

Django - 장고 폼

한 가지만 더 하면 웹사이트가 완성된다. 바로 블로그 글을 추가하거나 수정하는 기능을 추가하는 것이다.

폼

폼(양식, `forms`)으로 강력한 인터페이스를 만들 수 있다.

장고 폼이 정말 좋은 것은 아무런 준비 없이도 양식을 만들 수 있고, `ModelForm`을 생성해 자동으로 모델에 결과물을 저장할 수 있다는 것이다.

폼을 생성해보자.

```
blog
└── forms.py
```

```
blog/forms.py

from django import forms

from .models import Post

class PostForm(forms.ModelForm):

    class Meta:
        model = Post
        fields = ('title', 'text',)
```

위 코드를 보면 첫 번째로 `forms` `model`을 `import` 해야 하고 `(from django import forms)`, 그다음으로 `Post` `model`도 `import` 해야 한다. `(from .models import Post)`.

`PostForm`은 우리가 만들 폼의 이름이다. 그리고 장고에 이 폼이 `ModelForm`이라는 것을 알려줘야 한다.

다음으로 `class Meta`가 나오는데, 이 구문은 이 폼을 만들기 위해서 어떤 `model`이 쓰여야 하는지 장고에 알려주는 구문이다. `(model = Post)`.

이번 폼에서는 `title`과 `text`만 보여지게 할 것이다. `- author`는 현재 로그인 하고 있는 사람이 될 것이고 `created_date`는 글이 등록되는 시간이 될 것이다.

이제 `뷰`에서 이 폼을 사용해 템플릿에서 보여주기만 하면 된다.

폼과 페이지 링크

`blog/templates/blog/base.html` 파일을 열고, `page-header` 라는 div class에 링크를 하나 추가할 것이다.

```
blog/templates/blog/base.html
```

```
<a href="{% url 'post_new' %}" class="top-menu"><span class="glyphicon gly  
phicon-plus"></span></a>
```

위 구문을 추가하고 나면, 이제 html 파일이 아래처럼 보일 것이다.

```
{% load static %}  
<html>  
  <head>  
    <title>Django Girls blog</title>  
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/  
3.2.0/css/bootstrap.min.css">  
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/  
3.2.0/css/bootstrap-theme.min.css">  
    <link href='//fonts.googleapis.com/css?family=Lobster&subset=lati  
n,latin-ext' rel='stylesheet' type='text/css'>  
    <link rel="stylesheet" href="{% static 'css/blog.css' %}">  
  </head>  
  <body>  
    <div class="page-header">  
      <a href="{% url 'post_new' %}" class="top-menu"><span class="g  
lyphicon glyphicon-plus"></span></a>  
      <h1><a href="/">Django Girls Blog</a></h1>  
    </div>  
    <div class="content container">  
      <div class="row">  
        <div class="col-md-8">  
          {% block content %}  
          {% endblock %}  
        </div>  
      </div>  
    </div>  
  </body>  
</html>
```

페이지를 새로고침하면 아래와 같은 에러가 발생할 것이다.

NoReverseMatch at /

Reverse for 'post_new' not found. 'pos

```
Request Method: GET
Request URL: http://127.0.0.1:8000/
Django Version: 3.0.5
Exception Type: NoReverseMatch
Exception Value: Reverse for 'post_new'
Exception Location: /Users/yong-kwangsoon/
Python Executable: /Users/yong-kwangsoon/
Python Version: 3.7.7
Python Path: [ '/Users/yong-kwangsoon',
               '/usr/local/Cellar/py'
```

url

이제 `blog/urls.py` 를 열고 아래 구문을 추가한다.

```
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'),
    ⚠ path('post/new/', views.post_new, name='post_new'),
]
```

브라우저에 사이트를 다시 불러오면 `AttributeError` 가 보이게 된다. 왜냐하면, 아직 `post_new` 뷰를 구현하지 않았기 때문이다.

post_new 뷰 만들기

`blog/views.py` 파일을 열어서 `from` 줄에 아래와 같은 코드를 추가한다.

```
blog/views.py

from .forms import PostForm
```

