Day6 Form 사용하기

구분	Django
과목	

test33.zip

[시작 스켈레톤 코드는 저번시간에 완성했던 부분 부터 이어시 진행하겠습니다.]

오늘 학습할 내용은 Django에서 제공해주는 form 을 사용하는 것을 학습할 것
form 을 사용하기 전에 저번 시간에 했던 코드를 조금 수정 해 줄 것이다.

Create 그리고 Update 에서 views.py에서 해당 함수를 작성할 때

1. 게시글을 작성하는 함수
2. 작성한 게시글을 DB에 저장하는 함수

이렇게 각각 기능 별로 따로 함수를 정의해 주었다. 하지만 오늘은 두 함수를 하나로 합칠 것이다.

- request가 POST 일때 new_todo 함수 코드를 적용시킬 것이고
- request가 GET 일때 create_todo 함수 코드가 적용되도록 수정하면 될 것이다.

수정 내용은 아래와 같다.

Create

- new_todo 와 create_todo 함수를 합침
- 각각의 역할은 `request.method` 값을 기준으로 나뉨

```
def create_todo(request):
    if request.method == 'POST':
        work = request.POST.get('work')
        content = request.POST.get('content')
        is_completed = False

        todo = Todo(work=work, content=content, is_completed=is_completed)
        todo.save()
        return redirect('todos:detail', todo.pk)
```

```
    else:
        return render(request, 'todos/create_todo.html')
```

필요없는 경로 삭제

```
# path('new_todo/', views.new_todo, name='new_todo'),
```

create_todo.html 경로 수정

```
<form action="{% url 'todos:create_todo' %}" method="POST">
```

Update

- edit_todo 와 update_todo view 함수 합침
- 각각의 역할은 `request.method` 값을 기준으로 나뉨

```
def update_todo(request, todo_pk):
    todo = Todo.objects.get(pk=todo_pk)

    if request.method == 'POST':
        work = request.POST.get('work')
        content = request.POST.get('content')

        todo.work = work
        todo.content = content
        todo.save()
        return redirect('todos:detail', todo.pk)
    else:
        context = {
            'todo': todo
        }
        return render(request, 'todos/update_todo.html', context)
```

필요없는 경로 삭제

```
# path('<int:todo_pk>/edit_todo/', views.edit_todo, name='edit_todo'),
```

update_todo.html 경로 수정

```
<form action="{% url 'todos:update_todo' todo.pk %}" method="POST">
```

이왕 리팩토링을 하는 김에 delete 함수도 손을 보자.

- Httpmethod를 사용해서 POST 요청에 대해서만 삭제가 가능하도록 수정 해 보자.

Delete

- 각각의 역할은 `request.method` 값을 기준으로 나뉜다.

```
def delete_todo(request, todo_pk):
    todo = Todo.objects.get(pk=todo_pk)
    if request.method == 'POST':
        todo.delete()
        return redirect('todos:index')
    else:
        return redirect('todos:detail', todo.pk)
```

여기까지 우리가 작성했던 코드를 수정해 보았다.

`request.method` 값을 기준으로 나눴던 create관련 2개의 함수를 하나로 합쳤으며
update관련 함수 2개도 하나로 합치는 과정이였다.

그동안 우리가 HTML Form, input 태그를 사용해서 사용자(유저)로 부터 입력을 받았다.
그러나 사용자로부터 받은 데이터가 유효한 데이터 인지?
비정상적인 값이 입력이 되었는지? 를 판별하는 유효성 검사를 하는 코드는 작성하지 않았다.

유효성 검증을 하기 위해서는 많은 부가적인 것들을 고려해서 소스코드를 구현해야 하는데
이는 개발 생산성을 늦출 뿐더러 쉽지 않은 작업이다.

그래서 Django에서 제공해 주는 form을 사용해서 사용자 입력을 받는다면
과중한 유효성검사를 하는 소스코드를 직접 작성 할 필요가 없어진다.

따라서 지금까지는 HTML의 Form 그리고 input 태그를 이용해서 사용자로부터 입력받는 형태였다면,
이제부터는 Django 프레임 워크에서 제공해 주는 Django form 을 이용하는 코드로
한번 더 수정해 보자.

