My Pet Community

-Report-

Prepared by

강 형 철

32190103@dankook.ac.kr
(Dept. of Mobile System Engineering)

April 12, 2023

목차

0. About this Project

이 웹페이지의 주된 기능은 “커뮤니티”입니다. 반려동물을 기르는 사람들은 물론 머지않아 반려동물을 키우게 될 사람 모두 자유롭게 이야기 나눌 수 있는 온라인 상 작은 카페입니다. 아직은 프로토타입이라 UI 및 기능이 많이 부족하지만, 차후에 계속 업데이트해 나갈 예정입니다. (※ 웹 호스팅은 진행하고 있지 않습니다.)

Source Code(Git Repository) 주소 : https://github.com/hckang17/MyPet.git

1. Main Features

1-1. Post Poster and Comment

이 서비스의 주된 기능은 커뮤니티 이기 때문에, 기본적이게 글(Poster)을 올리거나 게시글에 댓글(Comment)를 작성&수정을 포함한 여러 기능을 제공합니다.

1) Poster

가장 기본적인 ‘글쓰기’ 기능입니다. 로그인한 유저라면 누구라도 글을 작성할 수 있으며, 로그인 하지 않았을 경우 글쓰기가 제한됩니다. 또한, 자신이 작성한 게시글이라면 별도의 ‘수정하기’ 버튼이 활성화 되어 이미 작성했던 게시글을 수정할 수 있으며, 이때 수정된 날짜가 추가로 기록됩니다.

2) Comment

또 하나의 기본적인 기능인 ‘댓글’ 기능입니다. 로그인한 유저라면 누구라도 작성되어있는 게시글에 댓글을 작성할 수 있습니다. 이 기능을 통해 타 유저 혹은 자신이 작성한 게시글에 댓글을 남길 수 있습니다. 물론, 자신이 작성한 댓글이라면 별도의 ‘수정하기’ 버튼이 활성화 되어 이미 작성했던 댓글을 수정할 수 있습니다. 이 역시 또한 게시글과 같이 수정된 날짜가 추가로 기록됩니다.

3) Post List

게시글 목록을 확인할 수 있는 기능을 제공합니다. 이때, 각 게시글의 제목 우측에 해당 게시글에 남겨져있는 댓글의 수를 확인할 수 있도록 했습니다. 또한, 게시글이 특정 개수를 넘어가게 되면 다른 페이지에서 확인할 수 있도록 pagination기능도 추가하였습니다. 추가로 원하는 게시글내용 & 글쓴이 & 댓글내용을 확인할 수 있도록 검색기능도 제공합니다.

2. MVC Architecture & Design

2-1. Config

Config는 “django-admin startproject config . ”명령어로 만들어진 이 프로젝트의 최상위 디렉토리입니다. 프로젝트 설정 파일인 settings.py와 다른 Django 관련 파일들이 이곳에 포함됩니다.

1) url

```
from django.contrib importadmin
from django.urls importpath, include
from . import views

app_name='config'

urlpatterns= [
   path('admin/', admin.site.urls, name='admin'),
   path('myPet/', include('myPet.urls')),
   path('common/', include('common.urls')),
   path('', views.welcome_page, name='welcome_page'),
]
```

<표1 Config/urls.py>

Config/urls.py 는 이번 Django 어플리케이션의 URL 패턴을 정의합니다.

ⓐ path('admin/', admin.site.urls, name='admin'): 이 패턴은 'admin/'로 시작하는 URL에 대해서 Django의 관리자 사이트(admin site)로 연결됩니다. name 매개변수를 사용하여 이 URL 패턴에 대한 이름을 'admin'으로 정의했습니다. 앞으로 이 url을 호출할 때 http://127.0.0.1:8000/admin 과 같이 작성할 필요 없이 Django 문법인 {% url ’config:admin’ %} 과 같이 사용할 수 있습니다.

ⓑ path('myPet/', include('myPet.urls')): 이 패턴은 'myPet/'로 시작하는 URL에 대해서 myPet 앱의 URL 설정 파일로 연결합니다. include 함수를 사용하여 다른 URL 설정 파일을 포함시킬 수 있습니다. 이를 통해 앱 간의 URL 패턴을 모듈화하고 관리할 수 있습니다.

ⓒ path('common/', include('common.urls')): 이 패턴은 'common/'으로 시작하는 URL에 대해서 common 앱의 URL 설정 파일로 연결합니다.

ⓓ path('', views.welcome_page, name='welcome_page'): 이 패턴은 루트 URL, 즉 사이트의 기본 URL에 대해서 views.welcome_page 함수를 호출합니다. name 매개변수를 사용하여 이 URL 패턴에 대한 이름을 'welcome_page'로 정의했습니다.

2) Views

```
def welcome_page(request):
   ...
```

<표2 Config/views.py>

Config/views.py는 Config/urls.py에서 매핑된 URL 패턴에 해당되는 웹페이지를 표시하기 위해 사용됩니다.

welcomd_page(request)는 render 함수를 사용하여 welcome_page.html 템플릿을 렌더링합니다. 이때, request 객체와 템플릿 파일의 경로를 매개변수로 전달합니다. 그리고 렌더링된 결과를 클라이언트에게 반환합니다. 따라서 사용자는 http://127.0.0.1:8000/ 웹페이지를 확인할 수 있습니다.

(※ 이 파트에서는 사용한 뷰 함수의 세부구현사항은 제외하였습니다.)

2-2. MyPet

MyPet은 “django-admin startapp MyPet“으로 만들어진 이번 웹페이지의 주요 기능을 담당하는 어플리케이션입니다.

1) Urls

```
from django.urls importpath
from .views import base_views, poster_views, comment_views

app_name='mypet'

urlpatterns= [
    path('', base_views.index, name='index'),
    path('<int:poster_id>/',base_views.detail, name='detail'),
    # poster_views.py
    path('poster/create/',poster_views.poster_create, name='poster_create'),
    path('poster/modify/<int:poster_id>/',poster_views.poster_modify, name='poster_modify'),
    path('poster/delete/<int:poster_id>/',poster_views.poster_delete, name='poster_delete'),
    # comment_views.py
    path('answer/create/<int:poster_id>/',comment_views.comment_create, name='comment_create'),
    path('answer/modify/<int:comment_id>/',comment_views.comment_modify, name='comment_modify'),
    path('answer/delete/<int:comment_id>/',comment_views.comment_delete, name='comment_delete'),
]
```

<표3 MyPet/urls.py>

MyPet/urls.py 에서는 ‘게시글 작성’, ‘게시글 상세’, ‘게시글 수정’, ‘게시글 삭제’, ‘댓글 작성’, ‘댓글 수정, ’댓글 삭제‘ 등의 페이지로 연결되도록 Url Mapping 및 naming을 담당하고 있습니다.

2) Views

```
    〈 views/base_views.py 〉
def index (request):
    ...

def detail (request, poster_id):
    ...

    〈 views/comment_views.py 〉
@login_required(login_url='common:login')
def comment_create(request, poster_id):
    ...

@login_required(login_url='common:login')
def comment_modify(request, comment_id):
    ...

@login_required(login_url='common:login')
def comment_delete(request, comment_id):
    ...

    〈 views/poster_views.py 〉
@login_required(login_url='common:login')
def poster_create(request):
    ...

@login_required(login_url='common:login')
defposter_modify(request, poster_id):
    ...

@login_required(login_url='common:login')
defposter_delete(request, poster_id):
    ...
```

<표4 MyPet/views>

일반 Views.py에 모두 넣기에는 views의 나중에 수정하거나 추가기능을 넣을 때 관리하기 어려울 것 같아 base_views.py, comment_views.py, poster_views.py 세가지로 나누어 views폴더에 보관하였습니다.

ⓐ base_views.py 에서는 게시글 목록을 보여주거나 게시글 페이징을 담당하는 함수를 선언했습니다.

ⓑ comment_views.py 에서는 댓글 작성 및 수정, 삭제와 관련된 함수를 선언했습니다.

ⓒ poster_views.py 에서는 게시글 작성 및 수정, 삭제와 관련된 함수를 선언했습니다.

3) Models

```python
from django.contrib.auth.models import User
from django.db import models

# Create your models here.

class Poster(models.Model):   #게시글 class
    author=models.ForeignKey(User, on_delete=models.CASCADE, related_name='Poster_author')
    subject=models.CharField(max_length=40)
    content=models.TextField()
    create_date=models.DateTimeField()
    modify_date=models.DateTimeField(null=True, blank=True)

    def __str__(self):
      return self.subject

class Comment(models.Model):
    author=models.ForeignKey(User, on_delete=models.CASCADE, related_name='Comment_author')
    poster=models.ForeignKey(Poster, on_delete=models.CASCADE)   # 코멘트는 포스터에 연결될 예정
    content=models.TextField()
    create_date=models.DateTimeField()
    modify_date=models.DateTimeField(null=True, blank=True)
```

<표5 MyPet/models.py>

게시글과 댓글을 생성하고 수정, 관리하기 위해 Models.py에 두 객체를 정의했습니다.

ⓐ Poster 클래스는 게시글을 나타내는 모델입니다. author, subject, content, create_date, modify_date 필드로 구성되어 있습니다. author 필드는 사용자와의 외래 키 관계를 가지며, subject와 content는 각각 제목과 내용을 저장하는 필드입니다. create_date는 게시글의 생성 날짜와 시간을 저장하고, modify_date는 게시글이 수정된 날짜와 시간을 저장합니다.

ⓑ Comment 클래스는 댓글을 나타내는 모델입니다. author, poster, content, create_date, modify_date 필드로 구성되어 있습니다. author 필드는 사용자와의 외래 키 관계를 가지며, poster는 댓글이 속하는 게시글을 나타내는 외래 키 필드입니다. content는 댓글의 내용을 저장하고, create_date는 댓글의 생성 날짜와 시간을 저장하며, modify_date는 댓글이 수정된 날짜와 시간을 저장합니다.

위와같이 모델을 정의하면 Django는 이를 기반으로 데이터베이스의 테이블을 생성하고, 모델의 필드들을 테이블의 컬럼으로 매핑합니다. 이를 통해 데이터를 저장, 검색, 수정, 삭제 등의 작업을 수행할 수 있습니다.

물론, 위 모델을 사용하려면 “python manage.py makemigrations”와 “python manage.py migrate”명령어가 필요합니다.

2-3. Common

Common은 회원가입과 로그인, 회원정보 수정과 같이 User Control 역할을 주로 하는 MyPet내 어플리케이션(app)입니다.

1) Urls

```
from django.urls import path
from django.contrib.auth import views as auth_views
from common import views, admin

app_name='common'

urlpatterns= [
    path('login/', auth_views.LoginView.as_view(template_name='common/login.html'), name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
    path('signup/', views.signup, name='signup'),
    path('change_info/', views.change_info, name='change_info'),
]
```

<표6 common/urls.py>

common/urls.py 로 URL 패턴 정의를 통해 사용자가 특정 URL에 접속하면 '로그인', '로그아웃', '회원가입', '정보수정'등과 같은 기능이 실행되도록 매핑시켰습니다.

