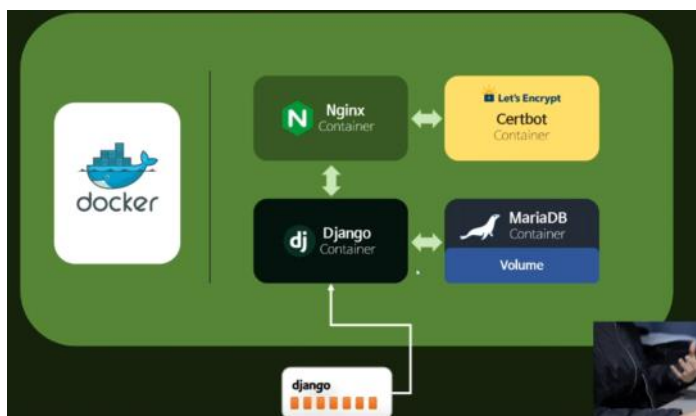


Django를 이용한 Pinterest 구현

2021년 7월 5일 월요일 오후 8:52



django



- 실습

<https://github.com/ddubi6796/hjasm>

Django Tutorial

2021년 5월 30일 일요일 오후 2:44

Django feat.Pinterest

- 웹서비스 구조

- 기술스택

Front-end: JS, HTML, CSS

Back-end: MariaDB, NGINX, django, docker

- Django 내부 앱 구성

Account, Article, Project, Comment

- django 개발 패턴(MVT)

- MVC패턴과 유사

- Model: django <-> database

- View: user <-> server; 사용자의 request에 대한 서버의 response(사용자 인증절차, 유효성체크, 응답구성)

- Template: Front-end에 해당, user interface 구현부

[Template]

- django 프로젝트에 앱 추가하기

- 신규 앱 추가 시, 메인 앱 내부 setting.py 내 INSTALLED_APPS에 해당 신규 앱 명시적으로 추가 필요함

- View 생성 후 라우팅을 위하여 메인 앱 내부 urls.py 내 urlpatterns에 해당 view의 urlpattern을 명시함

ex) path('account/', include('accountapp.urls')), -> accountapp.urls 내부 url을 모두 포함하여 하위 디렉터리로 분기하도록 설정

- 새로운 템플릿 추가하기

- 신규 템플릿 추가 시, 메인 앱 내부 setting.py 내 TEMPLATES의 DIRS에 해당 신규 템플릿 경로를 명시적으로 추가 필요

- tip) 각각의 앱(ex. Accountapp)에 템플릿 경로 추가 시 templates/accountapp과 같이 앱명과 동일한 경로를 추가한다.(추후 view에서 의 가독성을 위함)

- 재사용 가능한 템플릿 만들기: django 템플릿의 extends, include

- extends: pre-made html template을 가져와서 해당 템플릿을 기반으로 내용을 구현하는 개념

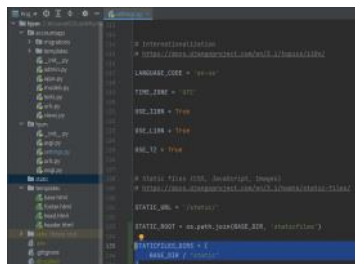
- include: 작성된 템플릿 내에 특정 소스를 가져와 내용을 채우는 개념

[View]

- Static 설정 및 CSS 파일 분리

- collect static: Static 파일을 특정 디렉터리에 모아서 관리

- static 디렉터리 설정



- static 구문 사용: {%loadstatic%}

<https://docs.djangoproject.com/en/3.2/howto/static-files/>

- CSS Essentials

- DISPLAY 속성

1) Block: 각 태그의 Parent-Children 구조에서 Parent's 100% 사이즈를 가지는 속성값

- 2) Inline: Just text line height "in" line (텍스트의 사이즈 만큼의 높이를 가지는 속성값)
- 3) Inline-block: Inline, But behave like "block" (100% width를 가지지 않음)
- 4) NONE: 태그는 존재하나 보이지 않음 visibility="hidden"과 비슷
- SIZE 속성(반응형 웹 구축 시 중요)
 - 1) px: parent-children 독립적 관계 (영향 X)
 - 2) em: parent의 폰트 사이즈의 동일한 비율로 child 영향, 부모가 여러개일 경우 중첩되어 영향
 - 3) rem: root HTML의 기본 폰트 사이즈의 동일한 비율로 child 영향(parent의 영향 X) = 브라우저 기본 px (=16px)
 - 4) %: parent-children 독립적 관계 (영향 X)
- => rem, %를 주로 사용함

• CSS 적용순서

- inline 스타일 > 소스 <style>태그 내 작성된 스타일 > 외부 css

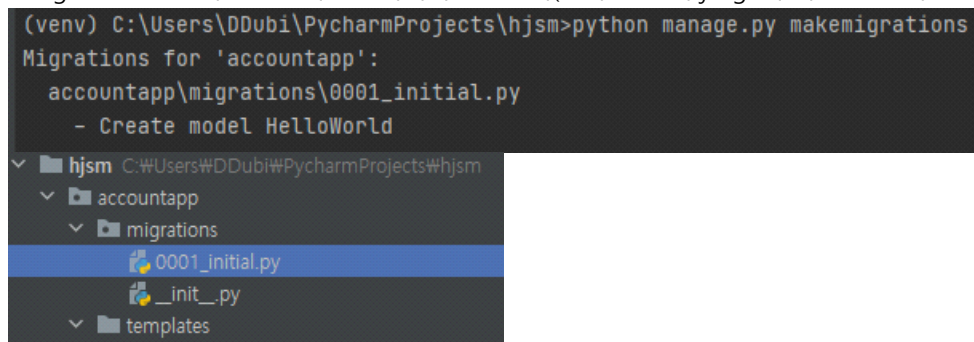
[Model]

• Model, DB 연동

- makemigrations: models.py에 작성한 내용을 DB와 연동하는python파일로 생성하는 명령어(실행 시 아래와 같이 initial.py 파일 생성)

DB와 애플리케이션을 연동해주는 역할

- migrate: 생성된 파일을 실제 연동하기위한 명령어(초기 실행 시 django의 기본 initial파일도 적용됨)



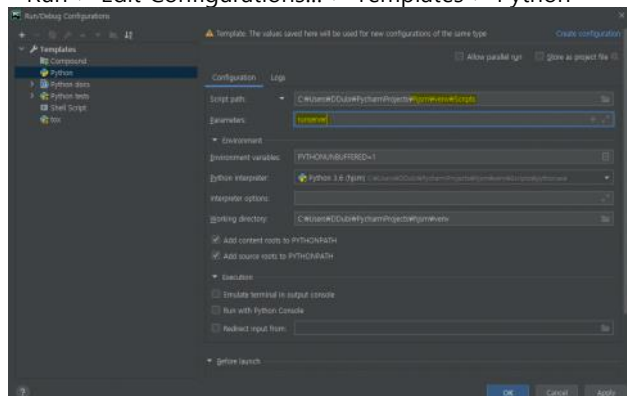
• HTTP Protocol

- GET: Inquiry; 주로 조회를 위한 정보 전달 시 사용, 주소 내부에 파라미터를 추가하여 전달
- POST: BODY 내부에 데이터를 넣어서 전달, 많은 양의 데이터 전달 시, 암호화 필요할 경우 사용
- csrf_token사용: django에서 사용하는 보안 기능. POST를 사용하기 위해 작성 필요

```
<form action="/account/hello_world/" method="post">
  {% csrf_token %}
  <input type="submit" class="btn btn-primary" value="POST">
</form>
```

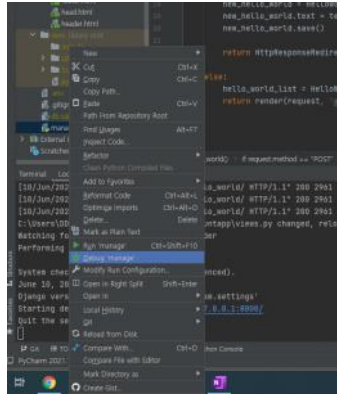
• Pycharm 디버깅 설정

- Run > Edit Configurations... > Templates > Python



- Script Path: 프로젝트/venv/Scripts
- Parameters: runserver(서버 기동 시의 명령어)

- 디버깅 시작: manage.py 우클릭 > Debug 'manage'



- Django의 CRUD, Class Based View

- Class Based View: Django는 CRUD를 간결하고 쉽게 할 수 있는 Class를 제공함.

주요 파라미터만 정의하면 CRUD 가능

(Create View, Read View, Update View, Delete View)

<-> Function Based View

Accountapp implementation

2021년 7월 5일 월요일 오후 8:49

- CreateView를 통한 회원가입 구현

- views.py

```
class AccountCreateView(CreateView):
    model = User # 장고가 제공하는 기본 유저 클래스
    form_class = UserCreationForm # model을 사용하기 위하여 장고가 기본 제공하는 form
    success_url = reverse_lazy('accountapp:hello_world') # 함수와 클래스간의 import 방식의 차이로 인해 reverse 함수 사용 불가
    template_name = 'accountapp/create.html'
```

- urls.py

```
urlpatterns = [
    path('hello_world/', hello_world, name='hello_world'), # 함수형 View
    path('create/', AccountCreateView.as_view(), name='create') # 클래스형 View
]
```

- create.html

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

{% block content %}

<div>
    <form action="{% url 'accountapp:create' %}" method="post">
        {% csrf_token %}
        {{ form }}
        <input type="submit" class="btn btn-primary">
    </form>
</div>

{% endblock %}
```

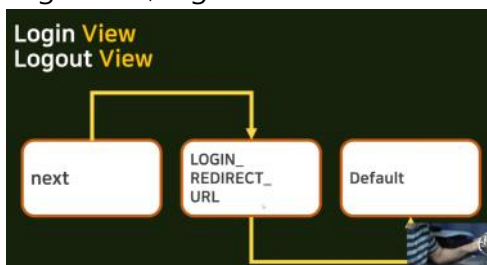
=> html 내부에 {{ form }}을 통해 view에서 지정한 form_class(UserCreationForm)을 사용할 수 있음

- Login/Logout 구현

- views.py

```
path('login/', LoginView.as_view(template_name='accountapp/login.html'), name='login'),
path('logout/', LogoutView.as_view(), name='logout'),
```

- Login view, logout view의 redirect?



next 파라미터 또는 settings 내부 LOGIN_REDIRECT_URL에 값이 없을 경우 default(<http://127.0.0.1:8000/accounts/profile/>)로 redirect

- header.html