그리고 `post_new` 함수를 추가한다.

```
blog/views.py

def post_new(request):
    form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})
```

post_edit 템플릿 만들기

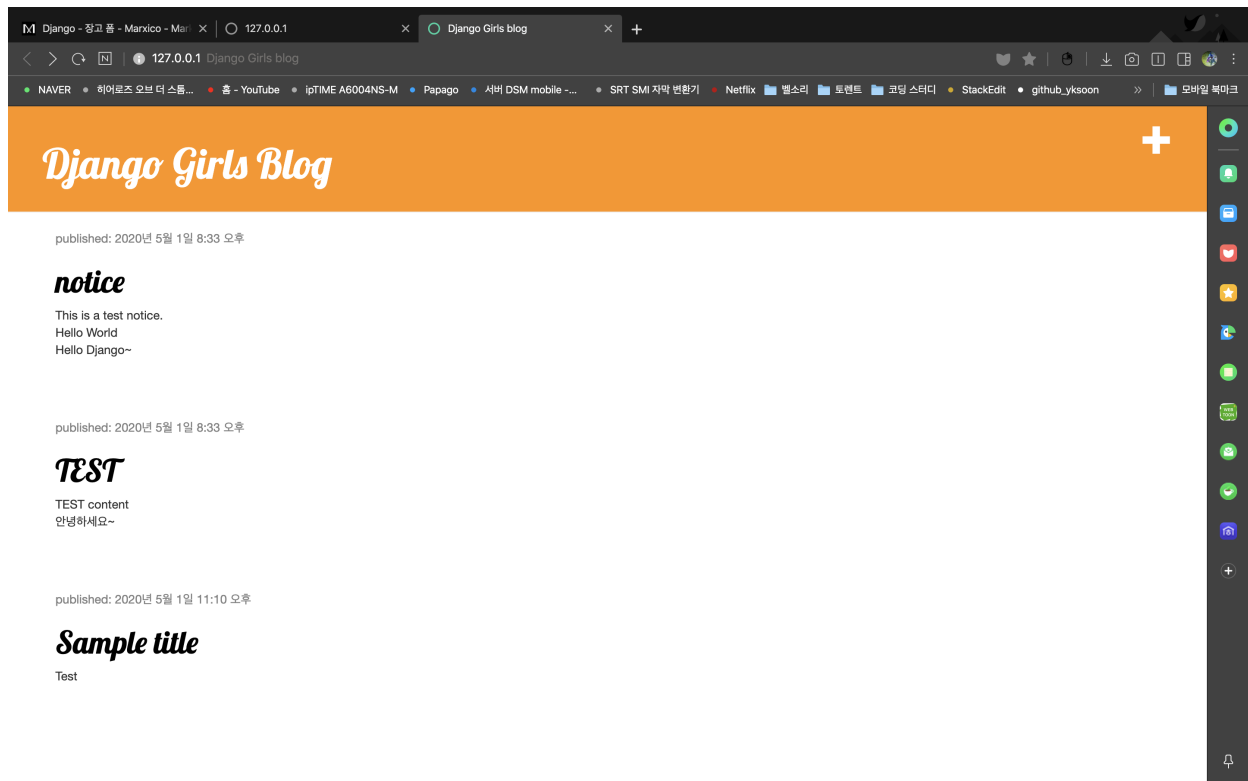
`views` 에서 만든 `post_new` 함수에서 `blog/post_edit.html` 템플릿으로 넘기게 된다.

`blog/templates/blog` 디렉터리 안에 `post_edit.html` 파일을 생성해 폼이 작동할 수 있게 만들 것이다.