2) Views

```
def signup(request):
    ...

def change_info(request):
    ...
```

<표7 common/views.py>

coomon/views.py 에서는 이 '회원가입'과 '회원정보수정'의 기능을 담당하고있는 view를 정의했습니다. '로그인'과 '로그아웃'을 별도로 정의하지 않은 이유는, django.contrib.auth 라이브러리에 'login'과 'logout'을 담당하는 메서드가 이미 들어가 있기 때문입니다.

3. Implementation & Test Results

3-1. Implementation

0) Config/setting.py

우선, 이 서비스는 Config라는 최상위 디렉토리에서 작동하기 때문에, MyPet디렉토리의 모델을 사용하거나 Common디렉토리의 모델을 사용하기 위해서는 별도의 세팅이 필요합니다. 또한, 각 앱(app)에서 사용하는 템플렛의 최상위 template인 “base.html”에서 참조하는 static 폴더의 CSS나 부트스트랩(자바스크립트, 스타일시트) 및 template이 들어있는 폴더와 연결해주어야 합니다. 마지막으로, 로그인&로그아웃시 Redirect될 URL도 매핑해 주어야 합니다.

※ 기본적으로 설정되어있는 부분 외에 수정한 부분에 대해서만 간략하게 설명하겠습니다.

```
    ...
INSTALLED_APPS= [
    'common.apps.CommonConfig',
    'myPet.apps.MypetConfig',
]
    ...
```

① ‘common.apps.CommonConfig’ 파일은, common 앱을 django-admin으로 생성할 때 자동으로 생성되는 파일이며, 이 파일을 연결해줌으로써 Config에서 Common의 모델과 URL패턴을 등록하여 사용할 수 있습니다.
② 'myPet.apps.MypetConfig', 역시 ①과 같이 Config에 Mypet을 등록하는 과정입니다.

<Config/Setting.py의 INSTALLED_APPS>

```
    ...
STATIC_URL='static/'
STATICFILES_DIRS= [
    BASE_DIR/'static',
]
    ...
```

STATIC_URL은 웹 애플리케이션에서 정적 파일을 참조할 때 사용하는 URL을 설정합니다. static폴더 내의 이미지 요소를 가져오기 위해 설정해두었습니다.

STATICFILES_DIRS는 추가적인 정적 파일 디렉토리의 경로를 설정합니다. 여러 개의 디렉토리를 리스트로 지정할 수 있습니다. 이를 통해, static폴더 내에 있는 style.css 파일, boostrap.min.css, bootstrap.min.js 파일등을 불러와 사용할 수 있도록 했습니다.

여기서 **BASE_DIR**은 Config폴더가 있는 최상위 디렉토리입니다.

<Config/Setting.py의 STATIC_URL, STATIC_DIRS>

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

'DIRS': [BASE_DIR/'templates']는 Django 애플리케이션에서 템플릿 파일을 검색할 디렉토리를 설정합니다. [BASE_DIR/'templates']는 프로젝트의 루트 디렉토리에 있는 'templates' 디렉토리를 템플릿 파일의 검색 경로로 지정합니다. 이를 통해 Django는 해당 디렉토리에서 템플릿 파일을 찾아 사용할 수 있습니다.

<Config/Setting.py의 TEMPLATES>

```
# 로그인 성공후 이동하는 URL
LOGIN_REDIRECT_URL='/myPet'
# 로그아웃시 이동하는 URL
LOGOUT_REDIRECT_URL='/myPet
'
```

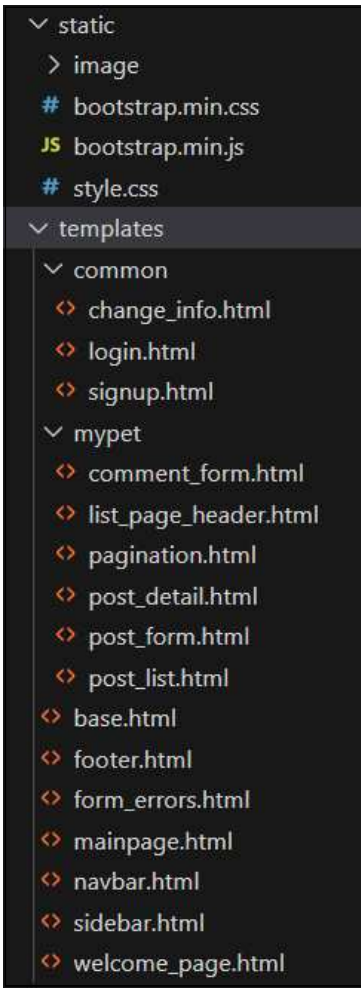
LOGIN_REDIRECT_URL은 로그인 성공 후 사용자가 이동할 URL을 설정합니다. LOGOUT_REDIRECT_URL은 로그아웃 후 사용자가 이동할 URL을 설정합니다. 각각 '/myPet'로 설정되어 있으므로, 로그인 성공 시와 로그아웃 시에 사용자는 '/myPet' URL로 이동하게 됩니다.

만약 위와같이 Redirect URL을 설정해주지 않으면 settings.LOGIN_REDIRECT_URL에 정의되어 있는 곳으로 이동하게 됩니다.

<Config/Setting.py의 REDIRECT_URL>

1) HTML templates & static

서비스가 직접적으로 제공되는 HTML 파일들과 Html을 꾸며주는 static입니다.



◆ 왼쪽의 이미지는, static폴더와 templates폴더 트리입니다.

static폴더에는 image폴더와 HTML을 꾸며줄 style.css와 bootstrap파일들이 위치하고 있습니다.

templates폴더에는 common폴더와 mypet 폴더, 그리고 몇가지의 기본 html 파일들이 위치하고 있습니다.

1) common

common 폴더에는 http://127.0.0.1:8000/common/* 에서 렌더링 될 웹페이지들이 위치하고 있으며, 모두 base.html을 상속받아 사용하고 있습니다.

2) mypet

mypet 폴더에는 http://127.0.0.1:8000/mypet/* 에서 렌더링 될 웹페이지들이 위치하고 있으며 이 또한 모두 base.html을 상속받아 사용하고 있습니다.

3) base.html

base.html은 기본적으로 html을 구성하고 있는 <head><body><footer>태그를 모두 가지고 있습니다. <head>태그에선 각 html을 꾸며줄 CSS stylesheet과 부트스트랩, <meta>태그, <script>태그가 모두 포함되어 있습니다. <body>태그 내의 세부내용은 자식 템플릿들에서 각자 정의할 수 있도록 Django문법인 {% block content %} 으로 section을 정해두었습니다. 그 외의 <header>태그는 navbar.html 파일을 참조하고, <aside>태그는 sidebar.html을 참조, <footer>태그는 footer.html 파일을 참조하도록 해 반복되는 코드의 내용을 줄였습니다. 마지막으로, {% block script %}를 통해, 각 페이지에서 사용할 자바스크립트 파일을 정의할 수 있도록 하였습니다.

4) welcome_page.html

이 html파일은, http://127.0.0.1:8000/ 에 접속하였을 때, 나타나는 페이지입니다. 이 또한 base.html을 부모 템플릿으로 두고 있습니다.

1> base.html

```
{% load static %}
<!doctype html>

<htmllang="kr">
<head>
                <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport"content="width=device-width, initial-scale=1, shrink-to-fit=no">
                <!-- Bootstrap CSS & JS -->
  <linkrel="stylesheet"type="text/css"href="{% static 'style.css' %}">
  <linkrel="stylesheet"type="text/css"href="{% static 'bootstrap.min.css' %}">
  <scriptsrc="{% static 'bootstrap.min.js' %}"></script>
                <!--  CSS -->
  <title>My Pet Community</title>
</head>
<body>
  <header>
                <!-- 네비게이션바 -->
   <navclass="navbar navbar-expand-lg navbar-light bg-light border-bottom">{% include"navbar.html" %}</nav>
  </header>
                <!-- main -->
  <sectionclass="main-section">
   {% blockcontent %}{% endblock %}
  </section>
                <!-- 사이드바-->
  <asideclass="main-aside">{% include"sidebar.html" %}</aside>
                <!-- 푸터 시작 -->
  <footer>{% include"footer.html" %}</footer>
                <!-- 자바스크립트 Start -->
  {% blockscript %}{% endblock %}
</body>
</html>
```

<표13 base.html>

ⓐ 정적 파일 로드: static 태그 라이브러리를 로드하여 정적 파일에 접근할 수 있도록 합니다.
ⓑ CSS 스타일시트 로드: style.css 파일을 참조하여 웹 페이지에 스타일을 적용합니다.
ⓒ JavaScript 파일 로드: bootstrap.min.js 파일을 참조하여 웹 페이지에 JavaScript 기능을 추가합니다.
ⓓ 페이지 제목 설정: 웹 페이지의 제목을 "My Pet Community"로 설정합니다.
ⓔ 네비게이션 바: 네비게이션 바를 렌더링하고, navbar.html 템플릿 파일을 포함하여 내용을 구성합니다.
ⓕ 주요 콘텐츠 섹션: 주요 콘텐츠를 표시하고, 다른 템플릿에서 이 영역을 확장할 수 있는 블록을 정의합니다.
ⓖ 사이드바: 사이드바를 렌더링하고, sidebar.html 템플릿 파일을 포함하여 내용을 구성합니다.
ⓗ 푸터: 푸터를 렌더링하고, footer.html 템플릿 파일을 포함하여 내용을 구성합니다.
ⓘ JavaScript 코드 블록: 각 페이지에 필요한 JavaScript 코드를 추가할 수 있는 블록을 정의합니다.

“base.html”파일은 위와같이 다른 모든 HTML파일에 공통적으로 들어가야 할 요소를 모두 선언함으로써 사용하게 되는 매 html마다 반복되는 코드를 확 줄였습니다.

2> navbar.html

```
<!-- 모든 페이지 상단에 공통으로 나타날 퀵메뉴바 -->
  <divclass="container-fluid">
   <aclass="navbar-brand"href="{% url'mypet:index' %}">My Pet!</a>
   <buttonclass="navbar-toggler"type="button"
     data-bs-toggle="collapse"
     data-bs-target="#navbarSupportedContent"
     aria-controls="navbarSupportedContent"
     aria-expanded="false"
     aria-label="Toggle navigation">
    <spanclass="navbar-toggler-icon"></span>
   </button>
   <divclass="collapse navbar-collapse"id="navbarSupportedContent">
    <ulclass="navbar-nav me-auto mb-2 mb-lg-0">
     <liclass="nav-item"><aclass="nav-link"href="{% url'welcome_page' %}">처음으로</a></li>
     {% ifuser.is_superuser %}
     <li><aclass="nav-link"href="http://127.0.0.1:8000/admin">관리자모드</a></li>
     {% endif %}
    </ul>
   </div>
   <span>
    <ulclass="navbar-nav me-auto mb-2 mb-lg-0">
     <listyle="background-position:right;">
      {% ifuser.is_authenticated %}
      <aclass="nav-link"href="{% url'common:change_info'%}">회원정보수정</a>
      {% endif %}
     </li>
    </ul>
   </span>
  </div>
```

<표14 navbar.html>

“base.html”에 포함되는 파일로, { % include ‘navbar.html’ % }에 해당하는 부분입니다.

