃 버튼이 보여야 할 것이다.

`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('movies:index')
```



로그아웃이 생겼고,

클릭하면 정상적으로 로그아웃된다.

나중에 다양한 테스트를 더 하기 위해서 미리

회원가입을 통해서 유저를 두 명만 더 만들어보자.

지금까지 회원가입 그리고 로그아웃 기능을 구현 하였다.

다음, 회원 탈퇴를 만들어보자.

`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>
    <form action="{% url 'accounts:delete' %}" method="POST">
      {% csrf_token %}
      <input type="submit" value="회원탈퇴">
    </form>
  {% else %}
    <a href="{% url 'accounts:signup' %}">Signup</a>
  {% endif %}
</nav>
```

회원탈퇴를 담당하는 `<form>` 을 하나 더 추가 하였다.

`accounts/views.py` 의 `delete` 를 다음과 같이 작성한다.

```
# accounts/views.py

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

만약 로그인된 상태라면, 유저를 삭제하고 로그아웃한다.

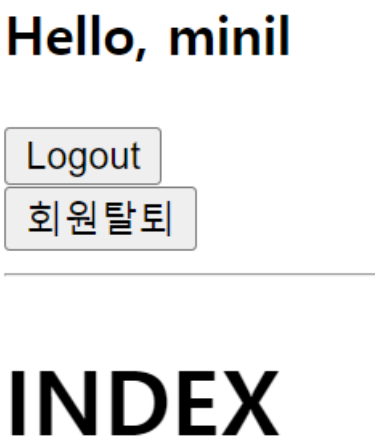
코드를 구현할 때 순서가 중요하다. 유저 삭제한 후에 로그아웃을 해야 한다.

로그아웃을 먼저 해 버리면 어떠한 유저를 회원탈퇴를 해야 할지 모를 것이기 때문이다.

구현하면서 실수를 많이 하는 부분이라 다시 강조하자면, `request.user.delete()`

를 통해서 먼저 회원삭제를 한 후에 `auth_logout` 을 통해서 로그아웃을 진행을 하자.
탈퇴를 한 후에는 `movies:index` 로 리다이렉트 했다.

나는 아까 minil 이라는 유저를 하나 만들었더니 아래와 같은 화면이 보였으며



로그인한 상태에서, 회원탈퇴 버튼이 보인다. 회원탈퇴를 눌러 보자.

그리고 sqlite DB를 통해서 해당유저가 존재하는지 확인해 보자.
minil 이라는 아이디는 더 이상 존재하지 않는 것을 확인 할 수 있다.

지금까지 회원가입 > 로그아웃 > 회원탈퇴를 구현 하였다.
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>
  <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 %}
</nav>
```

`else` 구문 안에서, signup 바로 위에 로그인 링크를 걸었다.

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

```
# accounts.py

from django.contrib.auth.forms import AuthenticationForm

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

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

`signup` 함수와 형태가 매우 비슷하니 자세한 설명은 생략한다.

다만, 여기에서 로그인 폼을 사용하기 위해 Django 에서 기본 제공하는 `AuthenticationForm` 을 사용했다.

`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 }}
```

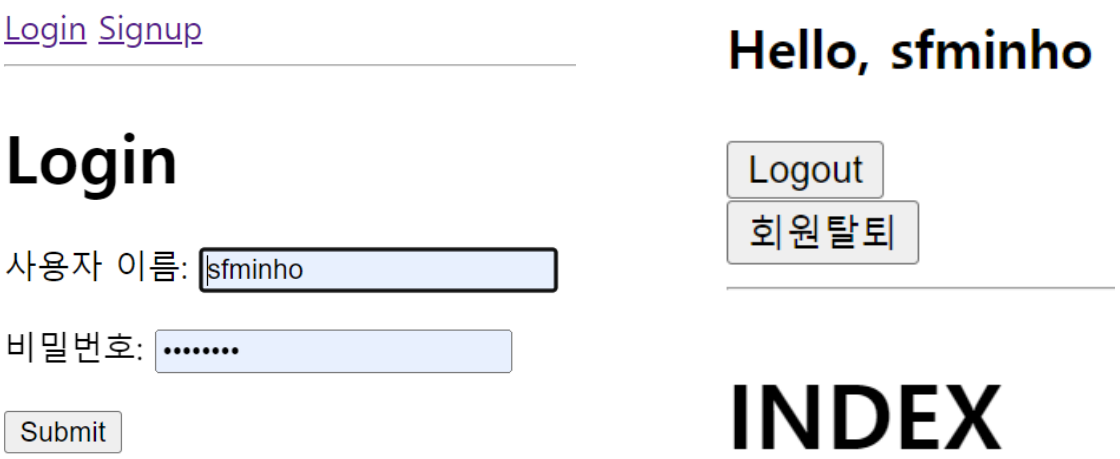
```
    <button type="submit">Submit</button>
  </form>
{% endblock content %}
```