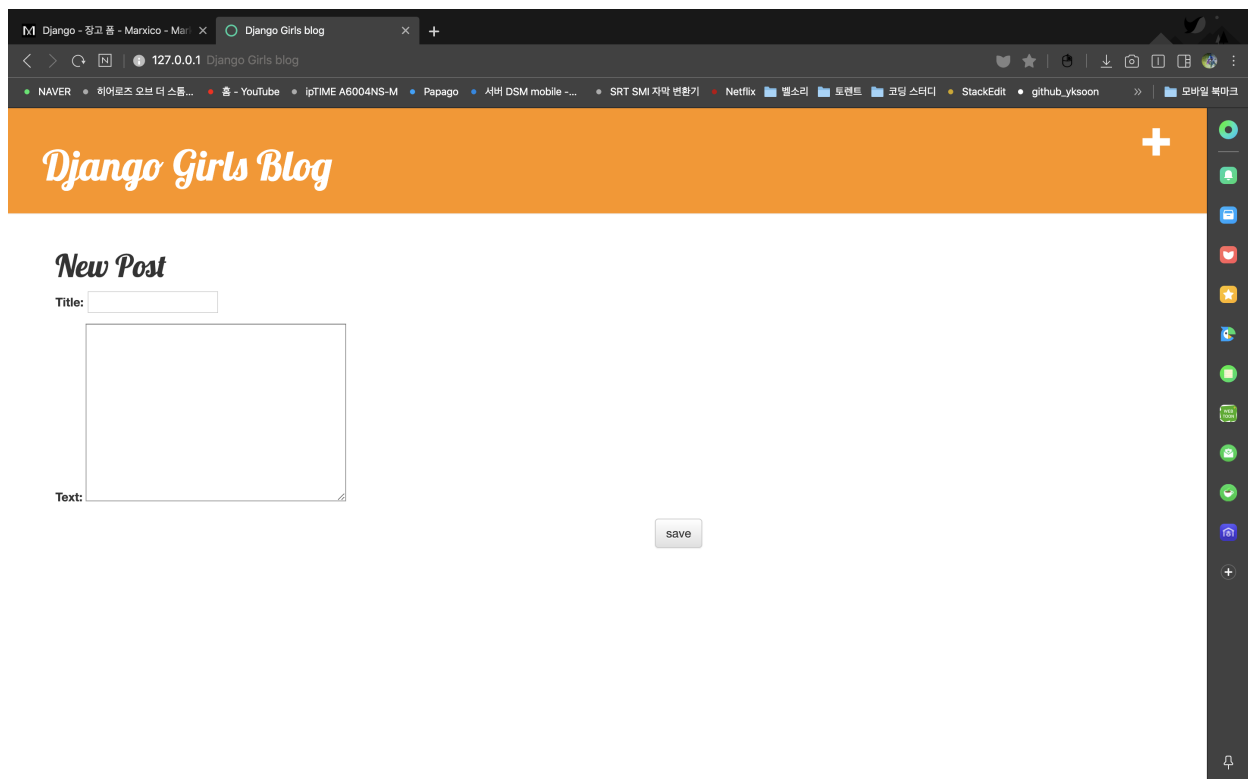
```
blog/templates/blog/post_edit.html

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

{% block content %}
    <h1>New post</h1>
    <form method="POST" class="post-form">{% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Save</button>
    </form>
{% endblock %}
```



상단에 + 버튼은 누르면 아래와 같이 나온다



하지만 제목과 글을 작성하고 save 버튼을 눌러도 작성한 글이 보이지 않는다.
정상이다. view에서 추가 작업을 해주어야 한다.

폼 저장하기

`blog/views.py` 를 다시 열고 확인해보면 `post_new` 뷰는 아래와 같을 것이다.

```
blog/views.py
```

```
def post_new(request):  
    form = PostForm()  
    return render(request, 'blog/post_edit.html', {'form': form})
```

폼을 제출할 때, 같은 뷰를 불러온다. 이때 `request` 에는 우리가 입력했던 데이터들을 가지고 있는데, `request.POST` 가 이 데이터를 가지고 있다. (POST는 글 데이터를 “등록하는(posting)”하는 것을 의미한다. 블로그 “글”을 의미하는 “post”와 관련이 없다) HTML에서 `<form>` 정의에 `method="POST"` 라는 속성은 POST 로 넘겨진 폼 필드의 값들은 이제 `request.POST` 에 저장된다. POST 로 된 값을 다른 거로 바꾸면 안 된다.