서비스 상단의 네비게이션 바를 담당하고 있으며, 최상단페이지(default page)와, 만약 로그인 한 유저가 슈퍼유저일 경우 [관리자 모드]로 연결되는 링크버튼이 추가적으로 보이게 되며, 네비게이션바 우측에 로그인 한 유저라면 [회원정보 수정] 메뉴로 연결되도록 링크버튼을 구현했습니다. 구현과정에서 사용되었던 class는 제가 직접 커스터마이징한 style.css 파일과 bootstrap.min.css 파일을 참조하여 UI를 구성했습니다.

3> sidebar.html

```
<divstyle="line-height: 20%;">
  <br>
</div>

<divclass="container-fluid">
    {% ifnotuser.is_authenticated %}<!-- 로그인 하지 않았을 경우 -->
    <divclass="mx-auto sidebar-menubox"style="font-size: 0.7em;">
     <formmethod="post"action="{% url'common:login' %}">
     {% csrf_token %}<inputtype="hidden"name="next"value="{{ next }}"> <!-- 로그인 성공후 이동되는 URL -->
      <divclass="mb-2">
       <inputtype="text"class="sidebar-idinput form-control"name="username"id="username"placeholder="ID"
        value="{{ form.username.value|default_if_none:'' }}">
      </div>
      <divclass="mb-2">
       <inputtype="password"class="sidebar-psinput form-control"name="password"id="password"placeholder="PS"
        value="{{ form.password.value|default_if_none:'' }}">
      </div>
     <buttontype="submit"class="sidebar-login mb-1">로그인</button>
     </form>
     <ahref="{%url'common:signup'%}"><buttontype="button"class="sidebar-login">회원가입</button></a>
    </div>

    {%else%}<!-- 로그인 하였을 경우 -->
    <divclass="sidebar-userdata">
     <!-- 유저정보 표시 -->
     <divstyle="line-height:50%;">
      <br><br>
      <pstyle="text-align:center; font-size: 1.3em;">{{user.username}}</p><br>
     </div>
     <divclass="mb-2"style="text-align: center;">
      <!-- 자신 블로그로 가는 링크버튼 -->
  <ahref="https://github.com/hckang17"><buttontype="button"class="btn-sm btn-outline-secondary"style="font-size:0.7em;
display: inline-block;">내 블로그로</button></a>
       <!-- 단국대학교 포털 링크버튼 -->
   <ahref="https://portal.dankook.ac.kr/"><buttontype="button"class="btn-sm btn-outline-secondary"style="font-size:0.7em;
display: inline-block;">단국대포털</button></a>
     </div>
     <div>
      <!-- 로그아웃 버튼 -->
       <ahref="{% url'common:logout' %}"><buttontype="button"class="sidebar-login" style="font-size: 0.7em;">로그아웃
</button></a>
     </div>
     <div>
      <!-- 회원정보 수정 버튼 -->
       <ahref="{% url'common:change_info' %}"><buttontype="button"class="sidebar-login" style="font-size: 0.7em;">내
정보 수정</button></a>
     </div>
    </div>
    <hrwidth="auto"color="#797979"noshade />
    {% endif %}

</div>
```

<표15 sidebar.html>

이 코드 역시 “base.html”의 { % include ‘sidebar.html’ % }에 포함되는 파일로, 화면좌측의 <aside>사이드바를 담당하고 있습니다.

ⓐ 로그인하지 않은 경우:

ID와 비밀번호를 입력할 수 있는 입력 필드를 표시합니다. 또한 로그인 버튼과 회원가입 버튼이 있는 양식(form)을 표시합니다.

ⓑ 로그인한 경우:

유저 정보를 표시하고, 자신의 블로그(여기서는 제 깃헙(http://github.com/hckang17)로 이동하는 링크 버튼과 단국대학교 포털로 이동하는 링크 버튼을 표시합니다. 또한, 로그아웃 버튼과 회원정보 수정 버튼을 표시합니다.

ⓒ 공통부분 : 광고와 같은 <img>태그 등이 들어올 장소입니다.

4> footer.html

```
<div>
  <br>
  <span><pstyle="font-size:14px;">Developed by 32190103 강형철</p></span>
  <address>
   <pstyle="font-size:10px;">http://github.com/hckang17/MyPet</p>
  </address>
</div>
```

<표16 footer.html>

footer.html 또한 “base.html”의 {% include ‘footer.html’ %}에 포함되는 부분입니다.
개발자 정보를 담도록 했습니다.

5> post_list.html

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

<divclass="container my-3">
  <!-- 글쓰기 버튼 & 포스트 검색 버튼 블록-->
  {% include"mypet/list_page_header.html" %}
  <!-- 끝 -->
  <tableclass="table">
   <thead>
    <trclass="text-center table-dark">
    <th>번호</th>
    <thstyle="width:50%">제목</th>
    <th>글쓴이</th>
    <th>작성일시</th>
    </tr>
   </thead>
   <tbody>
    {% ifpost_list %}
    {% forposterinpost_list %}
    <trclass="text-center">
     <td>
      <!-- 번호 = 전체건수 - 시작인덱스 - 현재인덱스 + 1 -->
      {{ post_list.paginator.count|sub:post_list.start_index|sub:forloop.counter0|add:1 }}
     </td>
     <tdclass="text-start">
      <ahref="{% url'mypet:detail'poster.id %}"style="color:black;">{{ poster.subject }}</a>
      {% ifposter.comment_set.count>0 %}
      <spanclass="text-danger small mx-2">{{ poster.comment_set.count }}</span>
      {% endif %}
     </td>
     <td>{{ poster.author.username }}</td> <!-- 글쓴이 추가 -->
     <td>{{ poster.create_date }}</td>
    {% endfor %}
    </tr>

    {% else %}
    <tr>
     <tdcolspan="4">게시물이 없습니다.</td>
    </tr>
    {% endif %}
   </tbody>
  </table>

  <!-- 페이징 처리 -->
  {% include"mypet/pagination.html" %}
  <!-- 페이징 처리 끝-->

</div>
<formid="searchForm"method="get"action="{% url'mypet:index' %}">
  <inputtype="hidden"id="kw"name="kw"value="{{ kw|default_if_none:'' }}">
  <inputtype="hidden"id="page"name="page"value="{{ page }}">
</form>

{% endblock %}

{% blockscript %}
<scripttype='text/javascript'>
constpage_elements=document.getElementsByClassName("page-link");
Array.from(page_elements).forEach(function(element) {
  element.addEventListener('click', function() {
   document.getElementById('page').value=this.dataset.page;
   document.getElementById('searchForm').submit();
  });
});
constbtn_search=document.getElementById("btn_search");
btn_search.addEventListener('click', function() {
  document.getElementById('kw').value=document.getElementById('search_kw').value;
  document.getElementById('page').value=1; // 검색버튼을 클릭할 경우 1페이지부터 조회한다.
  document.getElementById('searchForm').submit();
});
</script>
{% endblock %}
```

<표16 post_list.html>

위 html파일은, “base.html”을 상속받아 사용하고 추가적으로 paging을 위하여 선언한 “mypet_filter”와

“pagination.html”, “mypet/list_page_header”을 참조하여 게시물 목록을 확인하고 검색할 수 있도록 하였습니다. 게시글은 <table> 요소안에 게시글 목록이 나타나고, 이때 게시글 작성자, 작성일시, 댓글수를 확인할 수 있도록 하였습니다. 또한, 페이지 하단에 페이징을 추가하여 테이블에는 10개의 게시글만 표시되도록 하였습니다. 그리고 각 게시글은 클릭하면 해당 게시글로 넘어갈 수 있도록 <a> 요소를 통해 링크를 걸어두었습니다.

로그인하지 않은 유저가 본 페이지에 접근할 경우, 미리 “로그인 한 사람만 글을 작성할 수 있다.”는 안내 문구를 띄우게 했습니다.

마지막으로, {% block script %}에, 이 페이지에서만 사용하는 “검색 기능”과 “페이징”을 위한 자바스크립트를 추가했습니다.

6> post_detail.html

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

<divclass="container my-3">
  <!-- message 표시 -->
  {% ifmessages %}
  <divclass="alert alert-danger my-3"role="alert">
  {% formessageinmessages %}
   <strong>{{ message.tags }}</strong>
   <ul><li>{{ message.message }}</li></ul>
  {% endfor %}
  </div>
  {% endif %}
  <!-- 질문 -->
  <h2class="border-bottom py-1">제목 : {{ poster.subject }}</h2>
  <divclass="card my-3">
   <divclass="card-body">
    <divclass="text-selectable card-text">{{ poster.content|mark }}</div>
    <divclass="d-flex justify-content-end">
     {% ifposter.modify_date %}
      <divclass="badge bg-light text-dark p-2 text-start mx-3">
       <divclass="mb-2">modified at</div>
       <div>{{ poster.modify_date }}</div>
      </div>
     {% endif %}
     <divclass="badge bg-light text-dark p-2 text-start">
      <divclass="mb-2">{{ poster.author.username }}</div>
      <div>{{ poster.create_date }}</div>
     </div>
    </div>
    <divclass="my-3">
     {% ifrequest.user==poster.author %}
     <ahref="{% url'mypet:poster_modify'poster.id %}"
     class="btn btn-sm btn-outline-secondary">수정</a>
     <ahref="javascript:void(0)"class="delete btn btn-sm btn-outline-secondary"
      data-uri="{% url'mypet:poster_delete'poster.id %}">삭제</a>
     {% endif %}
    </div>
   </div>
  </div>
  <!-- 답변 -->
```