```

<div class="hjsm_header">
  <div>
    <h1 class="hjsm_logo">2021 Django Project</h1>
  </div>
  <div>
    <span>nav1</span>
    <span>nav2</span>
    <span>nav3</span>
    {% if not user.is_authenticated %}
    <a href="{% url 'accountapp:login' %}?next={{ request.path }}">
      <span>login</span>
    </a>
    {% else %}
    <a href="{% url 'accountapp:logout' %}?next={{ request.path }}">
      <span>logout</span>
    </a>
    {% endif %}
  </div>
</div>

```

Get 방식으로 next 파라미터 전달(request 보낸 경로, 즉 이전 경로를 전달함)

- settings.py

```

LOGIN_REDIRECT_URL = reverse_lazy('accountapp:hello_world')
LOGOUT_REDIRECT_URL = reverse_lazy('accountapp:login')

```

- Bootstrap을 이용한 Form 디자인 정리

- 장고의 bootstrap4 라이브러리 활용: <https://django-bootstrap4.readthedocs.io/en/latest/>
- pip install django-bootstrap4 설치
- settings.py의 INSTALLED_APPS에 추가

```

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'bootstrap4',
    'accountapp'
]

```

- Html 내부에 {% load bootstrap4 %} 구문을 사용하여 bootstrap이 적용된 form 사용

```

{% extends 'base.html' %}
{% load bootstrap4 %}

{% block content %}

<div style="text-align:center">
  <div>
    <h4>Login</h4>
  </div>

  <div>
    <form action="" method="post">
      {% csrf_token %}
      {{ form }}
      <input type="submit" class="btn btn-primary">
    </form>
  </div>
</div>

```

- Bootstrap에서 제공하는 class 사용
- DetailView를 이용한 개인 페이지 구현
 - view.py 내부 AccountDetailView 클래스 추가
 - 특정 pk를 가진 user정보에 접근할 경우, 해당 user의 정보를 보여주기 위하여 context_object_name을 사용함 (다른 user가 해당 user의 페이지에 접근했을 경우 해당 user의 정보를 보여주기 위함)

```
class AccountDetailView(DetailView):
    model = User
    context_object_name = 'target_user'
    template_name = 'accountapp/detail.html'
```

- detail.html 파일 생성
- url.py 내부 urlpatterns 추가 (pk라는 int형 파라미터를 받음)

```
urlpatterns = [
    path('hello_world/', hello_world, name='hello_world'), # 함수형 View

    path('login/', LoginView.as_view(template_name='accountapp/login.html'), name='login'),
    path('logout/', LogoutView.as_view(), name='logout'),

    path('create/', AccountCreateView.as_view(), name='create'), # 클래스형 View
    path('detail/<int:pk>', AccountDetailView.as_view(), name='detail') # 클래스형 View
]
```

• UpdateView를 이용한 비밀번호 변경 구현

- view.py 내부 AccountUpdateView 추가

```
class AccountUpdateView(UpdateView):
    model = User # 장고가 제공하는 기본 유저 클래스
    context_object_name = 'target_user'
    form_class = AccountUpdateForm # 커스터마이징한 form
    success_url = reverse_lazy('accountapp:hello_world')
    template_name = 'accountapp/update.html'
```

- UserCreationForm를 상속받아 커스터마이징한 AccountUpdateForm을 정의 (user정보 변경 시, username 필드는 업데이트가 안되도록 하기 위함)

```
class AccountUpdateForm(UserCreationForm):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    # username 필드 disable 처리하도록 커스터마이징
    self.fields['username'].disabled = True
```

- update.html 파일 생성

```
{% extends 'base.html' %}
{% load bootstrap4 %}

{% block content %}

<div style="text-align:center; max-width:500px; margin:4rem auto;">
  <div class="mb-4">
    <h4>Change Info</h4>
  </div>
  <form action="{% url 'accountapp:update' pk=target_user.pk %}" method="post">
    {% csrf_token %}
    {{ form }}
    {% bootstrap_form form %}
    <input type="submit" class="btn btn-dark rounded-pill col-6 mt-3">
  </form>
</div>

{% endblock %}
```

- url.py 내부 urlpatterns 추가

```
urlpatterns = [
    path('hello_world/', hello_world, name='hello_world'), # 함수형 View

    path('login/', LoginView.as_view(template_name='accountapp/login.html'), name='login'),
    path('logout/', LogoutView.as_view(), name='logout'),

    path('create/', AccountCreateView.as_view(), name='create'), # 클래스형 View
    path('detail/<int:pk>', AccountDetailView.as_view(), name='detail'), # 클래스형 View
    path('update/<int:pk>', AccountUpdateView.as_view(), name='update') # 클래스형 View
]
```

• DeleteView를 이용한 회원탈퇴 구현

- view.py 내부 AccountDeleteView 추가

```
class AccountDeleteView(DeleteView):
```

```
class AccountDeleteView(DeleteView):
    model = User # 장고가 제공하는 기본 유저 클래스
    context_object_name = 'target_user'
    success_url = reverse_lazy('accountapp:login')
    template_name = 'accountapp/delete.html'
```

- url.py 내부 urlpatterns 추가

```
path('delete/<int:pk>', AccountDeleteView.as_view(), name='delete') # 클래스형 View
```

- delete.html 파일 생성

```
{% extends 'base.html' %}
{% load bootstrap4 %}

{% block content %}

<div style="text-align:center; max-width:500px; margin:4rem auto;">
  <div class="mb-4">
    <h4>Quit</h4>
  </div>
  <form action="{% url 'accountapp:delete' pk=target_user.pk %}" method="post">
    {% csrf_token %}
    <input type="submit" class="btn btn-danger rounded-pill col-6 mt-3">
  </form>
</div>

{% endblock %}
```


Authentication

2021년 8월 1일 일요일 오후 2:05

- Authentication 인증시스템 구축

- views.py 인증 여부 확인 후 미인증 시 로그인페이지 노출

```
def hello_world(request):
    # return HttpResponse('안녕하세요 HJSM의 첫번째 View입니다!!')

    if request.user.is_authenticated:
        # POST와 GET방식 분기
        if request.method == "POST":
            temp = request.POST.get('hello_world_input')

            new_hello_world = HelloWorld()
            new_hello_world.text = temp
            new_hello_world.save()

            return HttpResponseRedirect(reverse('accountapp:hello_world'))

        else:
            hello_world_list = HelloWorld.objects.all()
            return render(request, 'accountapp/helloworld.html', context={'hello_world_list': hello_world_list})

    else:
        return HttpResponseRedirect(reverse('accountapp:login'))
```

- 다른 계정 UPDATE 및 DELETE 불가하도록 get, post 함수 재정의(비추천 방식)
- self는 현재 view를 담고 있는 객체, get_object 는 request 시 받은 object(pk에 해당하는 user)
- 아래의 소스는 UPDATE/DELETE 시 계정 인증이 되었고, 요청 받은 object(user)와 현재 계정의 동일여부를 판단하여 super와 동일하게 get/post 처리하고, 아닐 경우 forbidden 페이지를 노출하는 소스임

```
class AccountUpdateView(UpdateView):
    model = User # 장고가 제공하는 기본 유저 클래스
    context_object_name = 'target_user'
    form_class = AccountUpdateForm # 커스터마이징한 form
    success_url = reverse_lazy('accountapp:hello_world') # 함수와 클래스간의 import 방식의 차이로 인해 reverse 함수 사용 불가
    template_name = 'accountapp/update.html'

    def get(self, *args, **kwargs):
        if self.request.user.is_authenticated and self.get_object() == self.request.user:
            return super().get(*args, **kwargs)
        else:
            return HttpResponseForbidden()

    def post(self, *args, **kwargs):
        if self.request.user.is_authenticated and self.get_object() == self.request.user:
            return super().get(*args, **kwargs)
        else:
            return HttpResponseForbidden()
```

- Decorator를 이용한 코드 간소화

- 특정 함수를 인자로 받아 함수의 앞/뒤에 반복되는 소스를 붙여주는 것
- @login_required 어노테이션 사용하여 구현

```

@login_required
def hello_world(request):
    # return HttpResponse('안녕하세요 HJSM의 첫번째 View입니다!!')
    # POST와 GET방식 분기
    if request.method == "POST":
        temp = request.POST.get('hello_world_input')

        new_hello_world = HelloWorld()
        new_hello_world.text = temp
        new_hello_world.save()

        return HttpResponseRedirect(reverse('accountapp:hello_world'))

    else:
        hello_world_list = HelloWorld.objects.all()
        return render(request, 'accountapp/helloworld.html', context={'hello_world_list': hello_world_list})

```

- @method_decorator(): 일반 function에 사용하는 decorator를 method에 사용할 수 있도록 변환해줌
- 아래 소스는 get, post 메서드에 login_required를 적용하는 내용(요청과 동일 user 여부 확인부는 없음)

```

@method_decorator(login_required, 'get')
@method_decorator(login_required, 'post')
class AccountUpdateView(UpdateView):
    model = User # 장고가 제공하는 기본 유저 클래스
    context_object_name = 'target_user'
    form_class = AccountUpdateForm # 커스터마이징한 form
    success_url = reverse_lazy('accountapp:hello_world')
    template_name = 'accountapp/update.html'

```

- 요청 시 전달받은 user와 현재 user가 동일한지 확인하는 decorator는 별도로 정의 필요
- decorators.py 신규 파일 생성 후, account_ownership_required decorator 정의

```

from django.contrib.auth.models import User
from django.http import HttpResponseRedirect

def account_ownership_required(func):
    def decorated(request, *args, **kwargs):
        user = User.objects.get(pk=kwargs['pk'])
        if not user == request.user:
            return HttpResponseRedirect()
        return func(request, *args, **kwargs)
    return decorated

```

- AccountUpdateView, AccountDeleteView에 적용
- 별도의 decortor 배열을 선언하여 어노테이션을 사용하여 더 간소화된 코드를 작성할 수 있음