이제 view 에서 두 상황으로 나누어 처리해볼것이다.

- 첫 번째: 처음 페이지에 접속했을 때. 새 글을 쓸 수 있게 폼이 비어있어야한다.
- 두 번째: 폼에 입력된 데이터를 view 페이지로 가지고 올 때. 여기서 조건문을 추가시켜야 한다. (if를 사용)

`def post_new(request):` 안의 `form = PostForm()` 을 수정하자

```
if request.method == "POST":  
    pass  
else:  
    form = PostForm()
```

이제 `pass`부분을 수정하자. 만약 `method` 가 POST 라면, 폼에서 받은 데이터를 `PostForm` 으로 넘겨주어야 한다.

```
form = PostForm(request.POST)
```

다음에는 폼에 들어있는 값들이 올바른지를 확인해야한다.(모든 필드에는 값이 있어야하고 잘못된 값이 있다면 저장하면 되지 않아야함) 이를 위해 `form.is_valid()`을 사용할것이다.

`form = PostForm(request.POST)` 바로 아래줄에 작성하자


```
if form.is_valid():  
    post = form.save(commit=False)  
    post.author = request.user  
    post.published_date = timezone.now()  
    post.save()
```

일반적으로 이 작업을 두 단계로 나눌 수 있다. `form.save()` 로 폼을 저장하는 작업과 작성자를 추가하는 작업이다. (PostForm에는 작성자(author) 필드가 없지만, 필드 값이 필요함) `commit=False` 란 넘겨진 데이터를 바로 Post 모델에 저장하지는 말라는 뜻이다. 왜냐하면, 작성자를 추가한 다음 저장해야 하기 때문이다. 대부분의 경우에는 `commit=False` 를 쓰지 않고 바로 `form.save()` 를 사용해서 저장한다. 다만 여기서는 작성자 정보를 추가하고 저장해야 하므로 `commit=False` 를 사용하는 것이다. `post.save()` 는 변경사항(작성자 정

보를 포함)을 유지할 것이고 새 블로그 글이 만들어질 것이다.

새 블로그 글을 작성한 다음에 `post_detail` 페이지로 이동할 수 있는 기능을 추가할 것이다.

`import` 부분에 `redirect` 를 추가로 import 하자

```
from django.shortcuts import render, get_object_or_404,  redirect
```

그리고 새로 작성한 글을 볼 수 있도록 `post_detail` 페이지로 가라고 수정한다.

`post.save()` 바로 다음줄에 작성한다.

```
return redirect('post_detail', pk=post.pk)
```

`post_detail` 은 우리가 이동해야 할 뷰의 이름이다. `post_detail` 뷰 는 `pk` 변수가 필요하다. `pk=post.pk` 를 사용해서 뷰에게 값을 넘겨줄 것이다. 여기서 `post` 는 새로 생성한 블로그 글이다.

아래는 `post_new` 전체 코드이다.

```
blog/views.py

def post_new(request):
    if request.method == "POST":
        form = PostForm(request.POST)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.published_date = timezone.now()
            post.save()
            return redirect('post_detail', pk=post.pk)
        else:
            form = PostForm()
            return render(request, 'blog/post_edit.html', {'form': form})
```

<http://127.0.0.1:8000/post/new/> 페이지로 접속해서 `title` 과 `text` 를 입력하고, 저장하면 새로운 블로그 글이 추가되고 `post_detail` 페이지가 나타난다.

폼 검증하기

블로그 글은 `title` 과 `text` 필드가 반드시 있어야 한다. 우리가 만든 `Post` 모델에서는 이 필드 값들이 필요 없다고 했지만(`published_date`는 제외하고) 장고는 모두 기본값으로 설정되어 있다고 생각한다.

아무것도 작성하지 않고 저장하게 되면 아래와 같이 나온다.

Django Girls Blog

New Post

Title:



이 입력란을 작성하세요.

