

# WEB PJT 4. 프레임워크를 활용한 웹페이지 구현 연습

📎 자료	<u>관통프로젝트</u>
☰ 구분	Django,PJT

## 1. 기초 설정

바탕화면에 폴더하나 만들고 vscode를 이용해서 열어보자.

맨 처음, 가상환경부터 생성하자.

홈디렉토리의 venv 폴더 안에 가상환경을 하나 만들자.

```
$ python -m venv ~/venv/practice
$ source ~/venv/practice/Scripts/activate
```

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

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

우리는 영화에 관련된 게시글을 포스팅 하고  
(수업중에는 articles 라고 app을 주고 만들었었음)

누군가가 게시글을 포스팅 하면 해당 게시글에 댓글을 작성할 수 있는  
서비스를 구현하고자 한다.

프로젝트와 필요한 앱 생성한다.

회원정보를 관리하는 accounts앱 과 영화관련 게시글을 올릴 수 있는 movies앱을 하나 만들자.

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

일단, `settings.py` 부터 건드린다.

```
# settings.py

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

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

LANGUAGE_CODE = 'ko-kr'

USE_TZ = 'Asia/Seoul'

AUTH_USER_MODEL = 'accounts.User'
```

다음, 전역 `urls.py` 설정하자.

```
# urls.py

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

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'),
]
```

이제 `accounts` 앱을 설정하자.

`accounts/models.py` 는 다음과 같다.

```
# accounts/models.py

from django.contrib.auth.models import AbstractUser

class User(AbstractUser):
    pass
```

`auth.models.AbstractUser` 를 상속받은 `User` 모델을 사용할 것이다. `pass` 라고 되어 있지만, 실제로 상속된 내용 모두가 들어가며, 이것이 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)
```

어드민 사이트에 커스텀유저 모델을 등록한다.

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

```
# accounts/urls.py

from django.urls import path
from . import views

app_name = 'accounts'

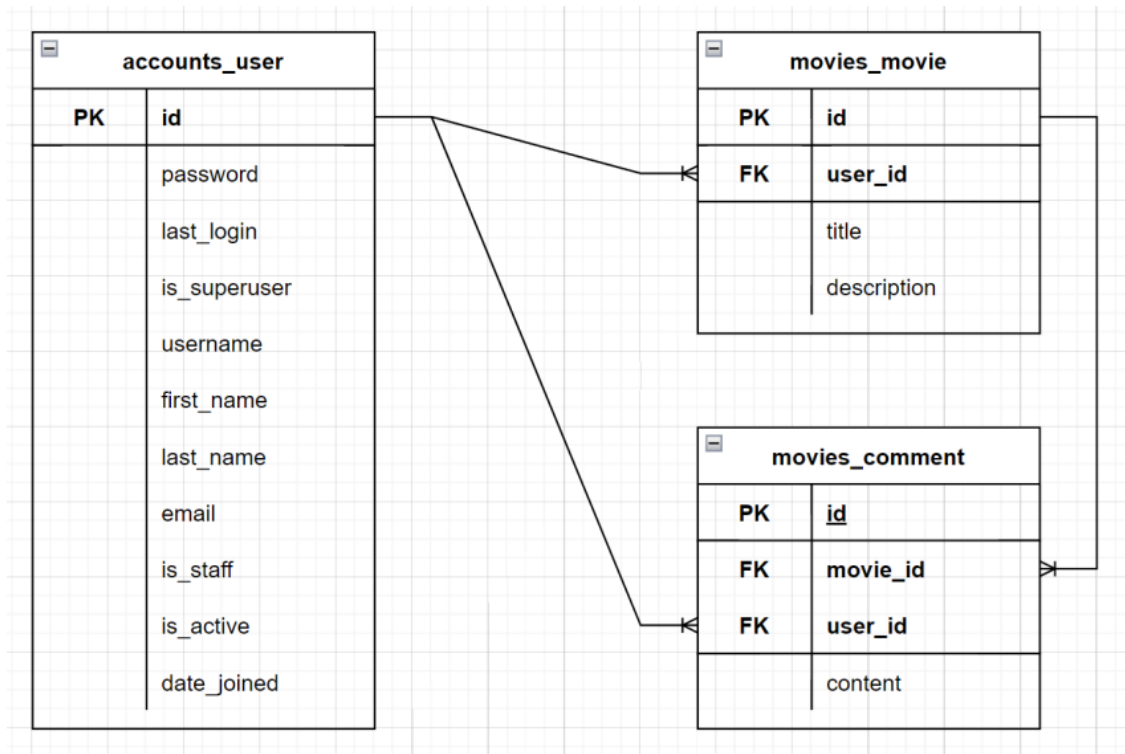
urlpatterns = []
```

경로는 나중에 적어줄 것이다. 커스텀유저 모델 등록만 먼저 해 놓고

movies 앱을 먼저 구현을 한 후에 accounts 앱도 구현을 하면서 마무리 짓도록 하겠다.

`movies` 앱을 설정하자.

우리가 구현할 ERD 는 다음과 같은 형태이다.



이 중, `accounts_user` 는 복잡해 보이지만 하나도 신경 쓸 것이 없다. 왜냐면, 저 필드들은 직접 구현하는 게 아니라, 이미 Django 에서 제공하는 커스텀유저의 필드들이기 때문이다.

다만, `accounts_user` , `movies_movie` , `movies_comments` DB 테이블의 관계가 중요하다. 모두 1 대 N 관계이다. ERD의 PK 와 FK 의 위치를 잘 살펴보아라. PK는 accounts user의 ID가 있고 또 다른 PK로는 movies\_movie의 게시글 ID가 있다.

`accounts_user` 의 PK (회원정보) 는

`movies_movie` , [영화관련 게시판] 에 게시글을 포스팅 한 사람이 누구인지와 연관되어 있고 그리고

`movies_comments` , [게시글의 “댓글”]이 있다면 그 댓글을 누가 작성 하였는지 와도 연관되어 있다고 유추가 될 것이다.

그리고 `movies_comments` [게시글의 “댓글”]이 있다면 해당 댓글이

어떠한 게시글의 댓글인지 알 수 있도록 `movies_movie` ID와 연관되어 있다는 것을 확인 할 수 있을 것이다.

`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)
```