```

has_ownership = [account_ownership_required, login_required]

```

```

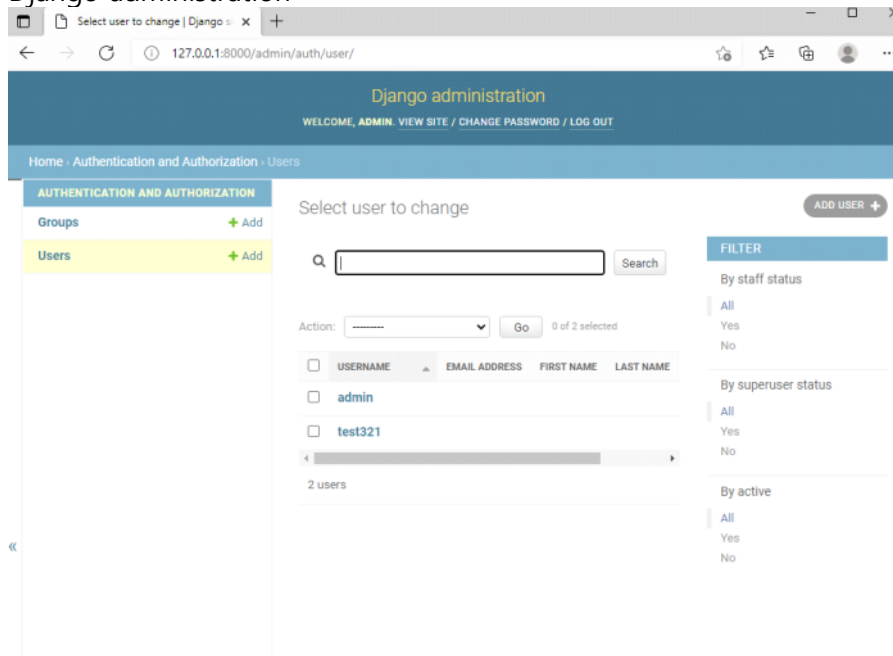
@method_decorator(has_ownership, 'get')
@method_decorator(has_ownership, 'post')
class AccountUpdateView(UpdateView):
    model = User # 장고가 제공하는 기본 유저 클래스
    context_object_name = 'target_user'
    form_class = AccountUpdateForm # 커스터마이징한 form
    success_url = reverse_lazy('accountapp:hello_world') # 함수와 클래스
    template_name = 'accountapp/update.html'

@method_decorator(has_ownership, 'get')
@method_decorator(has_ownership, 'post')
class AccountDeleteView(DeleteView):
    model = User # 장고가 제공하는 기본 유저 클래스
    context_object_name = 'target_user'
    success_url = reverse_lazy('accountapp:login')
    template_name = 'accountapp/delete.html'

```

- superuser, media 관련 설정

- Django administration



- superuser 생성

```

(venv) C:\Users\DDubi\PycharmProjects\hjsm>python manage.py createsuperuser
Username (leave blank to use 'ddubi'): admin
Email address:
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.

```

- setting.py 내 media_url, media_root 설정(media관련 설정)

- MEDIA_URL: media접근 시 사용 경로

- MEDIA_ROOT: MEDIA가 실제 저장되는 경로

```

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

```

- Django에서 이미지를 관리할때 필요한 라이브러리 설치

```
(venv) C:\Users\DDubi\PycharmProjects\hjsm>pip install pillow
Collecting pillow
  Downloading Pillow-8.3.1-1-cp36-cp36m-win_amd64.whl (3.2 MB)
    |████████████████████████████████████████| 3.2 MB 1.6 MB/s
Installing collected packages: pillow
Successfully installed pillow-8.3.1
```

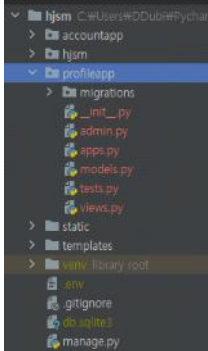
Profileapp Implementation

2021년 8월 9일 월요일 오후 8:32

- Profileapp 구현 그리고 ModelForm

- Account : Profile 은 1:1구조
- Profile은 Image, Nickname, Message 3가지로 구성
- Profile을 위한 새로운 App추가

```
(venv) C:\Users\DDubi\PycharmProjects\hjsm>python manage.py startapp profileapp
```



- Hjsm프로젝트의 Settings.py 내의 INSTALLED_APPS에 profileapp 추가

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'bootstrap4',  
    'accountapp',  
    'profileapp',  
]
```

- urls.py 파일 생성 및 Hjsm프로젝트의 Urls.py에 라우팅을 위한 urlpatterns 추가

```
hjsm\urls.py    profileapp\urls.py  
1    app_name = 'profileapp'  
2  
3    urlpatterns = [  
4    ]  
5  
6  
7  
8  
9  
10    urlpatterns = [  
11       path('admin/', admin.site.urls),  
12       path('accounts/', include('accountapp.urls')),  
13       path('profiles/', include('profileapp.urls')),  
14    ]
```

- Models.py 내 Profile 객체 정의

아래와 같이 related_name을 정의할 경우 request.user.profile.nickname와 같이 접근 가능

```
class Profile(models.Model):  
    # profile과 user객체를 연결해주는 내장 함수  
    # CASCADE user 객체 delete 시 profile도 삭제되도록  
    user = models.OneToOneField(User, on_delete=models.CASCADE, related_name='profile')  
  
    # settings.py 내 MEDIA_ROOT에서 정의한 디렉터리 하위에 해당 경로 생성 후 업로드  
    image = models.ImageField(upload_to='profile/', null=True) #nullable  
  
    nickname = models.CharField(max_length=20, unique=True, null=True) #중복방지  
    message = models.CharField(max_length=100, null=True)
```

- ModelForm: Model에 정의한 내용을 그대로 Form으로 변환해주는 기능

- forms.py 추가

```
class ProfileCreationForm(ModelForm):
    class Meta:
        model = Profile
        fields = ['image', 'nickname', 'message']
```

- DB에 반영하기
- migration파일 생성 및 마이그레이션

```
(venv) C:\Users\DDubi\PycharmProjects\hjsm>python manage.py makemigrations
Migrations for 'profileapp':
  profileapp\migrations\0001_initial.py
    - Create model Profile

(venv) C:\Users\DDubi\PycharmProjects\hjsm>python manage.py migrate
Operations to perform:
  Apply all migrations: accountapp, admin, auth, contenttypes, profileapp, sessions
Running migrations:
  Applying profileapp.0001_initial... OK
```

- views.py 내 createview 정의

```
class ProfileCreateView(CreateView):
    model = Profile
    context_object_name = 'target_profile'
    form_class = ProfileCreationForm
    success_url = reverse_lazy('accountapp:hello_world')
    template_name = 'profileapp/create.html'
```

- profileapp 하위에 create.html 파일 생성 (enctype 필수)

```
<div style="text-align:center; max-width:500px; margin:4rem auto;">
  <div class="mb-4">
    <h4>Profile Create</h4>
  </div>
  <form action="{% url 'profileapp:create' %}" method="post" enctype="multipart/form-data">
    {% csrf_token %}
    {{ form }}
    {{ bootstrap_form form }}
    <input type="submit" class="btn btn-dark rounded-pill col-6 mt-3">
  </form>
</div>
```

- urls.py 내 urlpatterns 추가

```
urlpatterns = [
    path('create/', ProfileCreateView.as_view(), name='create'),
]
```

- 아래와 같이 프로파일 속성값 사용 가능

```
{% if target_user.profile %}
  <h2 style="font-family: 'NanumSquareB'">
    {{ target_user.profile.nickname }}
  </h2>
{% else %}
  <a href="{% url 'profileapp:create' %}">
    <p>
      Create Profile
    </p>
  </a>
{% endif %}
```

- 참고) 로그인한 유저의 profile만 생성가능하도록 createView내부에 formValid 정의

```
class ProfileCreateView(CreateView):
    model = Profile
    context_object_name = 'target_profile'
    form_class = ProfileCreationForm
    success_url = reverse_lazy('accountapp:hello_world')
    template_name = 'profileapp/create.html'

    def form_valid(self, form):
        temp_profile = form.save(commit=False) #임시저장
        temp_profile.user = self.request.user #요청한 실제 유저 정보로 저장
        temp_profile.save()
        return super().form_valid(form)
```


- Profileapp 마무리

- views.py 내 updateview 정의

```
class ProfileUpdateView(UpdateView):
    model = Profile
    context_object_name = 'target_profile'
    form_class = ProfileCreationForm
    success_url = reverse_lazy('accountapp:hello_world')
    template_name = 'profileapp/update.html'
```

- urls.py 내 update urlpattern 추가

```
urlpatterns = [
    path('create/', ProfileCreateView.as_view(), name='create'),
    path('update/<int:pk>', ProfileUpdateView.as_view(), name='update'),
]
```

- profileapp 하위에 update.html 파일 생성 (enctype 필수)

```
{% extends 'base.html' %}
{% load bootstrap4 %}

{% block content %}

<div style="--: ">
    <div class="mb-4">
        <h4>Update Profile</h4>
    </div>
    <form action="{% url 'profileapp:update' pk=target_profile.pk %}" method="post" enctype="multipart/form-data">
        {% csrf_token %}
        {{ form }}
        <input type="submit" class="btn btn-dark rounded-pill col-6 mt-3">
    </form>
</div>

{% endblock %}
```

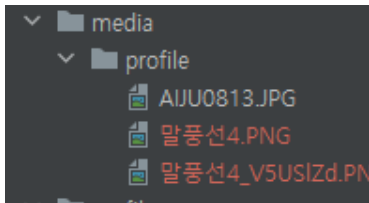
- Accountapp의 detail.html 내부에 profileapp의 update로 이동하는 edit <a>태그와 이미지 태그 추가

```
<div>
    <div style="text-align: center; max-width: 500px; margin: 4rem auto;">
        <p>
            {{ target_user.date_joined }}
        </p>
        
        {% if target_user.profile %}
        <h2 style="font-family: 'NanumSquareB'">
            {{ target_user.profile.nickname }}
            <a href="{% url 'profileapp:update' pk=target_user.profile.pk %}">
                edit
            </a>
        </h2>
        {% else %}
        <a href="{% url 'profileapp:create' %}">
            <h2 style="font-family: 'NanumSquareB'">
                Create Profile
            </h2>
        </a>
        {% endif %}
    </div>
</div>
```

- Hjsm의 urls.py 내부에 이미지 urlpatterns 추가

(이전에 설정한 MEDIA 경로 내부에 저장되는 이미지 경로에 접근 가능하도록 추가)

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('accountapp.urls')),
    path('profiles/', include('profileapp.urls')),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```



- 프로필의 nickname, message 및 image 출력



- profileapp 내 decorator.py 추가
- profile의 user가 request를 보낸 user와 동일한지 여부 체크하여 forbidden처리

```
def profile_ownership_required(func):
    def decorated(request, *args, **kwargs):
        profile = Profile.objects.get(pk=kwargs['pk'])
        if not profile.user == request.user:
            return HttpResponseForbidden()
        return func(request, *args, **kwargs)
    return decorated
```

- profileapp 내 view.py ProfileUpdateView에 decorator 적용

```
@method_decorator(profile_ownership_required, 'get')
@method_decorator(profile_ownership_required, 'post')
class ProfileUpdateView(UpdateView):
    model = Profile
    context_object_name = 'target_profile'
    form_class = ProfileCreationForm
    success_url = reverse_lazy('accountapp:hello_world')
    template_name = 'profileapp/update.html'
```