signup.html 코드와 비교를 해보면 구조 상 큰 차이가 없는 것을 확인 할 수 있을 것이다.

http://127.0.0.1:8000/movies/ 를 통해서 확인해 보자.

좌측 화면에서 Logout을 누른 후 Login 링크를 통해서 로그인을 테스트 해보자.

아까 만들어 두었던 아이디를 이용해서 로그인 테스트를 해보자.



이상 없이 잘 되었다면 회원정보를 수정 할 수 있도록 코드를 구현해 보자.

먼저 프로젝트파일/urls.py 에서 (전역 urls.py 파일) 주석 처리 해 놓았던 것을 풀자.

```
# urls.py

from django.contrib import admin
from django.urls import path, include
from accounts import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('movies/', include('movies.urls')),
    path('accounts/', include('accounts.urls')),
    path('<int:user_pk>/password/', views.change_password, name='change_password'),
]
```

`base.html` 로 가자.

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

`<h3>` 태그 바로 아래에 `<a>` 태그로, 회원 정보 수정으로 향하는 링크를 달았다.

`accounts/views.py` 의 `update` 와, `change_password` 를 다음과 같이 작성했다.

```
from django.contrib.auth.decorators import login_required
from .forms import CustomUserCreationForm, CustomUserChangeForm
from django.contrib.auth.forms import AuthenticationForm, PasswordChangeForm
from django.contrib.auth import update_session_auth_hash

@login_required
```

```
@require_http_methods(['GET', 'POST'])
def update(request):
    if request.method == 'POST':
        form = CustomUserChangeForm(request.POST, instance=request.user)
        if form.is_valid():
            form.save()
            return redirect('movies:index')
    else:
        form = CustomUserChangeForm(instance=request.user)
    context = {
        'form': form,
    }
    return render(request, 'accounts/update.html', context)


@login_required
@require_http_methods(['GET', 'POST'])
def change_password(request,user_pk):
    if request.method == 'POST':
        form = PasswordChangeForm(request.user, request.POST)
        if form.is_valid():
            form.save()
            update_session_auth_hash(request, form.user)
            return redirect('movies:index')
    else:
        form = PasswordChangeForm(request.user)
    context = {
        'form': form,
    }
    return render(request, 'accounts/change_password.html', context)
```

`update` 함수에서는 현재 접속한 유저의 정보를 그대로 폼에 적용하기 위해 `instance=request.user` 를 인자로 추가했다.

`change_password` 함수에서는 Django 에서 제공하는 `PasswordChangeForm` 을 가져와 사용한다. 특히 비밀번호는 해시 값 처리를 해야 하기 때문에, `update_session_auth_hash` 를 사용해 해시처리 해 두었다. 이는 암호 변경시 세션 무효화를 막아주기도 한다.

`accounts/update.html` 는 다음과 같다.

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

{% block content %}
  <h1>회원정보 수정</h1>
  <form action="{% url 'accounts:update' %}" method="POST">
```