`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)
```

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

# Create your views here.
def index(request):
    return render(request, 'movies/index.html')

```

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

```

<!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 content %}
    </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/> 확인을 해보면  
index 라고 잘 보일 것이다.

admin 관리자 페이지에서 Comments 와 Movies 테이블이 확인 되어야 한다.

## 사이트 관리

ACCOUNTS		
사용자(들)	+ 추가	✎ 변경
MOVIES		
Comments	+ 추가	✎ 변경
Movies	+ 추가	✎ 변경
인증 및 권한		
그룹	+ 추가	✎ 변경

## 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
<code>/accounts/login/</code>	로그인 페이지 조회 & 세션 데이터 생성 및 저장 (로그인)
<code>/accounts/logout/</code>	세션 데이터 삭제 (로그아웃)
<code>/accounts/signup/</code>	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, 사용하고 자 하는 필드만 적어준다. 유저 생성에서는 `username` 과 `email` 만 가져다가 사용하겠다. 유저 수정은 `email` 만 바꾸도록 허용 한다.

비밀번호 변경 폼은 Django 에서 기본적으로 제공하니 신경 쓰지 않아도 좋다.

`signup` 함수를 `views.py` 에 작성해보자.

```
# 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 URL들을 미리 작성했기에, 에러 방지를 위해 미리 함수들을 만들어 두고 `pass`를 넣어 두었다.

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

POST 는 `signup.html` 에서만 보낼 수 있다. 사용자가 폼을 다 작성 후에 `submit` 버튼을 눌러야만 동작하는 부분이다.

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.

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

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

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <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 content %}
    </div>
  </body>
</html>
```

`<nav>` 태그 안에, 만약 사용자가 로그인한 상태라면 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 잊지말고 꼭 넣어주자.

서버를 다시 켜서 `http://127.0.0.1:8000/movies/` 를 확인해 보자.

admin으로 로그인이 되어 있다면 다음과 같이 보일 것이며

**Hello, admin**

---

**INDEX**

관리자 페이지에서 다시 로그아웃을 한 후에 확인을 해보면

[Signup](#)

---

# INDEX

다음과 같이 보일 것이다.

그리고 signup을 클릭을 해서 url을 확인해 보고

<http://127.0.0.1:8000/accounts/signup/> 와 같을 것이며 버그 없이 화면이 잘 렌더링 되는지 확인해 보자. 그리고 회원가입을 통해서 유저를 하나 생성해 놓자.

나는 kevin 이라는 유저를 하나 생성했다.

Hello, kevin

---

# INDEX

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, 로그인인 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')

```

만약 사용자가 로그인된 상태라면 로그아웃 진행하고 `movies:index` 로 리다이렉트 할 것이며 혹시 로그인을 하지 않은 상태라면 그냥 리다이렉트이다.