- get_success_url 함수 그리고 리팩토링

- 프로필 수정 후 완료 시 계정 detail 페이지로 이동하도록
- view.py 내 ProfileCreateView 내부에 정의된 success_url 변경 > 별도의 파라미터 전달 불가능한 문제 有
- 내부 메서드로 정의


```

class ProfileCreateView(CreateView):
    model = Profile
    context_object_name = 'target_profile'
    form_class = ProfileCreationForm
    # success_url = reverse_lazy('accountapp:hello_world')
    template_name = 'profileapp/create.html'

    def form_valid(self, form):
        temp_profile = form.save(commit=False) #임시저장
        temp_profile.user = self.request.user #요청한 실제 유저 정보로 저장
        temp_profile.save()
        return super().form_valid(form)

    def get_success_url(self):
        return reverse('accountapp:detail', kwargs={'pk': self.object.user.pk}) #self.object는 Profile

```

- 로그인 시에만 edit버튼이 보이도록 수정

```

<div>
    <div style="text-align: center; max-width: 500px; margin: 4rem auto;">

        {% if target_user.profile %}
        

        <h2 style="font-family: 'NanumSquareB'">
            {{ target_user.profile.nickname }}
            {% if target_user == user %}
                <a href="{% url 'profileapp:update' pk=target_user.profile.pk %}">
                    edit
                </a>
            {% endif %}
        </h2>
        {% else %}
            {% if target_user == user %}
                <a href="{% url 'profileapp:create' %}">
                    <h2 style="font-family: 'NanumSquareB'">
                        Create Profile
                    </h2>
                </a>
            {% else %}
                <h2>
                    닉네임 미설정
                </h2>
            {% endif %}
        {% endif %}
    </div>

```

2021 Django Project

nav1 nav2 nav3 [MyPage](#) [logout](#)

닉네임 미설정

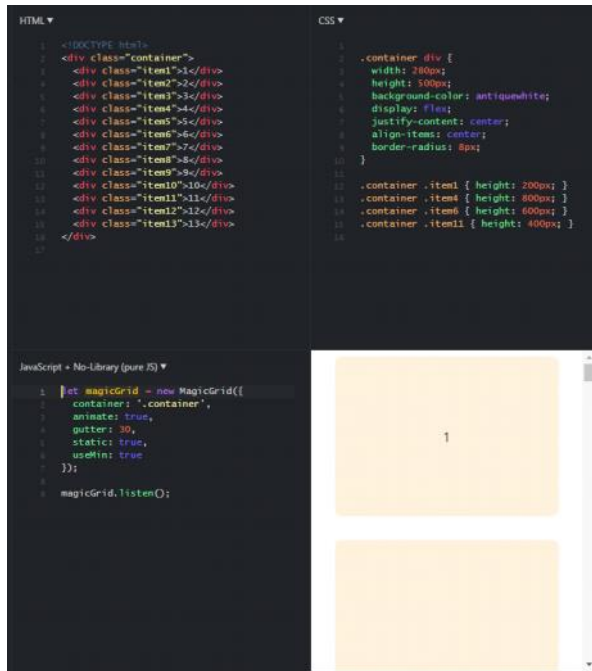
공지사항 | 제휴문의 | 서비스 소개

HJSM

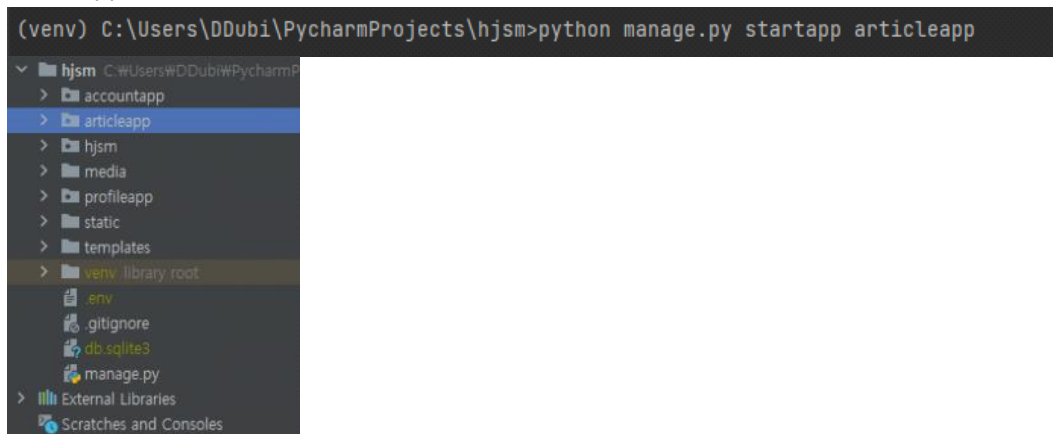
Articleapp Implementation

2021년 9월 4일 토요일 오후 3:56

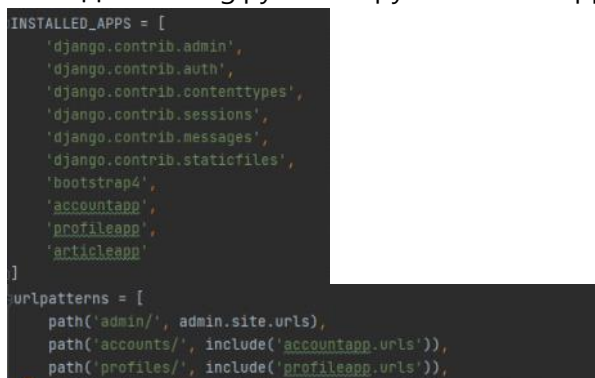
- MagicGrid 소개 및 Articleapp 시작
 - Magic Grid: Javascript 기반 카드형 동적 그리드 라이브러리
 - HTML: Container 클래스를 가진 div 내 items를 생성/활용하는 형식
 - CSS: 임의의 높이에 따라 조정 가능하도록 설정
 - JavaScript: 그리드 설정 및 listen() 함수를 통해 그리드 활성화



- articleapp 생성

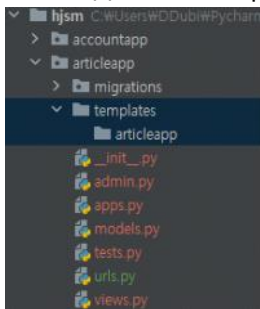


- Mainapp의 setting.py 및 urls.py 내부 articleapp관련 설정 추가

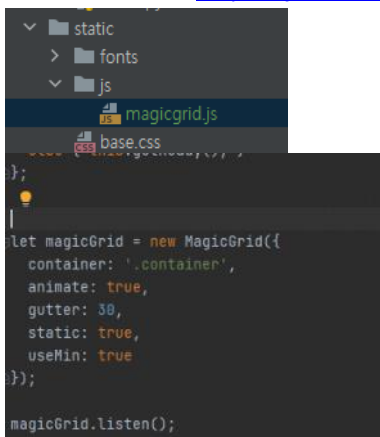


```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('accountapp.urls')),
    path('profiles/', include('profileapp.urls')),
    path('articles/', include('articleapp.urls'))
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

- articleapp 내부 urls.py 생성 및 templates>articleapp 경로 생성

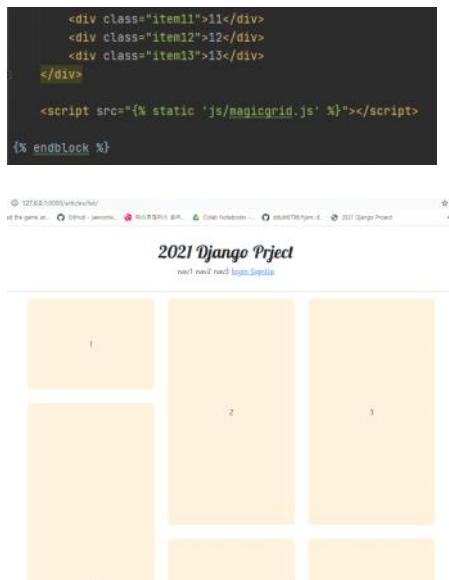


- magicgrid.js 파일 생성
- <https://github.com/e-oj/Magic-Grid/blob/master/dist/magic-grid.cjs.js> 소스 복사하여 사용
- 파일 최하단에 <https://jsfiddle.net/eolaojo/4pov0rdf/> 의 JavaScript 부를 추가하여 사용

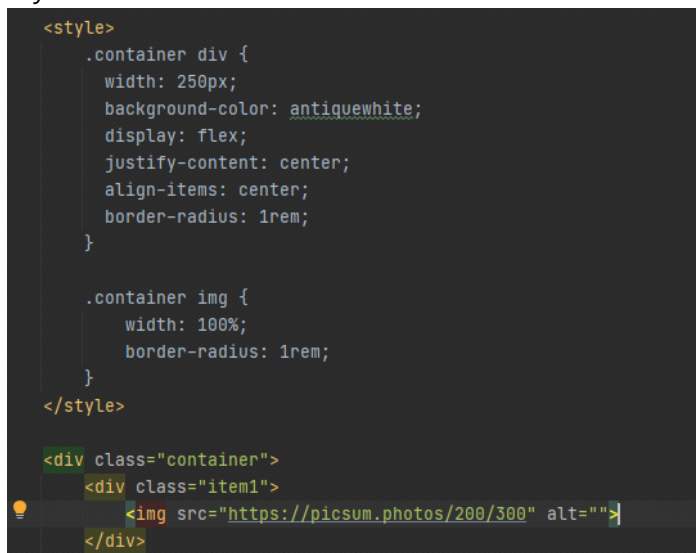


- Templates/articleapp 하위에 list.html 생성
- Magicgrid 테스트를 위한 스크립트 작성
- (<https://jsfiddle.net/eolaojo/4pov0rdf/> 의 html, css를 참고하여 div, style 작성 후 magicgrid.js import)

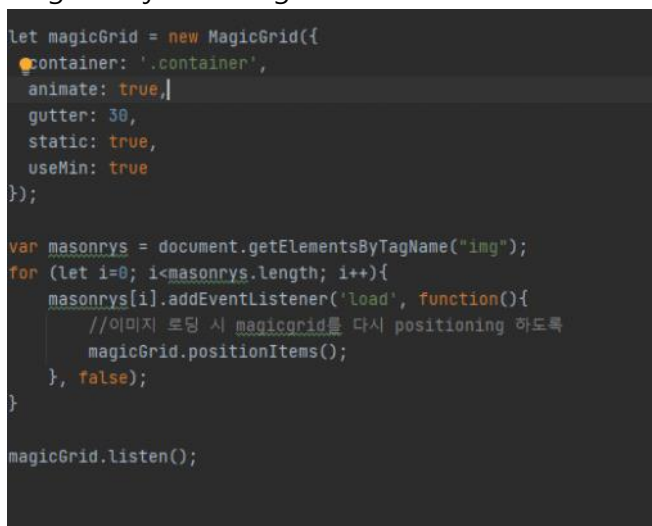


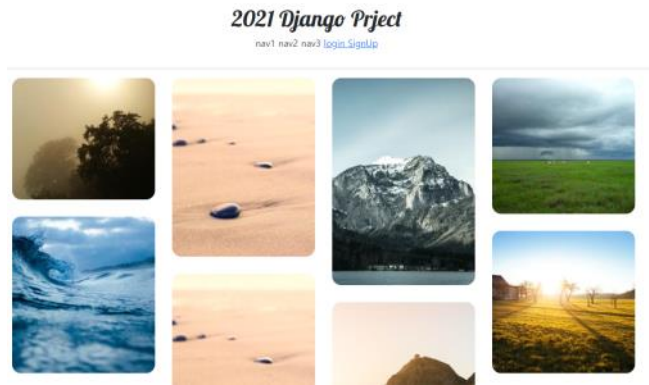


- 실제 이미지를 넣어 테스트해보기
- Picsum: 이미지를 자동 로딩 해주는 사이트(width, height을 파라미터로 전달) <https://picsum.photos/> 사용
- Style 및 div 변경



- 이미지 로딩 후 그리드 리사이징이 안되는 문제 有
- magicGrid.js 내부 img태그에 대한 load 이벤트 추가





- Articleapp 구현

- models.py 추가

