Django - 애플리케이션 확장하기

블로그 게시글이 각 페이지마다 보이게 만들어 보자.

이미 앞에서 `Post` 모델을 만들었으니 `models.py` 에 새로 추가할 내용은 없다.

Post에 템플릿 링크 만들기

`blog/templates/blog/post_list.html` 파일에 링크를 추가하는 것부터 시작한다.
`post` 제목 목록이 보이고 해당 링크를 클릭하면, `post상세 페이지` 로 이동하게 만들어 볼 것이다. `<h1><a href="">{{ post.title }}</a></h1>` 부분을 수정한다.

```
blog/templates/blog/post_list.html

{% extends 'blog/base.html' %}

{% block content %}
    {% for post in posts %}
        <div class="post">
            <div class="date">
                <p>published: {{ post.published_date }}</p>
            </div>
          ⚠️ <h1><a href="{% url 'post_detail' pk=post.pk %}">{{ post.titl
e }}</a></h1>
            <p>{{ post.text|linebreaksbr }}</p>
        </div>
    {% endfor %}
{% endblock %}
```

> `{% %}` 는 장고 템플릿 태그를 말한다.
> `blog.views.post_detail` 는 `post_detail` 뷰 경로입니다. blog는 응용프로그램(디렉터리 blog)의 이름인 것을 꼭 기억한다. `views` 는 `views.py` 파일명이다. 마지막 부분 `post_detail` 는 `view` 이름이다.

`pk` 는 데이터베이스의 각 레코드를 식별하는 `기본키(Prmiary Key)` 의 줄임말 이다.
Post 모델에서 기본키를 지정하지 않았기 때문에 장고는 `pk` 라는 필드를 추가해 새로운 블로그 게시물이 추가될 때마다 그 값이 1,2,3 등으로 증가하게 된다. Post 객체의 다른 필드 (제목, 작성자 등)에 액세스하는 것과 같은 방식으로 `post.pk` 를 작성하여 기본 키에 액세스한다. 같은 방법으로 Post객체내 다른 필드(title, author)에도 접근할 수 있다.

Post 상세 페이지를 url에 추가

`post_detail` 뷰가 보이게 `urls.py` 에 URL를 만든다.
첫 게시물의 상세 페이지 `URL` 이 http://127.0.0.1:8000/post/1/가 되게 만들것이다.

`blog/urls.py` 파일에 URL을 만들어, 장고가 `post_detail` 뷰로 보내, 게시글이 보일 수 있게 할것이다.
`path('post/<int:pk>/', views.post_detail, name='post_detail')` 코드를 `blog/urls.py` 파일에 추가하면 아래와 같이 보일 것이다.

```
blog/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('', views.post_list, name='post_list'),
  ⚠ path('post/<int:pk>/', views.post_detail, name='post_detail'),
]
```

여기서 `post/<int:pk/>/` 는 URL 패턴을 나타낸다.

- `post/` : URL이 post 문자를 포함해야 한다는 것을 말한다.
- `<int:pk>` : 장고는 정수 값을 기대하고 이를 pk라는 변수로 뷰로 전송하는 것을 말한다.
- /은 다음에 / 가 한 번 더 와야 한다는 의미이다.

브라우저에 http://127.0.0.1:8000/post/5/라고 입력하면, 장고는 `post_detail` 뷰를 찾아 매개변수 pk가 5인 값을 찾아 뷰로 전달하게 된다.

Post_detail 뷰 생성

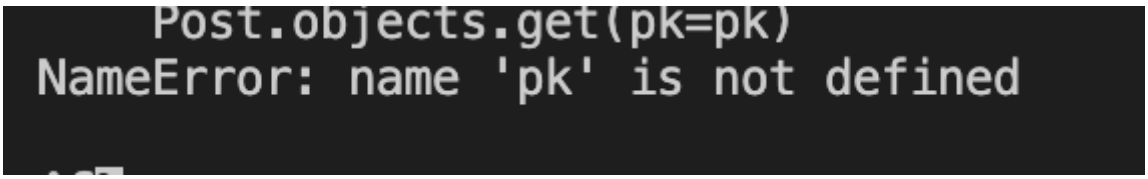
뷰가 `pk` 를 식별해야한다. 그래서 함수를 `def post_detail(request, pk):` 라고 정의한다. `urls(pk)` 과 동일하게 이름을 사용해야 한다.

블로그 게시글 한 개만 보려면, 아래와 같이 쿼리셋(queryset)을 작성해야한다.

```
blog/views.py

Post.objects.get(pk=pk)
```

`primary key(pk)` 의 `Post` 를 찾지 못하면 오류가 나올 것이다.



404에러 추가하기

장고에는 `get_object_or_404`라는 특별한 기능을 제공한다. `pk`에 해당하는 Post가 없을 경우, `페이지 찾을 수 없음 404 : Page Not Found 404`를 보여줄 것이다.

`views.py`파일에 새로운 뷰를 추가한다.
`blog/urls.py`파일에서 `views.post_detail`라는 뷰를 `post_detail`이라 이름을 붙이도록 URL 법칙을 만들었다. 이는 장고가 `post_detail`이라는 이름을 해석할 때, `blog/views.py`파일 내부의 `post_detail`이라는 뷰 함수로 이해하도록 해준다.

`blog/views.py`파일을 열고, 다음과 같이 코드를 import한다. :

```
blog/views.py

from django.shortcuts import render, get_object_or_404
```

마지막 부분에 `Post.objects.get(pk=pk)`를 수정한다.

```
blog/views.py

def post_detail(request, pk):
    post = get_object_or_404(Post, pk=pk)
    return render(request, 'blog/post_detail.html', {'post': post})
```

published: 2020년 5월 1일 8:33 오후

notice

This is a test notice.
Hello World
Hello Django~

잘 작동한다.
하지만 글 링크를 눌러 들어가면 에러가 발생된다.

post_detail 템플릿 만들기

`blog/templates/blog` 디렉터리 안에 `post_detail.html`라는 새 파일을 생성하고 아래와 같이 코드를 작성한다.

```
blog/templates/blog/post_detail.html

{% extends 'blog/base.html' %}

{% block content %}
    <div class="post">
        {% if post.published_date %}
            <div class="date">
                {{ post.published_date }}
            </div>
        {% endif %}
        <h1>{{ post.title }}</h1>
        <p>{{ post.text|linebreaksbr }}</p>
    </div>
{% endblock %}
```

> `base.html`을 확장한 것이다. `content` 블록에서 블로그 글의 게시일, 제목과 내용을 보이게 만들었다.

가장 중요한 부분은 `{% if ... %} ... {% endif %}`라는 템플릿 태그인데, 내용이 있는지 확인할 때 사용한다. `post`의 게시일`(published_date)`이 있는지, 없는지를 확인하는 것이다.

2020년 5월 1일 8:33 오후

notice

This is a test notice.
Hello World
Hello Django~