<표17 post_detail.html (일부분)>

post_detail.html 역시 base.html을 상속받아 사용하고 있으며, block content내에 오류를 출력하는 부분과 게시글 상세내용이 bootstrap의 Card요소로 제공됩니다. 이때, 유저 == 글쓴이 일 경우 “수정”버튼과 “삭제”버튼이 활성화 됩니다. 또한, 마크다운을 적용하여 ***글제목***과 같이 작성할 경우 Bold처리되어 출력되게 하였습니다. 만약, 글이 수정되었다면 수정된 날짜를 추가로 제공합니다. (글쓴이와 수정 혹은 작성날짜는 부트스트랩의 badge요소로 제공됩니다.)

```
    <!-- 답변 -->
    <h5class="border-bottom my-3 py-2">{{poster.comment_set.count}}개의 댓글이 있습니다.</h5>
    {% forcommentinposter.comment_set.all %}
    <aid="comment_{{ comment.id }}"></a>
    <divclass="card my-3">
      <divclass="card-body">
        <divclass="text-selectable card-text">{{ comment.content|mark }}</div>
        <divclass="d-flex justify-content-end">
          {% ifcomment.modify_date %}
            <divclass="badge bg-light text-dark p-2 text-start mx-3">
              <divclass="mb-2">modified at</div>
              <div>{{ comment.modify_date }}</div>
            </div>
          {% endif %}
          <divclass="badge bg-light text-dark p-2 text-start">
            <divclass="mb-2">{{ comment.author.username }}</div>
            <div>{{ comment.create_date }}</div>
          </div>
        </div>
        <divclass="my-3">
          </a>
          {% ifrequest.user==comment.author %}
          <ahref="{% url'mypet:comment_modify'comment.id %}"
            class="btn btn-sm btn-outline-secondary">수정</a>
          <ahref="#"class="delete btn btn-sm btn-outline-secondary "
            data-uri="{% url'mypet:comment_delete'comment.id %}">삭제</a>
          {% endif %}
        </div>
      </div>
    </div>
    {% endfor %}
    <!-- 답변 등록 -->
    <formaction="{% url'mypet:comment_create'poster.id %}"method="post"class="my-3">
      {% csrf_token %}
      <!-- 오류표시 Start -->
      {% ifform.errors %}
      <divclass="alert alert-danger"role="alert">
        {% forfieldinform %}
        {% iffield.errors %}
        <div>
          <strong>{{ field.label }}</strong>
          {{ field.errors }}
        </div>
        {% endif %}
        {% endfor %}
      </div>
      {% endif %}
      <!-- 오류표시 End -->
      <divclass="mb-3">
        <labelfor="content"class="form-label">댓글내용</label>
        <textarea{% ifnotuser.is_authenticated %}disabled{% endif %}
          name="content"id="content"class="form-control"rows="10"></textarea>
      </div>
      <inputtype="submit"value="댓글등록"class="btn btn-primary">
    </form>
</div>
{% endblock %}

{% blockscript %}
<scripttype='text/javascript'>
constdelete_elements=document.getElementsByClassName("delete");
Array.from(delete_elements).forEach(function(element) {
    element.addEventListener('click', function() {
        if(confirm("정말로 삭제하시겠습니까?")) {
            location.href=this.dataset.uri;
        };
    });
});
</script>
{% endblock %}
```

<표18 post_detail.html (일부분)>

이어서, { % block content % }에 “댓글”이 모두 표시되도록 하였습니다. { % for comment in poster.comment_set.all % }로 Poster모델의 외래키 pk로 연결되어있는 Comment모델을 모두 읽어와 모든 댓글을 부트스트랩의 Card요소, Badge요소로 댓글내용, 작성 혹은 수정된 시간을 모두 표시하도록 하였습니다. 또한, 로그인한 유저에 한하여 댓글작성 폼이 활성화되어 작성할 수 있고, “삭제”를 위한 자바스크립트를 구현하였습니다. 또한, 커스텀CSS파일로 댓글이 너무 많을 경우 overflow_y = scroll; 로 오버플로우를 스크롤

가능하게 하여 화면구성요소 문제가 없도록 하였습니다.

7> post_form.html

```
{% extends'base.html' %}
{% blockcontent %}
  <!-- message 표시 -->
  {% ifmessages %}
  <divclass="alert alert-danger my-3"role="alert">
  {% formessageinmessages %}
    <strong>{{ message.tags }}</strong>
    <ul><li>{{ message.message }}</li></ul>
  {% endfor %}
  {% endif %}
  </div>
<divclass="container">

  <h5class="my-3 border-bottom pb-2">글쓰기</h5>
  <form method="post">
   {% csrf_token %}
   <!-- 오류표시 Start -->
   {% ifform.errors %}
   <divclass="alert alert-danger"role="alert">
    {% forfieldinform %}
    {% iffield.errors %}
    <div>
     <strong>{{ field.label }}</strong>
     {{ field.errors }}
    </div>
    {% endif %}
    {% endfor %}
   </div>
   {% endif %}
   <!-- 오류표시 End -->
   <divclass="mb-3">
    <labelfor="subject"class="form-label">제목</label>
    <inputtype="text"class="form-control"name="subject"id="subject"
      value="{{ form.subject.value|default_if_none:'' }}">
   </div>
   <divclass="mb-3">
    <labelfor="content"class="form-label">내용</label>
    <textareaclass="form-control"name="content"
      id="content"rows="10">{{ form.content.value|default_if_none:'' }}</textarea>
   </div>
   <div>

    <spanclass="input-group">
     <div>
      <buttontype="submit"class="btn btn-primary">등록하기</button>
      {% ifuser.is_superuser %}
      <labelfor="is-notice"class="input-group-item">
<inputstyle="margin-left:10px;vertical-align:middle;"class="form-check-input mt-0"id="is-notice"type="checkbox"value="is_notice">
공지사항
      </label>
      {% endif %}
     </div>
    </span>
   </div>
  </form>
</div>
{% endblock %}
```

<표19 post_form.html>

post_form.html 역시 “base.html”을 상속받아 사용합니다. 게시글을 작성할 수 있는 input 폼을 제공하며, 만약 유저가 “제목” 혹은 “내용” input 요소에 아무것도 작성하지 않았을 경우 “에러”를 띄우는 부분이 포함되어있습니다. “에러” 및 “경고”는 부트스트랩 alert요소를 활용하였습니다. 만약 유저==슈퍼유저 일 경우, 공지사항 체크박스가 추가로 활성화되며, 공지사항으로 지정할 수 있는 기능도 포함되어있습니다.

8) comment_form.html

```
{% extends'base.html' %}
{% blockcontent %}
<!-- 답변 수정-->
<divclass="container my-3">
  <formmethod="post">
    {% csrf_token %}
    {% include"form_errors.html" %}
    <divclass="mb-3">
      <labelfor="content"class="form-label">댓글내용</label>
      <textareaclass="form-control"name="content"id="content"
          rows="10">{{ form.content.value|default_if_none:'' }}</textarea>
    </div>
    <buttontype="submit"class="btn btn-primary">저장하기</button>
  </form>
</div>
{% endblock %}
```

<표19 comment_form.html>

comment_form.html은 이미 작성된 댓글을 수정할 때 호출되는 html템플릿입니다. 수정할 댓글을 불러와 수정할 수 있으며, 이 역시 {% include 'form_errors.html' %}을 포함하고 있기 때문에 내용이 없는 댓글을 저장할 시 오류가 발생하도록 했습니다.

9) signup.html

```
{% extends"base.html" %}
{% blockcontent %}
<divclass="container my-3">
  <formmethod="post"action="{% url'common:signup' %}">
    {% csrf_token %}
    {% include"form_errors.html" %}
    <divclass="mb-3">
      <labelfor="username">사용자 이름</label>
      <inputtype="text"class="form-control"name="username"id="username"
        value="{{ form.username.value|default_if_none:'' }}">
    </div>
    <divclass="mb-3">
      <labelfor="password1">비밀번호</label>
      <inputtype="password"class="form-control"name="password1"id="password1"
        value="{{ form.password1.value|default_if_none:'' }}">
    </div>
    <divclass="mb-3">
      <labelfor="password2">비밀번호 확인</label>
      <inputtype="password"class="form-control"name="password2"id="password2"
        value="{{ form.password2.value|default_if_none:'' }}">
    </div>
    <divclass="mb-3">
      <labelfor="email">이메일</label>
      <inputtype="text"class="form-control"name="email"id="email"
        value="{{ form.email.value|default_if_none:'' }}">
    </div>
    <buttontype="submit"class="btn btn-primary">생성하기</button>
  </form>
</div>
{% endblock %}
```

<표20 signup.html>

signup.html은 새 유저가 새롭게 회원가입을 할 수 있도록 하는 페이지를 구성하고 있습니다. Django에서 기본적으로 제공해주는 User모델을 사용하고 있으며, {% crsf_token %} 으로 보안성을 높이고 있습니다.

{% include 'form_errors.html' %}을 포함하고 있어 사용자가 제출한 비밀번호1, 비밀번호2가 같지 않거나, 너무 단순하거나, 유효하지 않은 이메일을 작성할 경우 경고 메세지를 출력하게 했습니다.

10> login.html

```
{% extends"base.html" %}
{% blockcontent %}
<divclass="container my-3">
  <formmethod="post"action="{% url'common:login' %}">
    {% csrf_token %}
    <inputtype="hidden"name="next"value="{{ next }}"> <!-- 로그인 성공후 이동되는 URL. 여기선 /myPet -->
    {% include"form_errors.html" %}
    <divclass="mb-3">
      <labelfor="username">사용자ID</label>
      <inputtype="text"class="form-control"name="username"id="username"
        value="{{ form.username.value|default_if_none:'' }}">
    </div>
    <divclass="mb-3">
      <labelfor="password">비밀번호</label>
      <inputtype="password"class="form-control"name="password"id="password"
        value="{{ form.password.value|default_if_none:'' }}">
    </div>
    <buttontype="submit"class="btn btn-primary">로그인</button>
  </form>
</div>
{% endblock %}
```

<표21 login.html>

login.html은 사용자가 login을 하기 위한 html파일입니다. signup과 같이 crsf_token으로 보안성을 높였습니다. 로그인 성공시, Config/setting.py에서 미리 설정해둔 URL인 “myPet/”으로 이동하게 됩니다.

11> change_info.html

```
{% extends"base.html" %}
{% blockcontent %}
<divclass="container my-3">
  <formmethod="post"action="{% url'common:signup' %}">
    {% csrf_token %}
    {% include"form_errors.html" %}
    <divstyle="text-align: left; font-size:1em;">
      <p>비밀번호 혹은 이메일을 변경할 수 있습니다.</p>
    </div>
    <divclass="mb-3">
      <labelfor="username">사용자 이름</label>
      <inputtype="text"class="form-control"name="username"id="username"
        value="{{ form.username.value|default_if_none:'' }}">
    </div>
    <divclass="mb-3">
      <labelfor="password1">비밀번호</label>
      <inputtype="password"class="form-control"name="password1"id="password1"
        value="{{ form.password1.value|default_if_none:'' }}">
    </div>
    <divclass="mb-3">
      <labelfor="password2">비밀번호 확인</label>
      <inputtype="password"class="form-control"name="password2"id="password2"
        value="{{ form.password2.value|default_if_none:'' }}">
    </div>
    <divclass="mb-3">
      <labelfor="email">이메일</label>
      <inputtype="text"class="form-control disabled"name="email"id="email"
        value="{{ form.email.value|default_if_none:'' }}">
    </div>
    <buttontype="submit"class="btn btn-primary">정정하기</button>
  </form>
</div>
{% endblock %}
```

<표22 change_info.html>

change_info는 사용자가 “사용자 이름”, “비밀번호”, “이메일”등의 정보를 변경하기 위한 html페이지를 제공합니다.