```
class Article(models.Model):
    writer = models.ForeignKey(User, on_delete=models.SET_NULL, related_name='article', null=True)
    title = models.CharField(max_length=200, null=True)
    image = models.ImageField(upload_to='article/', null=False)
    content = models.TextField(null=True)

    created_at = models.DateField(auto_now_add=True, null=True) #auto_now_add: 각 저장 시 자동 생성
```

- forms.py 추가

```
class ArticleCreationForm(ModelForm):
    class Meta:
        model = Article
        fields = ['title', 'image', 'content']
```

- Migration (DB에 반영)

```
(venv) C:\Users\DDubi\PycharmProjects\hjsm>python manage.py makemigrations
Migrations for 'articleapp':
  articleapp\migrations\0001_initial.py
    - Create model Article

(venv) C:\Users\DDubi\PycharmProjects\hjsm>python manage.py migrate
Operations to perform:
  Apply all migrations: accountapp, admin, articleapp, auth, contenttypes, profileapp, sessions
Running migrations:
  Applying articleapp.0001_initial... OK
```

- views.py 내 createview, detailview 작성

```
@method_decorator(login_required, 'get')
@method_decorator(login_required, 'post')
class ArticleCreateView(CreateView):
    model = Article
    form_class = ArticleCreationForm
    template_name = 'articleapp/create.html'

    def form_valid(self, form):
        temp_article = form.save(commit=False)
        temp_article.writer = self.request.user
        temp_article.save()
        return super().form_valid(form)

    def get_success_url(self):
        return reverse('articleapp:detail', kwargs={'pk': self.object.pk})

class ArticleDetailView(DetailView):
    model = Article
    context_object_name = 'target_article'
    template_name = 'articleapp/detail.html'
```

- create.html 생성

```
{% extends 'base.html' %}
{% load bootstrap4 %}

{% block content %}

<div style="text-align:center; max-width:500px; margin:4rem auto;">
  <div class="mb-4">
    <h4>Article Create</h4>
  </div>
  <form action="{% url 'articleapp:create' %}" method="post" enctype="multipart/form-data">
    {% csrf_token %}
    {{ form }}
    <input type="submit" class="btn btn-dark rounded-pill col-6 mt-3">
  </form>
</div>

{% endblock %}
```

- detail.html 생성

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

{% block content %}

<div>
  <div style="border: 1px solid #ccc; padding: 10px;">
    <h1>
      {{ target_article.title }}
    </h1>
    
    <p>
      {{ target_article.content }}
    </p>
  </div>
</div>

{% endblock %}
```

- mainapp의 header.html 내에 articles/list로 이동하는 버튼 생성

```
<div>
  <a href="{% url 'articleapp:list' %}">
    <span>Articles</span>
  </a>
  <span>nav2</span>
  <span>nav3</span>
  {% if not user.is_authenticated %}
  <a href="{% url 'accountapp:login' %}?next={{ request.path }}">
    <span>login</span>
  </a>
</div>
```

- articles/list.html 내에 "Create Article" 버튼 생성

```
<div style="text-align: center">
  <a href="{% url 'articleapp:create' %}" class="btn btn-dark rounded-pill col-3 mt-3 mb-3">
    Create Article
  </a>
</div>
```

- views.py 내 updateView 생성

```
@method_decorator(article_ownership_required, 'get')
@method_decorator(article_ownership_required, 'post')
class ArticleUpdateView(DetailView):
    model = Article
    form_class = ArticleCreationForm
    context_object_name = 'target_article'
    template_name = 'articleapp/update.html'

    def get_success_url(self):
        return reverse('articleapp:detail', kwargs={'pk': self.object.pk})
```

- decorator 추가

```
def article_ownership_required(func):
    def decorated(request, *args, **kwargs):
        article = Article.objects.get(pk=kwargs['pk'])
        if not article.writer == request.user:
            return HttpResponseForbidden()
        return func(request, *args, **kwargs)
    return decorated
```

- detail.html 내 "Update Article" 버튼 추가


```
<a href="{% url 'articleapp:update' pk=target_article.pk %}">
  <p>Update Article</p>
</a>
```

- urls.py 내 urlpattern(delete) 추가

```
urlpatterns = [
    path('list/', TemplateView.as_view(template_name='articleapp/list.html'), name='list'),
    path('create/', ArticleCreateView.as_view(), name='create'),
    path('detail/<int:pk>', ArticleDetailView.as_view(), name='detail'),
    path('update/<int:pk>', ArticleUpdateView.as_view(), name='update'),
]
```

- update.html 생성

```
<div style="...">
  <div class="mb-4">
    <h4>Article Update</h4>
  </div>
  <form action="{% url 'articleapp:update' pk=target_article.pk %}" method="post" enctype="multipart/form-data">
    {% csrf_token %}
    {% bootstrap_form form %}
    <input type="submit" class="btn btn-dark rounded-pill col-6 mt-3">
  </form>
</div>
```

- views.py 내 deleteView 생성

```
@method_decorator(article_ownership_required, 'get')
@method_decorator(article_ownership_required, 'post')
class ArticleDeleteView(DeleteView):
    model = Article
    context_object_name = 'target_article'
    success_url = reverse_lazy('articleapp:list')
    template_name = 'articleapp/delete.html'
```

- delete.html 생성

```
<div style="...">
  <div class="mb-4">
    <h4>Delete Article : {{ target_article.title }}</h4>
  </div>
  <form action="{% url 'articleapp:delete' pk=target_article.pk %}" method="post">
    {% csrf_token %}
    <input type="submit" class="btn btn-danger rounded-pill col-6 mt-3">
  </form>
</div>
```

- detail.html 내 "Delete Article" 버튼 추가

```
<a href="{% url 'articleapp:delete' pk=target_article.pk %}">
  <p>Delete Article</p>
</a>
```

urls.py 내 urlpatterns(delete) 추가

```
urlpatterns = [
    path('list/', TemplateView.as_view(template_name='articleapp/list.html'), name='list'),
    path('create/', ArticleCreateView.as_view(), name='create'),
    path('detail/<int:pk>', ArticleDetailView.as_view(), name='detail'),
    path('update/<int:pk>', ArticleUpdateView.as_view(), name='update'),
    path('delete/<int:pk>', ArticleDeleteView.as_view(), name='delete')
]
```

- ListView, Pagination 소개 및 적용

- django의 페이징 처리 객체
- views.py 내 ArticleListView 작성

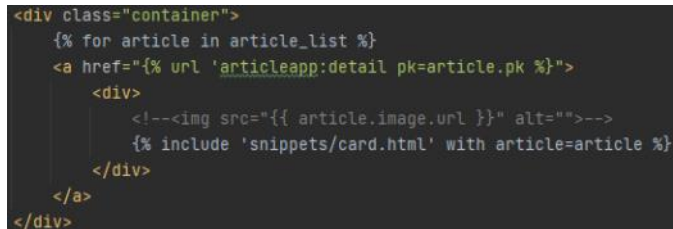
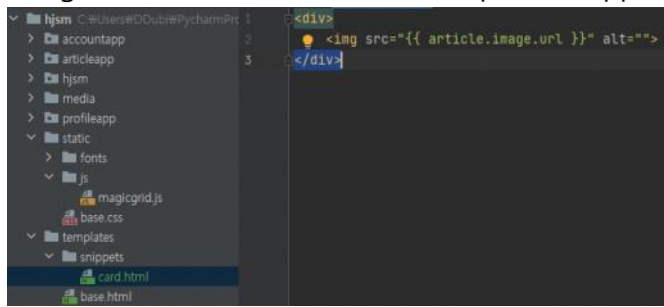
```
class ArticleListView(ListView):
    model = Article
    context_object_name = 'article_list'
    template_name = 'articleapp/list.html'
    paginate_by = 7
```

- urls.py 내 urlpatterns 추가

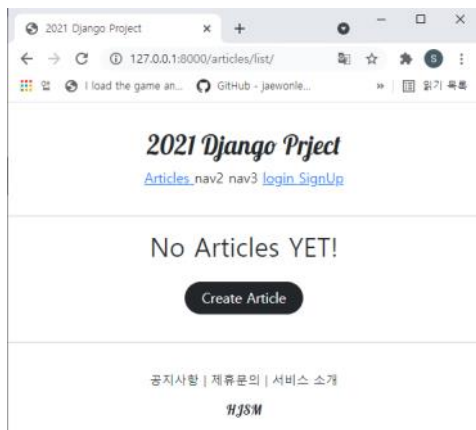
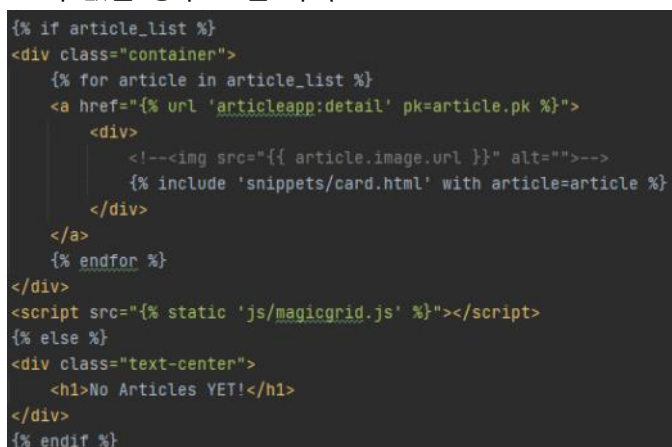
```
urlpatterns = [
    path('list/', ArticleListView.as_view(), name='list'),
    #path('list/', TemplateView.as_view(template_name='articleapp/list.html'), name='list'),
]
```