```
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Submit</button>
  </form>
{% endblock content %}
```

`accounts/change_password.html` 은 다음과 같다.

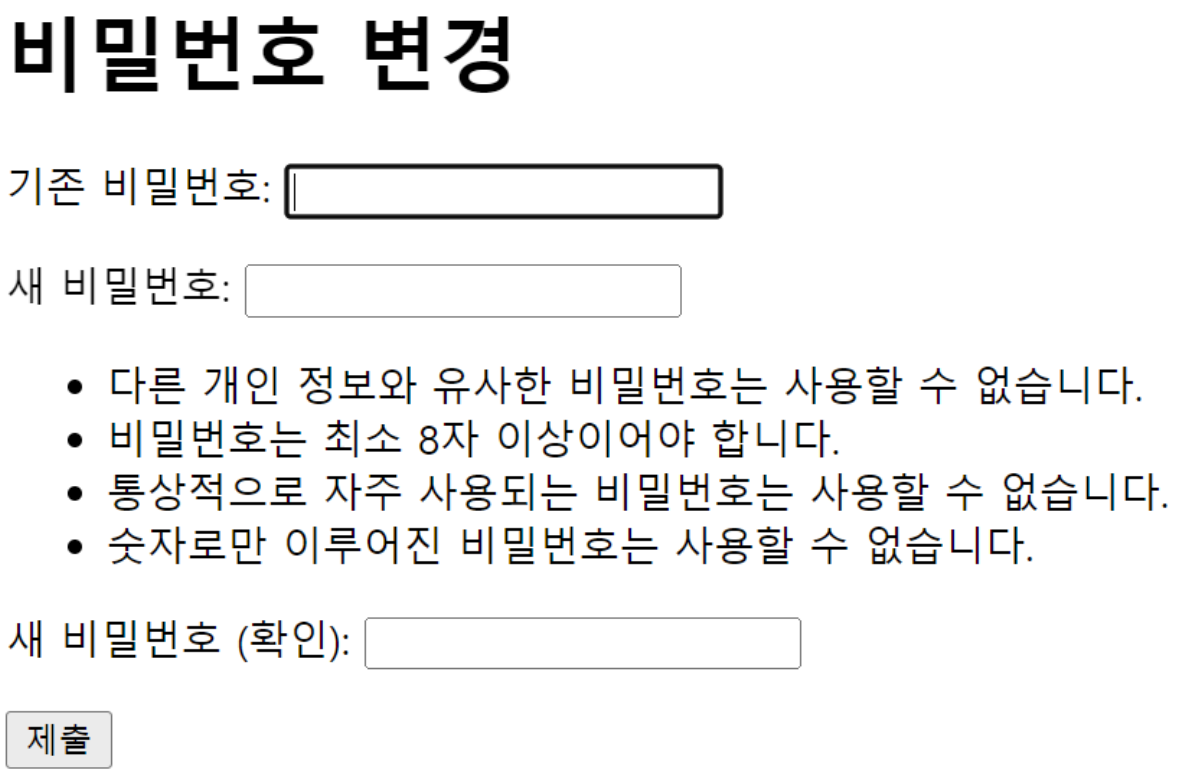
```
{% extends 'base.html' %}

{% block content %}
  <h1>비밀번호 변경</h1>
  <form action="{% url 'change_password' user.pk %}" method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Submit</button>
  </form>
{% endblock content %}
```

서버 켜서 확인해 보자.

여기서 변경할 수 있는 유저의 정보는 이메일로 한정했다. 그리고 비밀번호를 변경하려면

"폼" 이라는 링크를 클릭하면 되는데 [폼] 을 눌러 확인해 보자



새 비밀번호를 넣고 제출 후 다시 로그인이 되는지 확인해 보자.

여기까지 Accounts 앱을 완성했다.

우리는 처음 프로젝트 생성부터 하나씩 빌드업을 진행했다.

지금까지 한 것들을 정리해 보자.

1. 앱 만들고 settings.py에 등록한다. 그리고 전역 urls.py 에 경로 등록했다.
2. accounts 앱의 models.py / admin.py 에 커스텀유저 모델 생성 및 등록을 하고 urls.py 에 최소한으로 필요한 기본 코드만 작성했다.
3. movies 앱의 models.py 에 Movie 그리고 Comment 모델을 정의 했고 admin.py 에 등록까지 했다.
4. movies 앱의 기본적인 url view templete 순으로 index 페이지 하나 만들었다.
5. accounts 앱에 회원가입 > 로그아웃 > 회원탈퇴 > 로그인 > 회원정보수정 > 비밀번호수정 순으로 구현을 완성했다.

이제 본격적으로 movies 앱을 완성시켜 보자.

movies 앱은 영화관련 게시글을 포스팅 할 수 있는 게시판을 구현 할 것이다.

그리고 게시판의 게시글에 대한 댓글을 작성할 수 있는 기능까지 추가 할 것이다

다음 PDF 파일로 넘어가자.