12> welcome_page.html

```
{% extends'base.html' %}
{% load static %}
{% block content %}

<divid="carouselExampleInterval"class="carousel slide carouselstyle"data-bs-ride="carousel">
  <divclass="carousel-inner">
   <divclass="carousel-item active"data-bs-interval="10000">
    <imgsrc="{% static 'image/puppy.png' %}"class="d-block w-100"alt="...">
   </div>
   <divclass="carousel-item"data-bs-interval="2000">
    <imgsrc="{% static 'image/cat.png' %}"class="d-block w-100"alt="...">
   </div>
   <divclass="carousel-item">
    <imgsrc="{% static 'image/bird.png' %}"class="d-block w-100"alt="...">
   </div>
  </div>
  <buttonclass="carousel-control-prev"type="button"data-bs-target="#carouselExampleInterval"data-bs-slide="prev">
   <spanclass="carousel-control-prev-icon"aria-hidden="true"></span>
   <spanclass="visually-hidden">Previous</span>
  </button>
  <buttonclass="carousel-control-next"type="button"data-bs-target="#carouselExampleInterval"data-bs-slide="next">
   <spanclass="carousel-control-next-icon"aria-hidden="true"></span>
   <spanclass="visually-hidden">Next</span>
  </button>
</div>

<tableclass="table">
  <thead>
   <trclass="text-center table-dark">
    <thstyle="width:100%">환영합니다.</th>
   </tr>
  </thead>
  <tbody>
   <trclass="text-center">
    <td><ahref="/myPet/"style="color:black;">이 링크를 클릭하면 커뮤니티로 이동합니다!</a></td>
   </tr>
  </tbody>
</table>
{% endblock %}
```

<표22 welcome_page.html>

welcome_page에서는, 사용자가 서비스 이용을 위해 최초 접속했을 때 나타나는 페이지로, 커뮤니티로 이동하기 위한 링크와, myPetCommunity인 만큼 동물 사진을 Bootstrap의 캐러샐 요소로 구성되어 있습니다. 차후에 서비스를 더 개발하게 된다면 광고 등과 같은 요소를 추가할 생각입니다. <img>의 source는 Config/setting.py에 STATIC으로 설정해둔 파일을 참조하여 사진을 가져옵니다.

2) Views

HTML을 직접 렌더링하는 각 앱(app)의 View함수를 살펴보겠습니다. View함수에서는, Django가 Html을 렌더링 할 때, 필요한 변수들을 context = {'key' : value}형태의 딕셔너리 형태의 자료구조로 넘겨받습니다. View함수는 필요에 따라 render시 context를 넘길 필요가 있습니다.

이 보고서의 Part2에서 살펴보았던 View함수에 대해 살펴보겠습니다.

1> myPet/Views/base_views.py

```
fromdjango.core.paginatorimportPaginator
fromdjango.shortcutsimportrender, get_object_or_404
fromdjango.db.modelsimportQ

from ..modelsimportPoster

defindex(request):
    page=request.GET.get('page', '1')  # page
    KeyWord=request.GET.get('kw', '')
    poster_lists=Poster.objects.order_by('create_date')
    if(KeyWord):
      poster_lists=poster_lists.filter(
          …… # models의 Q를 사용하여 제목&내용, 글쓴이를 검색하는 코드입니다.
      ).distinct()
    paginator=Paginator(poster_lists, 10)
    page_obj=paginator.get_page(page)
    context= {'post_list': page_obj, 'page_obj': page, 'kw': KeyWord }
    returnrender(request, 'mypet/post_list.html', context)

defdetail(request, poster_id):
    poster=get_object_or_404(Poster, pk=poster_id)
    context= {'poster': poster}
    returnrender(request, 'mypet/post_detail.html', context)
```

<표23 myPet/views/baseviews.py>

(길이상 설명이 불필요한 코드는 ……로 생략하였습니다.)

index 함수 에서는 page_list.html을 렌더링 할 때 필요한 context를 생성하고, post_list.html을 렌더링할 때 해당 context 딕셔너리를 넘겨줍니다. 'post_list'에는 page_obj, 'page_obj'에는 page를, 'kw'에는 KeyWord를 넘겨줍니다.

detail 함수에서는 post_list.html을 렌더링 하기 위한 context인 'poster'를 Poster모델에서 찾아 넣습니다.

2> myPet/Views/posterviews.py

```
fromdjango.contribimportmessages
fromdjango.contrib.auth.decoratorsimportlogin_required
fromdjango.shortcutsimportrender, get_object_or_404, redirect
fromdjango.utilsimporttimezone
from ..formimportPosterForm
from ..modelsimportPoster

@login_required(login_url='common:login')
defposter_create(request):
    ifrequest.method =='POST':
        form=PosterForm(request.POST)
        ifform.is_valid():  # 폼이 유효하다면
            poster=form.save(commit=False)  # 임시 저장하여 question 객체를 리턴받는다.
            poster.author =request.user
            poster.create_date =timezone.now()  # 실제 저장을 위해 작성일시를 설정한다.
            poster.save()  # 데이터를 실제로 저장한다.
            returnredirect('mypet:index')
        else:  #폼이 잘못되었을 경우
            messages.error(request,'오류가 발생했습니다.')
            form=PosterForm(request.POST)
    else:
        form=PosterForm()
    context= {'form': form}
    returnrender(request, 'mypet/post_form.html', context)

@login_required(login_url='common:login')
defposter_modify(request, poster_id):
    poster=get_object_or_404(Poster, pk=poster_id)
    ifrequest.user !=poster.author:
        messages.error(request, '수정권한이 없습니다')
        returnredirect('mypet:detail', poster_id=poster.id)
    ifrequest.method =="POST":
        form=PosterForm(request.POST, instance=poster)
        ifform.is_valid():
            poster=form.save(commit=False)    # commit = False는 임시저장
            poster.modify_date =timezone.now()  # 수정일시 저장
            poster.save()
            returnredirect('mypet:detail', poster_id=poster.id)
    else:
        form=PosterForm(instance=poster)
    context= {'form': form}
    returnrender(request, 'mypet/post_form.html', context)

@login_required(login_url='common:login')
defposter_delete(request, poster_id):
    poster=get_object_or_404(Poster, pk=poster_id)
    ifrequest.user !=poster.author:
        messages.error(request, '삭제권한이 없습니다')
        returnredirect('mypet:detail', poster_id=poster.id)
    poster.delete()
    returnredirect('mypet:index')
```

<표24 myPet/views/posterviews.py>

posterviews.py에서는 글쓰기 관련된 View함수를 담고있습니다.

1] poster_create: 로그인된 사용자만 접근할 수 있는 페이지에서 게시글을 생성하는 함수입니다.

ⓐ HTTP 요청이 POST인 경우, 폼 데이터를 검증하고 유효한 경우 게시글을 생성합니다.

ⓑ 생성된 게시글의 작성자를 현재 로그인된 사용자로 설정하고, 작성일시를 저장한 후 데이터를 저장합니다.

ⓒ 생성된 게시글의 상세 페이지로 리디렉션합니다.

ⓓ 폼이 유효하지 않은 경우 오류 메시지를 생성하고, 폼과 함께 게시글 생성 페이지를 다시 렌더링합니다.

2] poster_modify: 로그인된 사용자만 접근할 수 있는 페이지에서 게시글을 수정하는 함수입니다.

ⓐ 해당 poster_id에 해당하는 게시글을 가져옵니다.

ⓑ 현재 로그인된 사용자와 게시글의 작성자가 동일한 경우에만 수정 권한이 부여됩니다.

ⓒ HTTP 요청이 POST인 경우, 폼 데이터를 검증하고 유효한 경우 게시글을 수정합니다.

ⓓ 수정된 게시글의 수정일시를 저장한 후 데이터를 저장합니다.

ⓔ 수정된 게시글의 상세 페이지로 리디렉션합니다.

ⓕ 폼이 유효하지 않은 경우 오류 메시지를 생성하고, 폼과 함께 게시글 수정 페이지를 다시 렌더링합니다.

3] poster_delete: 로그인된 사용자만 접근할 수 있는 페이지에서 게시글을 삭제하는 함수입니다.

ⓐ 해당 poster_id에 해당하는 게시글을 가져옵니다.

ⓑ 현재 로그인된 사용자와 게시글의 작성자가 동일한 경우에만 삭제 권한이 부여됩니다.

ⓒ 게시글을 삭제합니다.

ⓓ 게시글 목록 페이지로 리디렉션합니다.

3〉 myPet/Views/commentviews.py

```
fromdjango.contribimportmessages
fromdjango.contrib.auth.decoratorsimportlogin_required
fromdjango.shortcutsimportrender, get_object_or_404, redirect, resolve_url
fromdjango.utilsimporttimezone
from ..formimportCommentForm
from ..modelsimportPoster, Comment

@login_required(login_url='common:login')
defcomment_create(request, poster_id):
    """
    게시글 답변등록
    """
    poster=get_object_or_404(Poster, pk=poster_id)

    ifrequest.method =="POST":
        form=CommentForm(request.POST)
        ifform.is_valid():
            comment=form.save(commit=False)
            comment.author =request.user  # author 속성에 로그인 계정 저장
            comment.create_date =timezone.now()
            comment.poster =poster
            comment.save()
            returnredirect('{}#comment_{}'.format(
                resolve_url('mypet:detail', poster_id=poster.id), comment.id))
    else:
        form=CommentForm()
    context= {'poster': poster, 'form': form}
    returnrender(request, 'mypet/post_detail.html', context)

@login_required(login_url='common:login')
defcomment_modify(request, comment_id):
    comment=get_object_or_404(Comment, pk=comment_id)
    ifrequest.user !=comment.author:
        messages.error(request, '수정권한이 없습니다')
        returnredirect('mypet:detail', poster_id=comment.poster.id)
    ifrequest.method =="POST":
        form=CommentForm(request.POST, instance=comment)
        ifform.is_valid():
            comment=form.save(commit=False)
            comment.modify_date =timezone.now()
            comment.save()
            returnredirect('{}#comment_{}'.format(
                resolve_url('mypet:detail', poster_id=comment.poster.id), comment.id))
    else:
        form=CommentForm(instance=comment)
    context= {'comment': comment, 'form': form}
    returnrender(request, 'mypet/comment_form.html', context)

@login_required(login_url='common:login')
defcomment_delete(request, comment_id):
    comment=get_object_or_404(Comment, pk=comment_id)
    ifrequest.user !=comment.author:
        messages.error(request, '삭제권한이 없습니다')
    else:
        comment.delete()
    returnredirect('mypet:detail', poster_id=comment.poster.id)
```

<표25 myPet/views/commentviews.py>

commentviews.py 에서는 댓글관련된 view함수들이 포함되어있습니다.

1] comment_create: 로그인된 사용자만 접근할 수 있는 페이지에서 게시글에 댓글을 생성하는 함수입니다.

ⓐ 해당 poster_id에 해당하는 게시글을 가져옵니다.

ⓑ HTTP 요청이 POST인 경우, 폼 데이터를 검증하고 유효한 경우 댓글을 생성합니다.

ⓒ 댓글의 작성자를 현재 로그인된 사용자로 설정하고, 작성일시와 관련된 정보를 저장합니다.

ⓓ 생성된 댓글이 있는 게시글의 상세 페이지로 리디렉션합니다.

2] comment_modify: 로그인된 사용자만 접근할 수 있는 페이지에서 게시글의 댓글을 수정하는 함수입니다.
ⓐ 해당 comment_id에 해당하는 댓글을 가져옵니다.
ⓑ 현재 로그인된 사용자와 댓글의 작성자가 동일한 경우에만 수정 권한이 부여됩니다.
ⓒ HTTP 요청이 POST인 경우, 폼 데이터를 검증하고 유효한 경우 댓글을 수정합니다.
ⓓ 수정된 댓글의 수정일시를 저장합니다.
ⓔ 수정된 댓글이 있는 게시글의 상세 페이지로 리디렉션합니다.