- list.html 내부 container클래스를 가진<div>태그 내용 수정(수기 작성 부 for문으로 변경)

- 태그를 별도 파일로 작성(templates/snippets/card.html)



- list가 없을 경우 조건 처리



- paging처리를 위한 pagination.html 작성 (templates/snippets/pagination.html)
- [create Article] 버튼 위에 pagination 바 생성

```
<div style="text-align: center; margin: 1rem 0;">
  {%if page_obj.has_previous %}
  <a href="?page={{ page_obj.previous_page_number }}"
    class="btn btn-secondary rounded-pill">
    {{ page_obj.previous_page_number }}
  </a>
  {% endif %}
  <a href="?page={{ page_obj.number }}"
    class="btn btn-secondary rounded-pill active">
    {{ page_obj.number }}
  </a>
  {%if page_obj.has_next %}
  <a href="?page={{ page_obj.next_page_number }}"
    class="btn btn-secondary rounded-pill">
    {{ page_obj.next_page_number }}
  </a>
  {% endif %}
</div>
```

```
{% include 'snippets/pagination.html' with page_obj=page_obj %}
```



Commentapp Implementation

2021년 9월 21일 화요일 오전 10:17

- Mixin 소개 및 Commentapp 구현

- Mixin: 클래스에 추가적인 속성이나 메소드를 제공하는 것.
- ex) CreateView에는 Form 有, Object 無 / DetailView에는 Form 無, Object 有
DetailView에 Form을 사용하고 싶을 경우? (content+comment form) - Mixin 사용!



- Commentapp의 요건: createView/DeleteView, Success_url, Model(article, writer, content, created_at)
- 신규 애플리케이션; commentapp 추가

```
(venv) C:\Users\DDubi\PycharmProjects\hjsm>python manage.py startapp commentapp
```

- parent project(hjsm)의 settings.py 내 애플리케이션 추가

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'bootstrap4',
    'accountapp',
    'profileapp',
    'articleapp',
    'commentapp'
]
```

- parent project(hjsm)의 urls.py 내 urlpattern 추가

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('accountapp.urls')),
    path('profiles/', include('profileapp.urls')),
    path('articles/', include('articleapp.urls')),
    path('comments/', include('commentapp.urls'))
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

- commentapp 애플리케이션 내 urls.py, views.py, models.py, forms.py 생성

```
app_name = 'commentapp'

urlpatterns = [
    path('create/', CommentCreateView.as_view(), name='create'),
]
```

```
class CommentCreateView(CreateView):
    model = Comment
    form_class = CommentCreationForm
    template_name = 'commentapp/create.html'

    def get_success_url(self):
        return reverse('articleapp:detail', kwargs={'pk': self.object.article.pk})
```

```
class Comment(models.Model):
    article = models.ForeignKey(Article, on_delete=models.SET_NULL, null=True, related_name='comment')
    writer = models.ForeignKey(User, on_delete=models.SET_NULL, null=True, related_name='comment')

    content = models.TextField(null=False)
    created_at = models.DateTimeField(auto_now=True)
```

```
class CommentCreationForm(ModelForm):
    class Meta:
        model = Comment
        fields = ['content']
```

- model 생성 후 migration

```
(venv) C:\Users\DDubi\PycharmProjects\hjsm>python manage.py makemigrations
Migrations for 'commentapp':
  commentapp\migrations\0001_initial.py
  - Create model Comment

(venv) C:\Users\DDubi\PycharmProjects\hjsm>python manage.py migrate
Operations to perform:
  Apply all migrations: accountapp, admin, articleapp, auth, commentapp, contenttypes, profileapp, sessions
Running migrations:
  Applying commentapp.0001_initial... OK
```

- template/commentapp/create.html 경로 및 파일 생성
- article.pk를 내부적으로 담고 있을 hidden input 추가 필요

```
{% load bootstrap4 %}

{% block content %}

<div style="text-align:center; max-width:500px; margin:4rem auto;">
  <div class="mb-4">
    <h4>Comment Create</h4>
  </div>
  <form action="{% url 'commentapp:create' %}" method="post">
    {% csrf_token %}
    {% bootstrap_form form %}
    <input type="submit" class="btn btn-dark rounded-pill col-6 mt-3">
    <input type="hidden" name="article_pk" value="{{ article.pk }}">
  </form>
</div>

{% endblock %}
```

- articleapp의 detail.html 내부에 commentapp/create.html 내용 추가

```
{% include 'commentapp/create.html' with article=target_article %}
```

- articleapp의 views.html 내 ArticleDetailView 수정(Mixin 사용)

```
class ArticleDetailView(DetailView, FormMixin):
    model = Article
    form_class = CommentCreationForm
    context_object_name = 'target_article'
    template_name = 'articleapp/detail.html'
```

- view.py의 CommentCreateView 내부에 form_valid 메소드 정의(article_pk와 writer를 전달하기 위함)

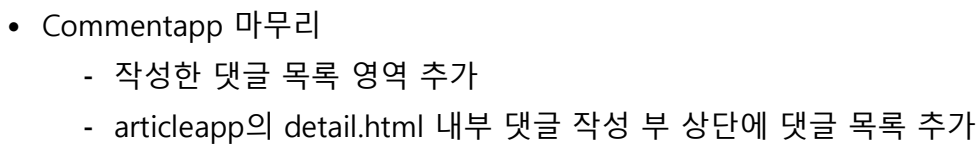
```
class CommentCreateView(CreateView):
    model = Comment
    form_class = CommentCreationForm
    template_name = 'commentapp/create.html'

    def form_valid(self, form):
        temp_comment = form.save(commit=False)
        temp_comment.article = Article.objects.get(pk=self.request.POST['article_pk'])
        temp_comment.writer = self.request.user
        temp_comment.save()
        return super().form_valid(form)

    def get_success_url(self):
        return reverse('articleapp:detail', kwargs={'pk': self.object.article.pk})
```

- ```
{% if user.is_authenticated %}
<input type="submit" class="btn btn-dark rounded-pill col-6 mt-3">
{% else %}
<a href="{% url 'accountapp:login' %}"?next={{ request.path }}"
 class="btn btn-dark rounded-pill col-6 mt-3">
 Login

{% endif %}
```



- commentapp 내 detail.html 추가

Django 프레임워크 28



- views.py 내 CommentDeleteView 추가

```
class CommentDeleteView(DeleteView):
 model = Comment
 context_object_name = 'target_comment'
 template_name = 'commentapp/delete.html'

 def get_success_url(self):
 return reverse('articleapp:detail', kwargs={'pk': self.object.article.pk})
```

- urls.py 내 urlpatterns 추가

```
urlpatterns = [
 path('create/', CommentCreateView.as_view(), name='create'),
 path('delete/<int:pk>', CommentDeleteView.as_view(), name='delete'),
]
```

- commentapp 내 detail.html 내부 우측에 delete버튼 추가

```
{% if comment.writer == user %}
<div style="text-align: right">
 <a href="{% url 'commentapp:delete' pk=comment.pk %}"
 class="btn btn-danger rounded-pill">
 Delete

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

- 삭제 확인을 위한 delete.html 생성

```
{% extends 'base.html' %}
{% load bootstrap4 %}

{% block content %}

<div style="background-color: #f0f0f0; padding: 10px; border-radius: 5px;">
 <div class="mb-4">
 <h4>Delete Comment : {{ target_comment.content }}</h4>
 </div>
 <form action="{% url 'commentapp:delete' pk=target_comment.pk %}" method="post">
 {% csrf_token %}
 <input type="submit" class="btn btn-danger rounded-pill col-6 mt-3">
 </form>
</div>

{% endblock %}
```



- commentapp/decorators.py 추가

```
def comment_ownership_required(func):
 def decorated(request, *args, **kwargs):
 comment = Comment.objects.get(pk=kwargs['pk'])
 if not comment.writer == request.user:
 return HttpResponseForbidden()
 return func(request, *args, **kwargs)
 return decorated
```

- CommentDeleteView에 decorator 적용

```
@method_decorator(comment_ownership_required, 'get')
@method_decorator(comment_ownership_required, 'post')
class CommentDeleteView(DeleteView):
```

# Mobile Responsive Layout

2021년 9월 21일 화요일 오전 11:42

- 모바일 디버깅, 반응형 레이아웃

- ALLOWED\_HOSTS: 다른 host 허용을 위한 django의 설정 값

디버깅 모드에서 ALLOWED\_HOSTS 변수가 빈 리스트일 경우 ['localhost', '127.0.0.1', ':::1'] 의미와 동일함. 즉, 로컬 호스트에서만 접속이 가능

디버깅 모드를 끄면 일체 접속이 허용되지 않고 명시적으로 지정한 호스트에만 접속 가능함.