# Hello, kevin

Logout

---

# INDEX

로그아웃이 생겼고,

[Signup](#)

---

# INDEX

클릭하면 정상적으로 로그아웃된다.

나중에 테스트를 더 할 수 있도록 미리 유저를 두 명 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 구현할때 순서가 중요하다. 유저 삭제한 후에 로그아웃해야한다.

로그아웃을 먼저 해 버리면 어떠한 유저를 회원탈퇴를 해야 할지 모를 것이다.

구현하면서 실수를 많이 하는 부분이라 다시 강조하자면, `request.user.delete()` 를 통해서 먼저 회원삭제를 한 후에 `auth_logout` 을 통해서 로그아웃을 진행을 한다.

탈퇴를 한 후에는 `movies:index` 로 리다이렉트 했다.

나는 아까 minil 이라는 유저를 하나 만들었더니 아래와 같은 화면이 보였으며



# Hello, minil

Logout

회원탈퇴

---

## INDEX

로그인한 상태에서, 회원탈퇴 버튼이 보인다. 회원탈퇴를 누르면

[Signup](#)

---

## INDEX

누르면, 다음 화면으로 바뀌고, DB를 통해서 해당유저가 존재하는지 확인해 보자.  
minil 이라는 아이디는 더 이상 존재하지 않는 것을 확인 할 수 있다.

SQL ▾ < 1 / 1 > 1 - 3 of 3				
	last_login	is_superuser	username	first_name
=	2023-10-05 01:21:46.273632	1	admin	
=	2023-10-05 01:04:04.134012	0	kevin	
=	2023-10-05 01:11:37.442461	0	sfminho	

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, 여기서 로그인 폼을 사용하기 위해 Django 에서 기존 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 비교해 보자. 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.

Hello, admin

[Login](#) [Signup](#)

---

Logout

회원탈퇴

INDEX

INDEX

아까 만들어 두었던 아이디를 이용해서 로그인 테스트를 해보자.

[Login](#) [Signup](#)

---

Login

사용자 이름:

비밀번호:

Submit

Hello, sfminho

Logout

회원탈퇴

INDEX

이상 없이 잘 되었다면

회원정보를 수정 할 수 있도록 코드를 구현해 보자.

`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 작성한다.

```
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 해시값 처리를 해야 하기 때문에, `update_session_auth_hash` 를 사용해 해시처리 해 두었다.

`accounts/update.html` 는 다음과 같다.

```

{% extends 'base.html' %}

{% block content %}
<h1>회원정보수정</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 'accounts:change_password' user.pk %}" method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Submit</button>

```

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

서버 켜서 확인해 보자.

Hello, sfminho

[회원정보수정](#)

Logout

회원탈퇴

## INDEX

### 회원정보수정

이메일 주소:

비밀번호:

비밀번호가 설정되지 않습니다.

원본 비밀번호는 저장되지 않으므로, 해당 사용자의 비밀번호를 확인할 수 없습니다. 다만 이 [폼](#)을 사용하여 비밀번호를 변경할 수 있습니다.

Submit

여기서 변경할 수 있는 유저의 정보는 이메일로 한정된다. 그리고 비밀번호를 변경하려면 “다만 이 폼을 사용하여~”에 적힌 “폼”이라는 링크를 클릭하면 되는데, Django에서는 이 문구에서 “폼”의 링크를 `password`로 걸어두었고, 이것은 `accounts/urls.py`에 의해, `change_password` 함수를 실행시키게 된다.

[폼]을 눌러 확인해 보자

# 비밀번호 변경

기존 비밀번호:

새 비밀번호:

- 다른 개인 정보와 유사한 비밀번호는 사용할 수 없습니다.
- 비밀번호는 최소 8자 이상이어야 합니다.
- 통상적으로 자주 사용되는 비밀번호는 사용할 수 없습니다.
- 숫자로만 이루어진 비밀번호는 사용할 수 없습니다.

새 비밀번호 (확인):

제출

새 비밀번호를 넣고 제출 후 다시 로그인인지 확인해 보자.

## 3. Movies

이번에는 영화관련 게시글을 포스팅 할 수 있는 게시판을 구현 할 것이고 그리고 게시판의 게시글 마다 댓글을 달 수 있는 기능까지 추가 할 것이다. to be continue