3] comment_delete: 로그인된 사용자만 접근할 수 있는 페이지에서 게시글의 댓글을 삭제하는 함수입니다.
ⓐ 해당 comment_id에 해당하는 댓글을 가져옵니다.
ⓑ 현재 로그인된 사용자와 댓글의 작성자가 동일한 경우에만 삭제 권한이 부여됩니다.
ⓒ 댓글을 삭제합니다.
ⓓ 해당 댓글이 있던 게시글의 상세 페이지로 리디렉션합니다.

4〉 Common/views.py

```
fromdjango.contrib.authimportauthenticate, login
fromdjango.shortcutsimportrender, redirect
fromcommon.formsimportUserForm

defsignup(request):
    ifrequest.method ==="POST":
        form=UserForm(request.POST)
        ifform.is_valid():
            form.save()
            username=form.cleaned_data.get('username')
            raw_password=form.cleaned_data.get('password1')
            user=authenticate(username=username, password=raw_password)  # 사용자 인증
            login(request, user)  # 로그인
            returnredirect('mypet:index')
    else:
        form=UserForm()
    returnrender(request, 'common/signup.html', {'form': form})

defchange_info(request):
    returnrender(request, 'common/change_info.html')
```

<표26 Common/views.py>

Common/views.py에서는 유저관리(회원가입, 회원정보 수정)과 같은 html을 렌더링 해주는 view함수들이 들어있습니다.

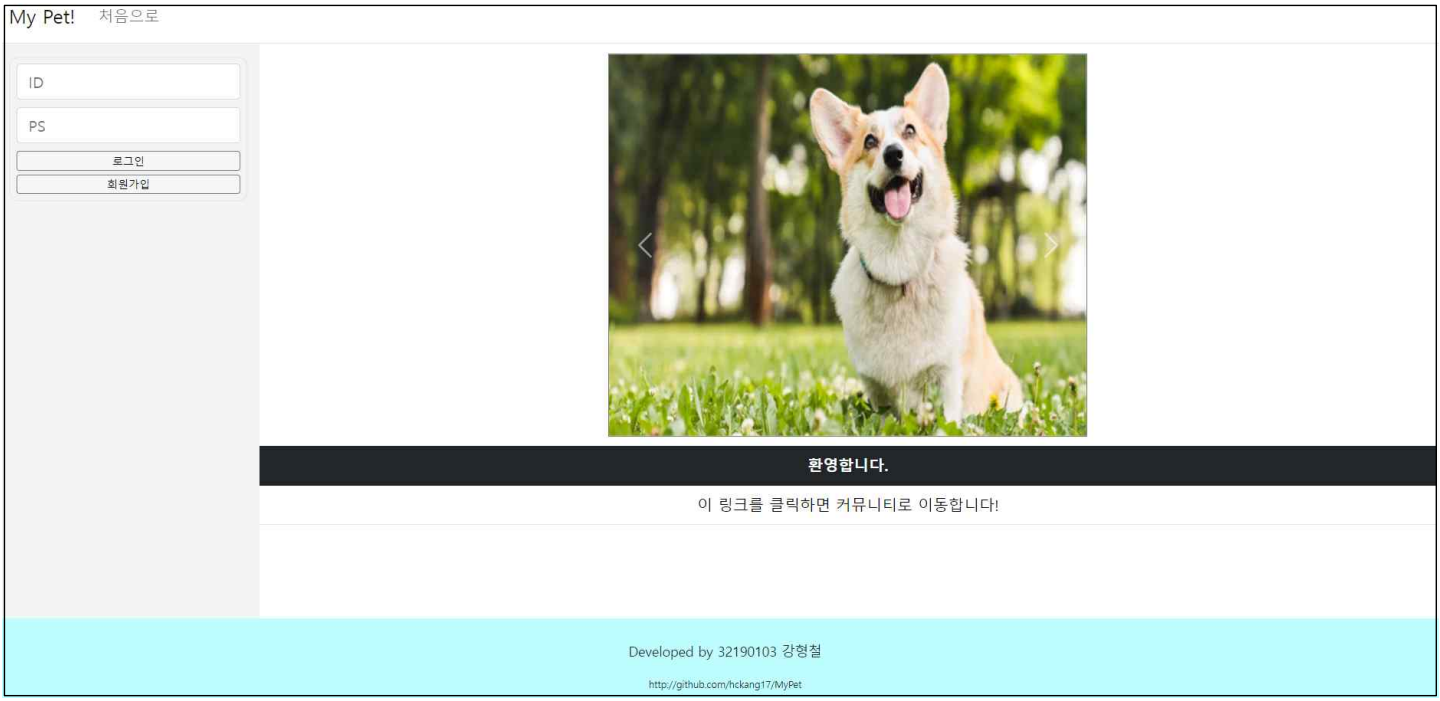
1] signup: 사용자 회원가입을 처리하는 함수입니다.
ⓐ HTTP 요청이 POST인 경우, 회원가입 폼 데이터를 검증하고 유효한 경우 사용자를 생성합니다.
ⓑ 생성된 사용자를 인증(authenticate)하고, 로그인(login)합니다.
ⓒ 회원가입 후 메인 페이지로 리디렉션합니다.

2] change_info: 회원정보 수정 페이지를 렌더링하는 함수입니다.
ⓐ common/change_info.html 템플릿을 렌더링하여 해당 페이지를 보여줍니다.

3-2. Test Results

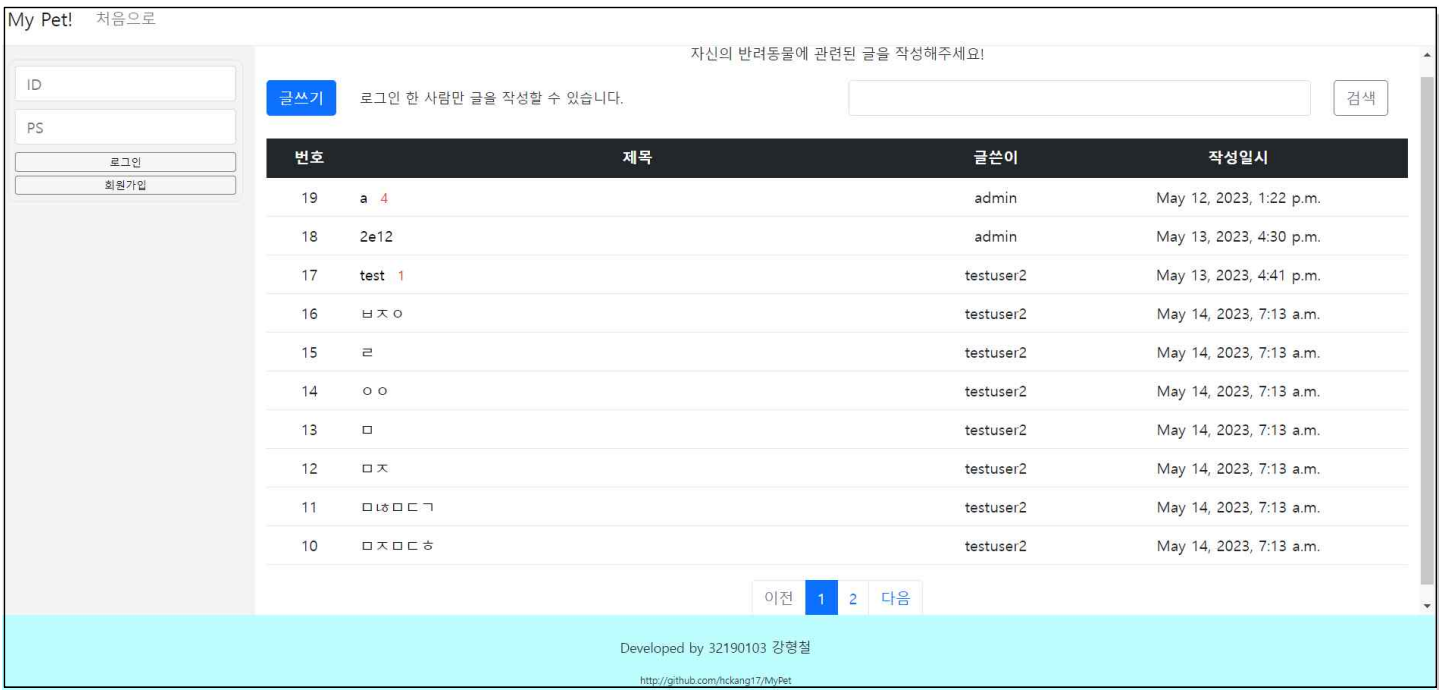
위 코드를 바탕으로 'python manage.py runserver'을 실행해보겠습니다.

1) http://127.0.0.1:8000/



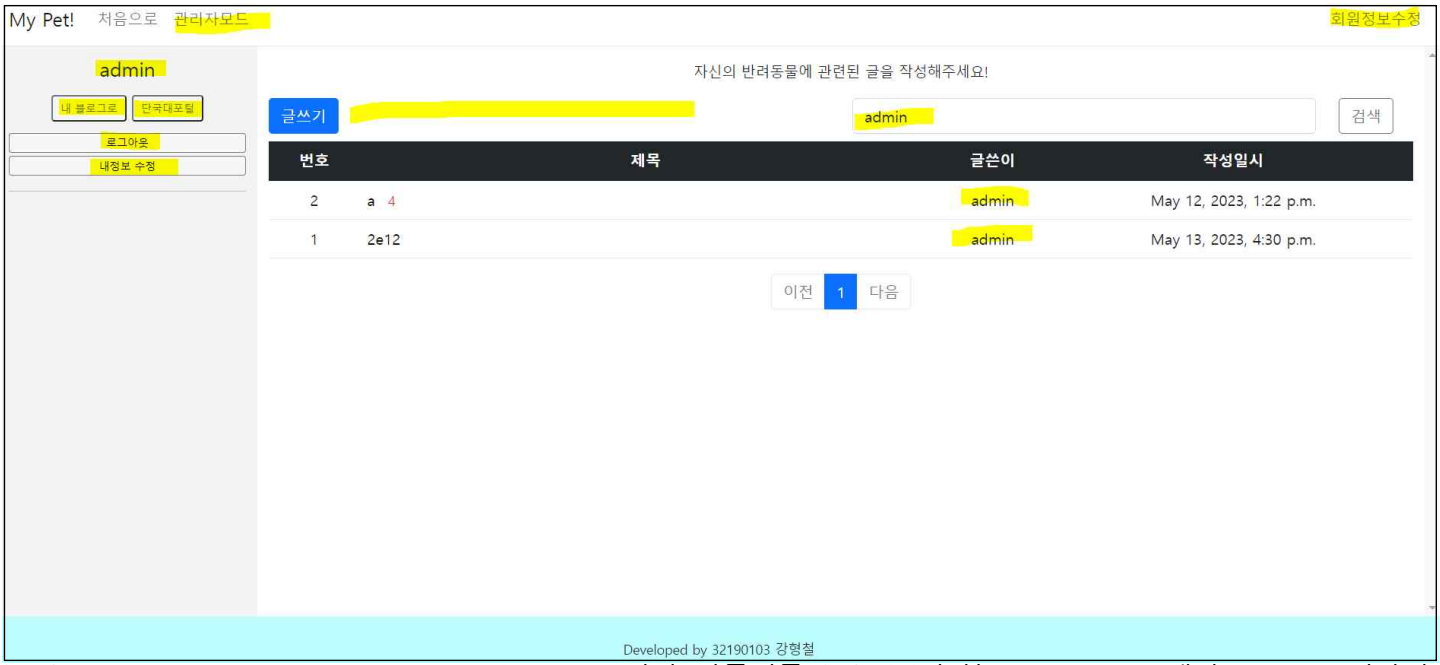
Config/urls.py 의 path('', views.welcome_page, name='welcome_page') 의 코드가 정상적으로 작동하여 웰컴화면이 나타나는 것을 확인할 수 있습니다. 또한, "base.html"에 포함되어있는 <aside>,<footer>,<header>가 모두 잘 나타나있는 것을 확인할 수 있습니다.