(본 프로젝트에서는 모두 허용하도록 설정함 settings.py 내 값 설정)

```
ALLOWED_HOSTS = ['*']
```

- python manage.py runserver 0.0.0.0:8000: localhost가 아닌 외부에서 해당 서버에 접속을 위함

```
(venv) C:\Users\DDubi\PycharmProjects\hjsm>python manage.py runserver 0.0.0.0:8000
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
September 21, 2021 - 11:55:03
Django version 3.1.7, using settings 'hjsm.settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with CTRL-BREAK.
```

- 반응형 레이아웃을 위한 설정: head.html 내 meta viewport 태그 추가

(device-width: 디바이스 너비에 맞춤, initial-scale: 초기화면 배율(1=100%), shrink-to-fit=no: 사파리 브라우저 자동 줄임 방지를 위한 속성)

```
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
 <title>2021 Django Project</title>
```

- articleapp의 list.html css 수정

```
<style>
.container {
 padding: 0;
 margin: 0, auto;
}

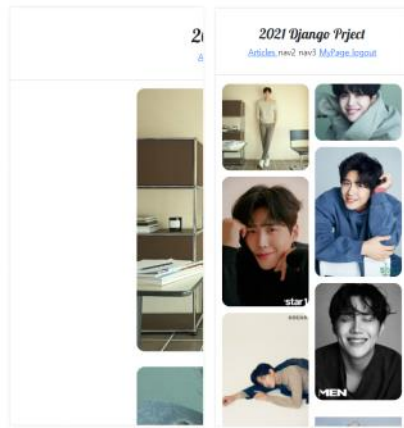
.container a {
 width: 45%;
 max-width: 250px;
}

.container div {
 display: flex;
 justify-content: center;
 align-items: center;
 border-radius: 1rem;
}
```

- magicgrid.js 수정

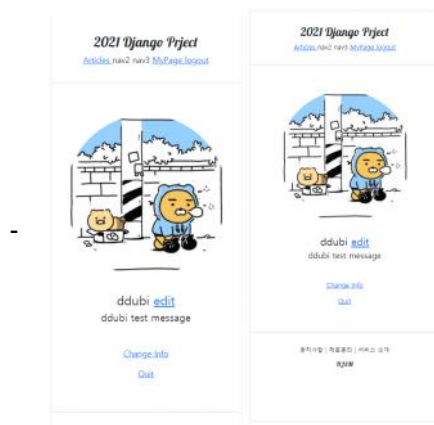
```
let magicGrid = new MagicGrid({
 container: '.container',
 animate: true,
 gutter: 12,
 static: true,
 useMin: true
});
```





- base.css 수정
- 화면 사이즈가 500px이하일 경우 적용되는 스타일 정의

```
@media screen and (max-width: 500px) {
 html {
 font-size: 13px;
 }
}
```



- Connect to WIFI local network server IP: 서버 IP로 접속해보기



# Projectapp Implementation

2021년 9월 21일 화요일    오후 1:45

- ProjectApp 구현

- 프로젝트 생성(python manage.py startapp projectapp)

```
(venv) C:\Users\DDubi\PYcharmProjects\hjsm>python manage.py startapp projectapp
```

- settings.py에 프로젝트 포함 및 urls.py에 프로젝트 urlpattern 포함

```
'projectapp'
path('projects/', include('projectapp.urls'))
```

- urls.py 생성

```
app_name = 'projectapp'

urlpatterns = [
 path('list/', ProjectListView.as_view(), name='list'),
 path('create/', ProjectCreateView.as_view(), name='create'),
 path('detail/<int:pk>', ProjectDetailView.as_view(), name='detail'),
]
```

- view.py 내부 생성

```
@method_decorator(login_required, 'get')
@method_decorator(login_required, 'post')
class ProjectCreateView(CreateView):
 model = Project
 form_class = ProjectCreationForm
 template_name = 'projectapp/create.html'

 def get_success_url(self):
 return reverse('projectapp:detail', kwargs={'pk': self.object.pk})

class ProjectDetailView(DetailView):
 model = Project
 form_class = ProjectCreationForm
 context_object_name = 'target_project'
 template_name = 'projectapp/detail.html'

class ProjectListView(ListView):
 model = Project
 context_object_name = 'project_list'
 template_name = 'projectapp/list.html'
 paginate_by = 7
```

- models.py 내부 생성

```
class Project(models.Model):
 image = models.ImageField(upload_to='project/', null=False)
 title = models.CharField(max_length=20, null=False)
 description = models.CharField(max_length=200, null=False)

 created_at = models.DateTimeField(auto_now=True)

 #프로젝트를 '프로젝트순번:프로젝트명' 형태로 나타내어 주기 위한
 def __str__(self):
 return f'{self.pk} : {self.title}'
```

- forms.py 생성

```
class ProjectCreationForm(ModelForm):
 class Meta:
 model = Project
 fields = ['image', 'title', 'description']
```

migration 작업 수행(python manage.py makemigrations python manage.py migrate)

```
(venv) C:\Users\DDubi\PcharmProjects\hjsm>python manage.py makemigrations
Migrations for 'projectapp':
 projectapp\migrations\0001_initial.py
 - Create model Project

(venv) C:\Users\DDubi\PcharmProjects\hjsm>python manage.py migrate
Operations to perform:
 Apply all migrations: accountapp, admin, articleapp, auth, commentapp, contenttypes, profileapp, projectapp, sessions
Running migrations:
 Applying projectapp.0001_initial... OK
```

- templates 작성 - create.html, detail.html, list.html 생성

```
{% extends 'base.html' %}
{% load bootstrap4 %}

{% block content %}

<div style="text-align:center; max-width:500px; margin:4rem auto;">
 <div class="mb-4">
 <h4>Create Project</h4>
 </div>
 <form action="{% url 'projectapp:create' %}" method="post" enctype="multipart/form-data">
 {% csrf_token %}
 {% bootstrap_form form %}
 <input type="submit" class="btn btn-dark rounded-pill col-6 mt-3">
 </form>
</div>

{% endblock %}
```

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

{% block content %}

<div>
 <div style="...">

 <h2 style="font-family: 'ManusSquare8'">
 {{ target_project.title }}
 </h2>
 <h5 style="...">
 {{ target_project.description }}
 </h5>
 </div>
</div>

{% endblock %}
```

```
.container div {
 display: flex;
 justify-content: center;
 align-items: center;
 border-radius: 1rem;
}

.container img {
 width: 7rem;
 height: 7rem;
 object-fit: cover;
 border-radius: 1rem;
}

</style>

{% if project_list %}
<div class="container">
 {% for project in project_list %}

 <div>
 {% include 'snippets/card_project.html' with project=project %}
 </div>

 {% endfor %}
</div>
<script src="{% static 'js/magicgrid.js' %}"></script>
{% else %}
<div class="text-center">
 <h1>No Project YET!</h1>
</div>
{% endif %}

{% include 'snippets/pagination.html' with page_obj=page_obj %}

<div style="...">

 Create Project

</div>
```

```

<div style="display: flex; justify-content: center; align-items: center; gap: 10px;">

 Create Project

</div>
{% endblock %}

```

- snippets/card\_project.html 추가

```

<div style="display: block; text-align: center;">

 <h5 class="mt-2" style="font-family: 'NanumSquareB';">
 {{ project.title | truncatechars:8 }}
 </h5>
</div>

```

- header.html project목록으로 연결되는 <a>태그 추가

```

<div class="header_navbar">

 Articles

 Projects

</div>

```



- MultipleObjectMixin을 통한 ProjectApp

- Project와 Article을 연력하는 작업을 위한 Article models.py, forms.py 수정 후 migrate

```

class Article(models.Model):
 writer = models.ForeignKey(User, on_delete=models.SET_NULL, related_name='article', null=True)
 project = models.ForeignKey(Project, on_delete=models.SET_NULL, related_name='article', null=True)

class ArticleCreationForm(ModelForm):
 class Meta:
 model = Article
 fields = ['title', 'image', 'project', 'content']

```

- Project project detail화면 내 Article 목록을 출력하는 작업
- MultipleObjectMixin 활용하여 목록 출력
- views.py 내 ProjectDetailView 수정

```

class ProjectDetailView(DetailView, MultipleObjectMixin):
 model = Project
 context_object_name = 'target_project'
 template_name = 'projectapp/detail.html'

 paginate_by = 25

```

```
class ProjectDetailView(DetailView, MultipleObjectMixin):
 model = Project
 context_object_name = 'target_project'
 template_name = 'projectapp/detail.html'

 paginate_by = 25

 def get_context_data(self, **kwargs):
 object_list = Article.objects.filter(project=self.get_object())
 return super(ProjectDetailView, self).get_context_data(object_list=object_list, **kwargs)
```

- snippets/list\_fragment.html 파일 생성

```
{% if article_list %}
<div class="container">
 {% for article in article_list %}

 <div>
 <!--
 {% include 'snippets/card.html' with article=article %}
 </div>

 {% endfor %}
</div>
<script src="{% static 'js/magicgrid.js' %}"></script>
{% else %}
<div class="text-center">
 <h1>No Articles YET!</h1>
</div>
{% endif %}

{% include 'snippets/pagination.html' with page_obj=page_obj %}

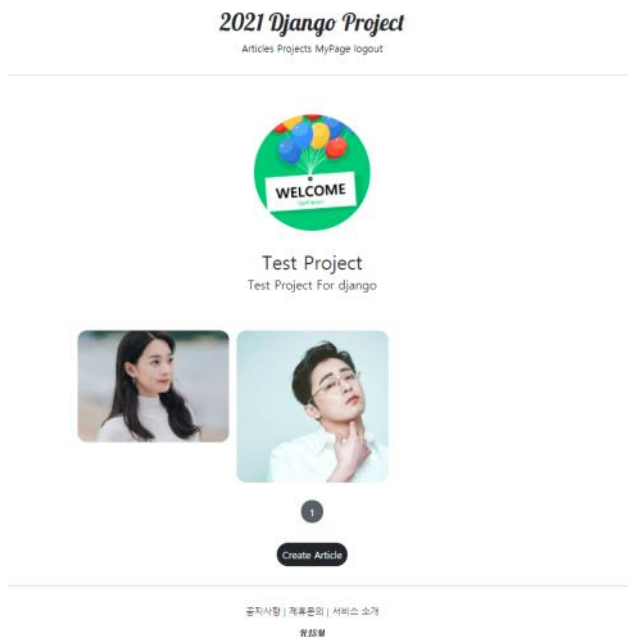
<div style="text-align: center;">

 Create Article

</div>
```

- projectapp의 detail.html 하단에 include

```
<div>
 {% include 'snippets/list_fragment.html' with article_list=object_list %}
</div>
```



- Articleapp의 detail도 동일하게 수정

```
class AccountDetailView(DetailView, MultipleObjectMixin):
 model = User
 context_object_name = 'target_user'
 template_name = 'accountapp/detail.html'

 paginate_by = 25

 def get_context_data(self, **kwargs):
 object_list = Article.objects.filter(writer=self.get_object())
 return super(AccountDetailView, self).get_context_data(object_list=object_list, **kwargs)
```

```
<div>
 {% include 'snippets/list_fragment.html' with article_list=object_list %}
</div>
```

## 2021 Django Project

Articles Projects MyPage logout



ddubi edit  
ddubi test message

Change info  
Quit



# Subscribeapp Implementation

2021년 9월 21일 화요일    오후 3:11

- RedirectView를 통한 SubscribeApp시작

- 프로젝트 생성(python manage.py startapp subscribeapp)

```
(venv) C:\Users\DDubi\PYcharmProjects\hjsm>python manage.py startapp subscribeapp
```

- settings.py에 프로젝트 포함 및 urls.py에 프로젝트 urlpattern 포함

```
'subscribeapp'
path('subscribe/', include('subscribeapp.urls'))
```

- urls.py 생성

```
app_name = 'subscribeapp'

urlpatterns = [
 path('subscribe/', SubscriptionView.as_view(), name="subscribe"),
]
```

- view.py 내부 생성