폼 수정하기

`blog/templates/blog/post_detail.html` 파일을 열어 아래 내용을 추가한다.

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

```
<a class="btn btn-default" href="{% url 'post_edit' pk=post.pk %}"><span class="glyphicon glyphicon-pencil"></span></a>
```

아래 처럼 나올것이다.

```
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 %}
```

```
⚠ <a class="btn btn-default" href="{% url 'post_edit' pk=post.pk %}"><span class="glyphicon glyphicon-pencil"></span></a>
```

```
<h1>{{ post.title }}</h1>
```



```
<p>{{ post.text|linebreaksbr }}</p>
</div>
{% endblock %}
```

`blog/urls.py` 에 다음 코드를 추가한다.

```
blog/urls.py

path('post/<int:pk>/edit/', views.post_edit, name='post_edit'),
```

`blog/templates/blog/post_edit.html` 템플릿을 재사용할 것이다. 마지막으로 할 일은 `view` 를 만드는 것이다.

`blog/views.py` 파일을 열어서 파일 맨 밑에 `post_edit` 함수를 추가한다.

```
blog/views.py

def post_edit(request, pk):
    post = get_object_or_404(Post, pk=pk)
    if request.method == "POST":
        form = PostForm(request.POST, instance=post)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.published_date = timezone.now()
            post.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = PostForm(instance=post)
    return render(request, 'blog/post_edit.html', {'form': form})
```

`post_new` 와 거의 비슷해 보이지만 완전히 같지는 않다.

- 첫 번째: `url` 로부터 추가로 `pk` 매개변수를 받아서 처리한다.
- 두 번째: `get_object_or_404(Post, pk=pk)` 를 호출하여 수정하고자 하는 글의 `Post` 모델 인스턴스(instance)로 가져온다. (pk로 원하는 글을 찾는다) 이렇게 가져온 데이터를 폼을 만들 때와(글을 수정할 때 폼에 이전에 입력했던 데이터가 있어야 한다) 폼을 저장할 때 사용하게 된다.

Django Girls Blog

New Post

Title:

This is a test notice.
Hello World
Hello Django~

Text:

보안

지금은 웹사이트를 방문하는 누구든지 글을 쓸 수 있지만, 그렇게 하고 싶지 않을 수 있다. 나에게만 보이고 다른 사람들에게는 보이지 않는 버튼을 만들어 볼 것이다.

`blog/templates/blog/base.html` 파일에서, `page-header div` 를 찾아 아래와 같이 작성된 `앵커 태그(Anchor Tag)` 를 찾는다.

```
blog/templates/blog/base.html
```

```
<a href="{% url 'post_new' %}" class="top-menu"><span class="glyphicon glyphicon-plus"></span></a>
```

여기에 `{% if %}` 태그를 추가해 관리자로 로그인한 유저들만 링크가 보일 수 있게 만들 것이다. `<a>` 태그를 아래와 같이 변경한다.

```
blog/templates/blog/base.html

{% if user.is_authenticated %}
    <a href="{% url 'post_new' %}" class="top-menu"><span class="glyphicon glyphicon-plus"></span></a>
{% endif %}
```

이 `{% if %}` 는 브라우저에 페이지를 요청 하는 사용자가 로그인 하는 경우 링크가 발생된다. 이는 새 게시글을 완전히 보호해주는 것은 아니지만, 바람직한 방법이다.

세부 페이지에 있는 수정 아이콘도 동일하게 다른 사람들이 게시글을 수정하지 못하게 할 것이다.

`blog/templates/blog/post_detail.html` 파일을 열어 아래와 같이 작성된 라인을 찾는다.

```
<a class="btn btn-default" href="{% url 'post_edit' pk=post.pk %}"><span class="glyphicon glyphicon-pencil"></span></a>
```

아래와 같이 변경

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

{% if user.is_authenticated %}
    <a class="btn btn-default" href="{% url 'post_edit' pk=post.pk %}"><span class="glyphicon glyphicon-pencil"></span></a>
{% endif %}
```