2) http://127.0.0.1:8000/myPet/



여기서, "유저가 로그인하지 않았기 때문에" 상단 네비게이션 바에 "처음으로" 버튼만 활성화 되어있고, "로그인 한 사람만 글을 작성할 수 있습니다." 문구가 활성화 되어있는 것을 확인할 수 있고, 테이블에는 10개의 글만 표시되어 있고 paging요소로 [1],[2] 페이지와 <이전><다음> 버튼이 활성화 되어있는 것을 확인할 수 있습니다.

3) http://127.0.0.1:8000/myPet/



'python manage.py createsuperuser' 코드로 미리 만들어둔 'admin'이라는 superuser 계정으로 로그인하여, [admin]이라는 키워드로 검색을 해 보았습니다. 형광펜으로 색칠해둔 곳이 2)에서와 달라진 점입니다.

ⓐ 사이드바

사이드바의 "로그인 창"이 사라지고, [유저이름=admin]과 [내블로그],[단국대포털],[로그아웃],[내정보 수정]버튼이 활성화 된 것을 확인할 수 있습니다.

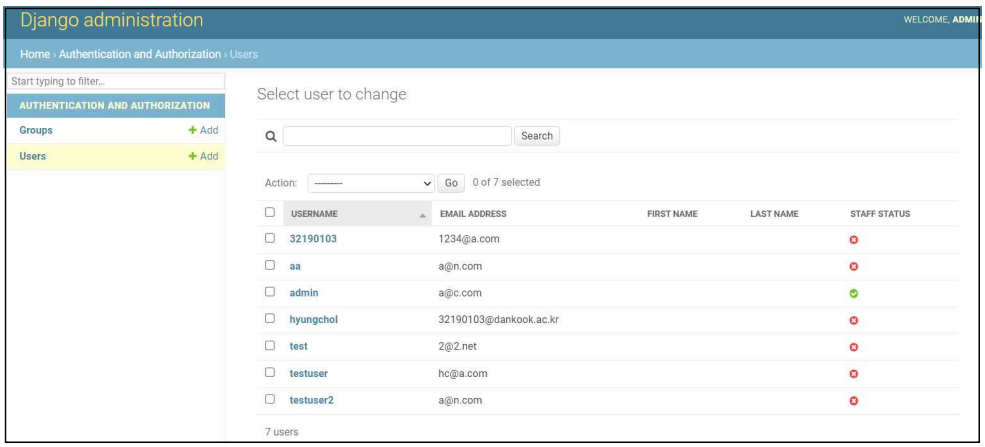
ⓑ 네비게이션바

네비게이션바에 user.is_superuser 이 True이기 때문에, [관리자모드] 버튼이 활성화 되었있습니다. 또한, 슈퍼유저 또한 User이기 때문에 오른쪽에 "회원정보수정" 버튼이 활성화 된 것을 확인할 수 있습니다.

ⓒ 본문(block content)

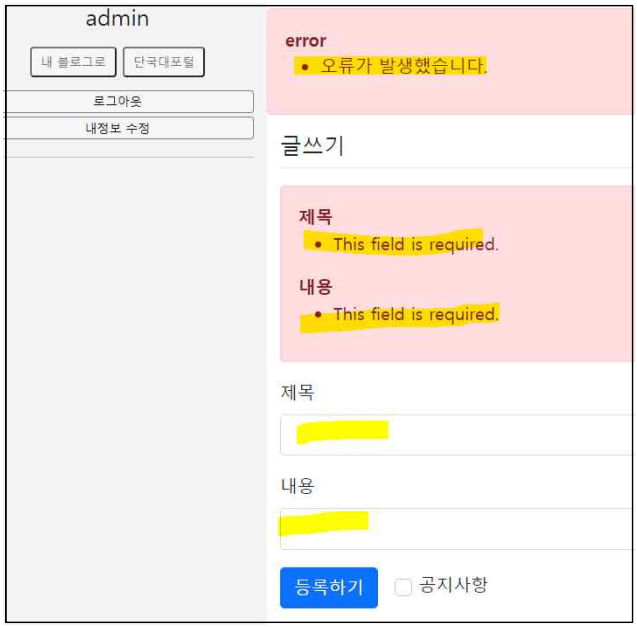
유저가 로그인 하였기 때문에 "로그인 한 사람만 글을 작성할 수 있습니다." 문구가 사라진 것을 확인할 수 있습니다. 또한, Keyword = admin 으로 검색한 결과로, 글쓴이 혹은 게시글내용 혹은 댓글내용 중에 "admin"을 포함하는 모들 게시글이 테이블에 나타났습니다.

4) http://127.0.0.1:8000/admin/auth/user/



[관리자 모드] 버튼을 열어, 생성된 관리자 화면에 User관리페이지입니다. 지금까지 만든 test계정과 admin계정을 포함한 모든 계정이 나타나는 것을 확인할 수 있었습니다.

5) http://127.0.0.1:8000/myPet/poster/create/

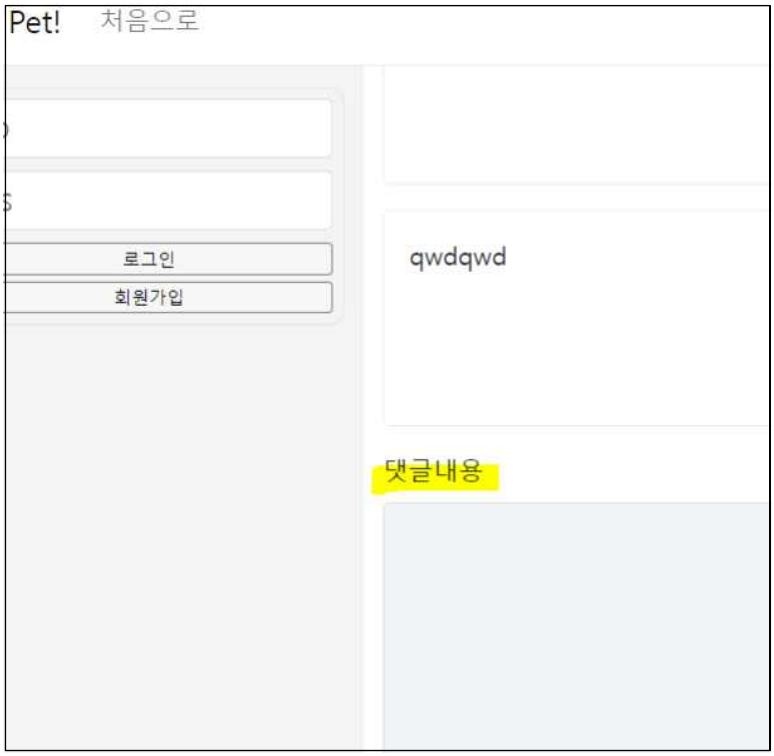


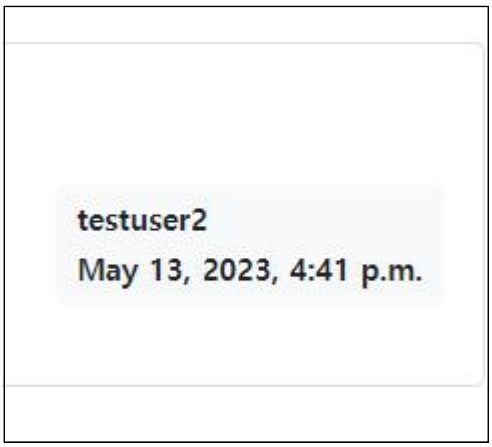
user.is_authenticated 상태이기 때문에 [글쓰기] 버튼을 클릭하면 게시글을 작성할 수 있는 페이지로 넘어갈 수 있었습니다. 이때, 추가적으로 user.is_superuser이기 때문에 [공지사항] 체크박스도 활성화 되어있는 것을 확인할 수 있었습니다. 또한, 아무것도 입력하지 않았을 경우 form_errors.html 이 활성화 되었음을 확인할 수 있었습니다.

6) http://127.0.0.1:8000/myPet/?kw=&page=2

[등록하기]버튼을 통해 글을 등록한 이후, [2] 버튼을 클릭해 Poster모델이 정상적으로 저장되었고, 또 페이징 작업까지 완료되어 있음을 확인할 수 있었습니다.

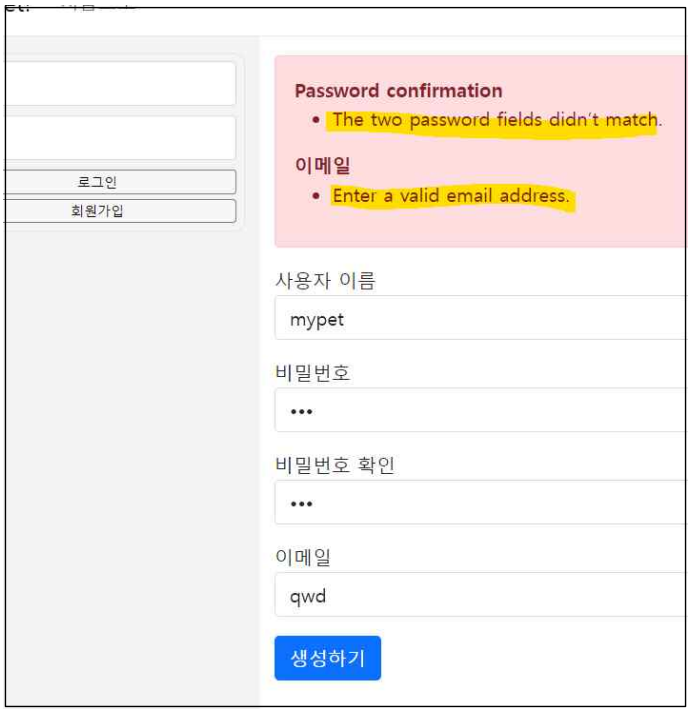
7) http://127.0.0.1:8000/myPet/1/

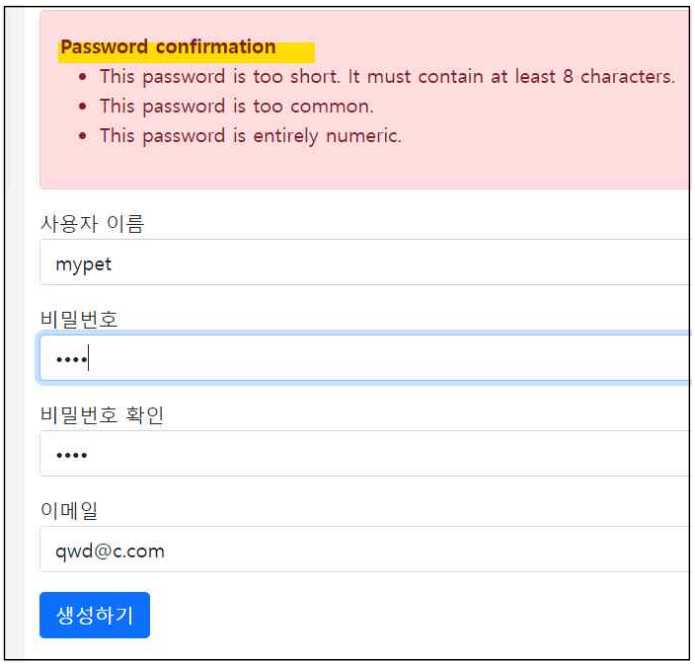




[로그아웃] 버튼을 눌러, 로그아웃을 한 뒤 게시글에 댓글을 작성하려고 하니 “댓글내용” input요소가 비활성화 되어 있는 것을 확인할 수 있었습니다. 또한, 이 게시글에 작성되어있는 댓글내용 및 “작성자, 작성날짜”가 card와 badge로 구현되어 있는 것을 확인할 수 있었습니다.