```
@method_decorator(login_required, 'get')
class SubscriptionView(RedirectView):

 def get_redirect_url(self, *args, **kwargs):
 return reverse('projectapp:detail', kwargs={'pk': self.request.GET.get('project_pk')})

 def get(self, request, *args, **kwargs):
 # 해당 project_pk를 가진 프로젝트가 없을 경우 404에러 발생
 project = get_object_or_404(Project, pk=self.request.GET.get('project_pk'))
 user = self.request.user

 # 구독 정보가 있을 경우 delete, 없을 경우 생성
 subscription = Subscription.objects.filter(user=user, project=project)
 if subscription.exists():
 subscription.delete()
 else:
 Subscription(user=user, project=project).save()
 return super(SubscriptionView, self).get(request, *args, **kwargs)
```

- models.py 내부 생성

```
class Subscription(models.Model):
 user = models.ForeignKey(User, on_delete=models.CASCADE, related_name='subscription')
 project = models.ForeignKey(Project, on_delete=models.CASCADE, related_name='subscription')

 # 사용자의 구독 Project는 단 한개만!
 class Meta:
 unique_together = ('user', 'project')
```

- migration 작업 수행(python manage.py makemigrations python manage.py migrate)

```
(venv) C:\Users\DDubi\PYcharmProjects\hjsm>python manage.py makemigrations
Migrations for 'subscribeapp':
 subscribeapp\migrations\0001_initial.py
 - Create model Subscription

(venv) C:\Users\DDubi\PYcharmProjects\hjsm>python manage.py migrate
Operations to perform:
 Apply all migrations: accountapp, admin, articleapp, auth, commentapp, contenttypes, profileapp, projectapp, sessions, subscribeapp
Running migrations:
 Applying subscribeapp.0001_initial... OK
```

- projectapp detail 내에 subscription 버튼 추가



```

<div class="text-center mb-5">
 {% if user.is_authenticated %}
 <a href="{% url 'subscribeapp:subscribe' %}?project_pk={{ target_project.pk }}"
 class="btn btn-primary rounded-pill px-3">
 Subscribe

 {% endif %}
</div>

<div>
 {% include 'snippets/list_fragment.html' with article_list=object_list %}
</div>

```

- Subscribe 버튼 수행 부 추가
- ProjectDetailView 수정

```

class ProjectDetailView(DetailView, MultipleObjectMixin):
 model = Project
 context_object_name = 'target_project'
 template_name = 'projectapp/detail.html'

 paginate_by = 25

 def get_context_data(self, **kwargs):
 project = self.object
 user = self.request.user

 # user가 로그인되어 있을 경우 구독정보 조회
 if user.is_authenticated:
 subscription = Subscription.objects.filter(user=user, project=project)
 else:
 subscription = None
 object_list = Article.objects.filter(project=self.get_object())
 return super(ProjectDetailView, self).get_context_data(object_list=object_list,
 subscription=subscription,
 **kwargs)

```

- subscribe 버튼 토글 형태로 변경
- projectapp detail.html 수정

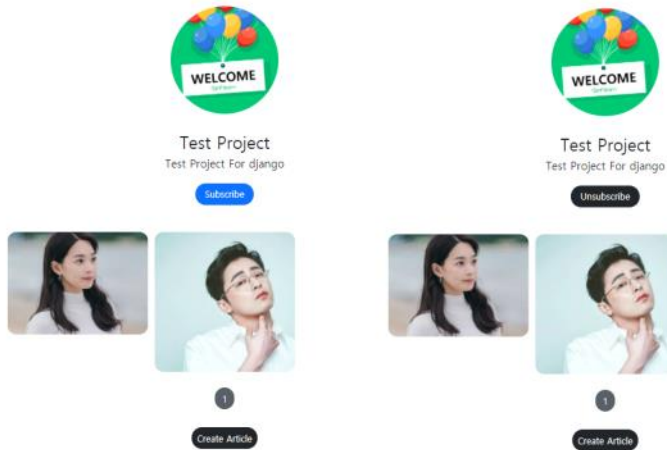
```

<div class="text-center mb-5">
 {% if user.is_authenticated %}
 {% if not subscription %}
 <a href="{% url 'subscribeapp:subscribe' %}?project_pk={{ target_project.pk }}"
 class="btn btn-primary rounded-pill px-3">
 Subscribe

 {% else %}
 <a href="{% url 'subscribeapp:subscribe' %}?project_pk={{ target_project.pk }}"
 class="btn btn-dark rounded-pill px-3">
 Unsubscribe

 {% endif %}
 {% endif %}
</div>

```



- Field Lookup을 사용한 구독 페이지 구현

- Field Lookup: django에서 제공하는 SQL의 WHERE에 해당하는 기능
- django의 documentation: <https://docs.djangoproject.com/en/3.0/ref/models/queries/#id4>
- User가 구독하고 있는 Projects 찾아 Projects에 해당하는 Articles 찾기
- view.py 내 SubscriptionListView 추가

```
@method_decorator(login_required, name='get')
class SubscriptionListView(ListView):
 model = Article
 context_object_name = 'article_list'
 template_name = 'subscribeapp/list.html'
 paginate_by = 5

 def get_queryset(self):
 projects = Subscription.objects.filter(user=self.request.user).values_list('project') #user가 구독한 프로젝트 리스트
 article_list = Article.objects.filter(project__in=projects) #user가 구독한 project 리스트에 해당하는 article
 return article_list
```

- /templates/subscribeapp/list.html 추가

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

{% block content %}

<div>
 {% include 'snippets/list_fragment.html' with article_list=article_list %}
</div>

{% endblock %}
```

- urls.py 에 list 추가

```
urlpatterns = [
 path('subscribe/', SubscriptionView.as_view(), name="subscribe"),
 path('list/', SubscriptionListView.as_view(), name="list"),
]
```

- header.html에 상단 메뉴 "Subscription" 추가

```

 Subscription

```

- user가 구독한 프로젝트에 해당하는 article목록이 보여짐





# 2021 Django Project

Articles Projects Subscription MyPage logout



1

Create Article

공지사항 | 제휴문의 | 서비스 소개

HJSM

# Django Wrap-up

2021년 10월 14일 목요일    오후 2:34

## • WYSIWYG 소개

- What You See Is What You Get의 약자로, 문서 및 문서 작성 방법을 GUI로 구현한 것
- MediumEditor: <https://github.com/yabwe/medium-editor>  
Usage

The next step is to reference the editor's script

```
<script src="js/medium-editor.js"></script>
```

You can now instantiate a new MediumEditor object:

```
<script>var editor = new MediumEditor('.editable');</script>
```

The above code will transform all the elements with the .editable class into HTML5 editable contents and add the medium editor toolbar to them.

## • articleapp에 WYSIWYG 적용하기

- forms.py 내 ArticleCreationForm 클래스 내 "editable"을 class 속성으로 가지는 textarea 추가해주기

```
class ArticleCreationForm(ModelForm):
 content = forms.CharField(widget=forms.Textarea(attrs={'class': 'editable',
 'style': 'height: auto;'}))

 class Meta:
 model = Article
```

- create.html 내부에 import 및 editor 선언

```
{% block content %}
<script src="//cdn.jsdelivr.net/npm/medium-editor@latest/dist/js/medium-editor.min.js"></script>
<link rel="stylesheet" href="//cdn.jsdelivr.net/npm/medium-editor@latest/dist/css/medium-editor.min.css" type="text/css" media="screen" charset="utf-8">

<div style="text-align:center; max-width:500px; margin:4rem auto;">
<script>var editor = new MediumEditor('.editable');</script>
{% endblock %}
```

## 2021 Django Project

Articles Projects Subscription MyPage logout

### Article Create

Title

Image

파일 선택    선택된 파일 없음

Project

Content

dfsdf

sdwkleiljsdkfi

B I U # H2 H3 "

sdf s

저장

공지사항 | 제류문의 | 서비스 소개

HJSM

- detail.html 내부 content 노출 시 filter 적용(html 태그 안보이도록 처리)

```
<p>
 {{ target_article.content | safe }}
</p>
```

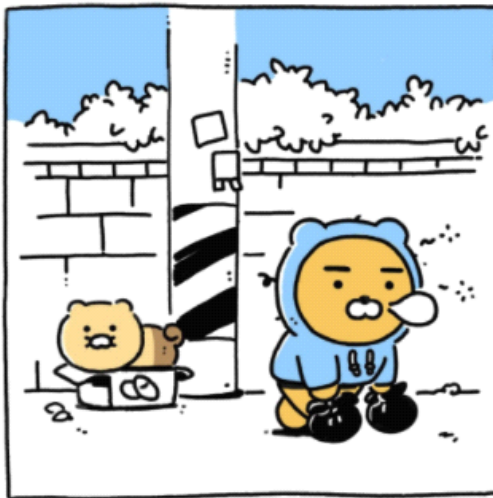
- 결과

## 2021 Django Project

Articles Projects Subscription MyPage logout

### WISIWIG 테스트

ddubi6796



WISIWIG 테스트

teststestsetsets

Update

Delete

### Comment Create

Content

Content

저출

- update.html 동일하게 editor 추가
- (추가) 기존에는 update 시 project 필수로 선택하도록함

Project

-----

목록에서 항목을 선택하세요

- project를 필수로 선택하지 않고 저장 가능하도록 수정(required=false)

```
class ArticleCreationForm(ModelForm):
 content = forms.CharField(widget=forms.Textarea(attrs={'class': 'editable text-left',
 'style': 'height: auto;'}))
 project = forms.ModelChoiceField(queryset=Project.objects.all(), required=False)
```

- (추가) <http://127.0.0.1:8000/> 로 접근했을때 default 페이지 설정
- urls.py 'home' 추가

```
urlpatterns = [
 path('', ArticleListView.as_view(), name='home'),
```

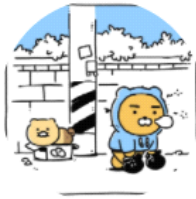
- (추가) Google Material Icons: 텍스트에 해당하는 아이콘으로 대체해주는 api
- <https://github.com/google/material-design-icons>
- head.html에 추가

```
<!-- GOOGLE MATERIAL ICONS -->
<link href="https://fonts.googleapis.com/css2?family=Material+Icons" rel="stylesheet">
```


- accountapp의 detail.html 내 icon버튼 스타일 입히기

```
<a class="material-icons"
 style="box-shadow: 0 0 4px #ccc; border-radius: 10rem; padding: .4rem;"
 href="{% url 'profileapp:update' pk=target_user.profile.pk %}">
 edit

```



-

ddubi 

ddubi test message



# Pycharm 단축키

2021년 7월 5일 월요일    오후 8:48

Alt + enter: auto import

Ctrl + b: go to definition