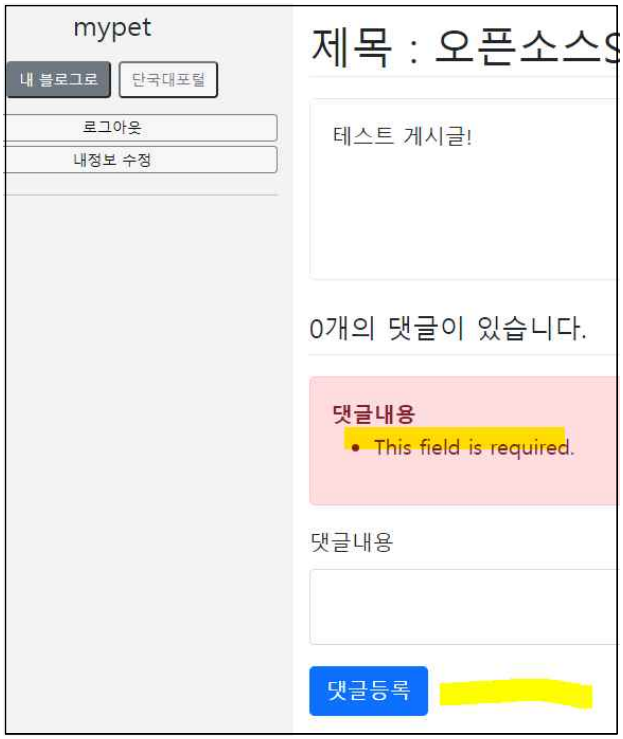
8) http://127.0.0.1:8000/common/signup/





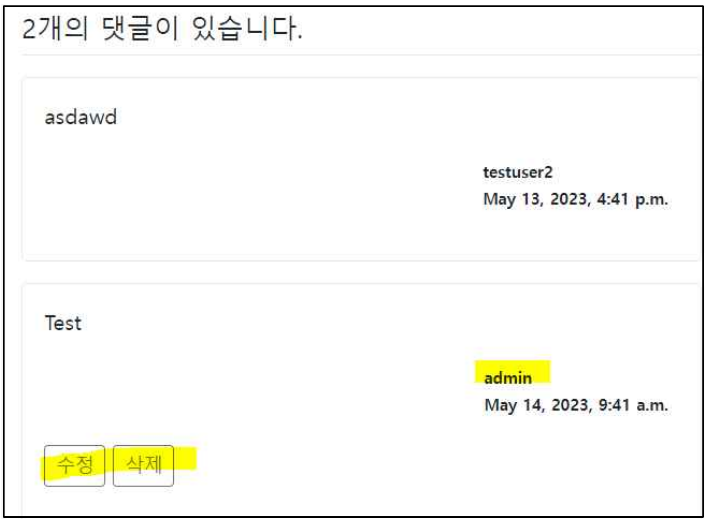
이제, 댓글을 작성하기 위해 일반 사용자계정을 추가해보겠습니다. 유효하지 않은 이메일을 작성하였을 경우 오류가 발생하고, “비밀번호”와 “비밀번호확인”이 같지 않을 경우 오류가 발생하고, 1234와 같은 너무 쉬운 비밀번호를 입력하니 보안관련 경고가 정상적으로 출력됩니다. 이후 정상적인 비밀번호로 계정을 생성해보겠습니다.

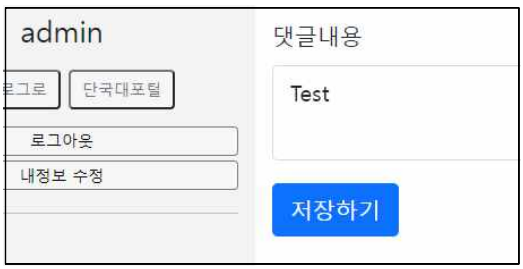
9) http://127.0.0.1:8000/myPet/20/

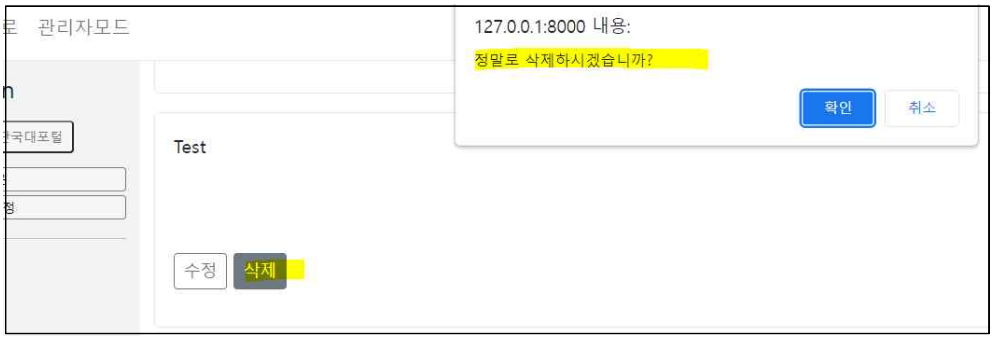


아까, admin 계정으로 작성했던 게시글입니다. 댓글창에 아무것도 입력하지 않고 댓글을 입력하려고 하니, 오류가 발생합니다. 또한, 이 계정을 일반계정이기 때문에, [댓글등록] 버튼 옆의 [공지사항] 체크박스가 사라진 것을 확인할 수 있었습니다.

10) http://127.0.0.1:8000/myPet/3/#comment_6

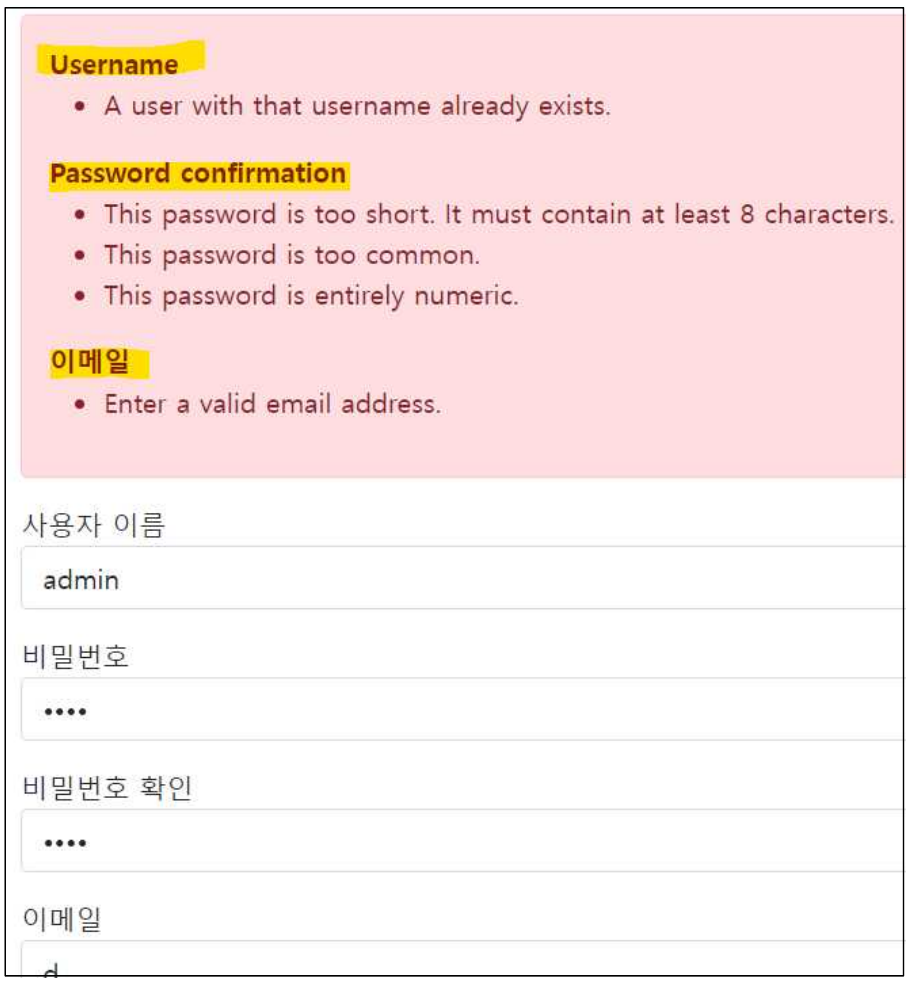






여기서 다시, admin 계정으로 로그인 한 뒤, 한 게시글에 댓글을 남겨보았습니다. 그랬더니 comment.author = user 이기 때문에 [수정] 및 [삭제] 버튼이 활성화 되었음을 알 수 있습니다. [수정]버튼을 눌렀을 때, comment_form이 다시 열리는 것을 확인할 수 있었습니다. 또한, [삭제] 버튼을 눌렀을 때, 확인메시지가 나오는 것을 확인할 수 있었습니다.

11) http://127.0.0.1:8000/common/change_info/



마지막으로, [내정보 수정] 혹은 [회원정보수정] 버튼을 클릭했을 때, 자신의 정보를 변경할 수 있는 창이 나타나는 것을 확인할 수 있었습니다. 이 역시, “비밀번호”나 “이메일” 부분에 유효하지 않은 input을 넣었을 때 오류 혹은 경고가 가 발생하는 것을 확인할 수 있었습니다.

4. Citations

이번 기회에 Django를 다시한번 공부하면서 참고했던 자료들입니다.

1. https://getbootstrap.com/docs/5.3/getting-started/introduction/
2. https://docs.python.org/3/
3. https://www.scylladb.com/glossary/ttl-value/
4. https://namu.wiki/w/JavaScript
5. https://wikidocs.net/book/4223
6. https://hackmd.io/@oW_dDxdsRoSpl0M64Tfg2g/ByfwpNJ-K