Django Form을 사용하면 코드작성도 더 간단해 질 뿐만 아니라
기본적인 유효성 검사기능까지 이미 탑재가 되어 있기 때문이다.

Django에서 제공하는 form은 크게 두 종류가 있다.

1. Django Form
2. Django ModelForm

Form과 ModelForm

- ModelForm과 Form 은 각각 역할이 다르다.
- Form
  - 사용자의 입력을 필요로 하며 직접 입력 데이터가 DB 저장에 사용되지 않거나
    혹은 일부 데이터만 사용될 때 사용
  - 예를들면 로그인 기능 구현 같은 경우 사용자의 데이터를 입력받아 인증 과정에서만 사용 후
    별도로 DB에 저장하지 않는 경우가 되겠다.

- ModelForm
  - 사용자의 입력을 필요로 하며 입력을 받은 것을 그대로 DB 필드에 맞춰 저장할 때 사용
  - 데이터의 유효성 검사가 끝나면 데이터를 각각 어떤 레코드에 맵핑 해야 할지 이미 알고 있기 때문에 곧바로 save() 호출이 가능하다

둘이 사용처 (역할이) 다르다는 것만 인지를 하고 직접 사용해 보자.

todos앱 폴더에 `forms.py`를 생성 하자.
우리는 새 게시글(할일)을 입력받으면 그 내용들이 DB에 저장까지 하는 Form을 만들 것이기 때문에
ModelForm을 정의 해 줄 것이다.

```
from django import forms
from .models import Todo

class TodoForm(forms.ModelForm):

    class Meta:
        model = Todo
        fields = '__all__'
```

- django 에서 제공해 주는 forms 모듈을 import 한다. 그리고 Todo 클래스도 불러오자.
- TodoForm 이라는 클래스를 새로 정의해 주고 Forms 모듈의 ModelForm을 상속 받는다.
- Meta Class 라는 것을 사용했다.
  - Meta Class는 ModelForm의 정보를 작성하는 곳이다.
  - ModelForm을 사용할 경우 참조할 모델이 있어야 하는데, Meta class의 model 속성이 이를 구성한다
  - 그리고 fields 속성에 `'__all__'`를 사용해서 모델의 모든 필드를 작성하는 Form에 포함할 것이다.
    - 즉, 입력받을 때 models.py 에 정의해 놓은 Todo클래스의 모든 필드를 입력받겠다는 뜻으로 'work' , 'content' 그리고 'is_completed' 모두 사용자로 부터 입력을 받겠다는 이야기 이다.

아래 코드는 적지 말고 보기만 하자.

만약에 아래와 같이 forms.py를 작성했다고 하자. (아래 코드는 작성하지 말고 눈으로만 보자)

```
# forms.py

from django import forms
from .models import Todo

class TodoForm(forms.ModelForm):

    class Meta:
        model = Todo
        fields = '__all__'
        exclude=('work',)
```

Meta class에 exclude 속성을 사용하면

사용자가 작성 할 Form에 포함시키지 않을 필드를 지정하는 것이 되겠다.

한가지 더 중요한 사실을 이야기를 하자면

Forms.py에 있는 Form Class는 models.py에서 model class와 달리 TextField 가 존재하지 않는다.

따라서 models.py에서 우리가 정의했던 것처럼 model class안에 TextField를 사용하려면

forms.py에서 form class 안에는 widgets 을 사용해야 한다.

사용 예제는 다음과 같다. 따라 작성해 보자.

```
from django import forms
from .models import Todo

class TodoForm(forms.ModelForm):

    content=forms.CharField(

        widget=forms.Textarea(
            attrs={
                'class':'work',
                'placeholder':'할일을 등록해 주세요',
                'rows': 4, 'cols': 15
            }
        )
    )
```

```
    class Meta:
        model = Todo
        fields = '__all__'
```

Widget을 통해서 사용자가 작성할 내용에 들어 갈 속성을 작성해 줄 수 있다.
(나중에 개발자도구에서 보면 input 태그에 해당함)

- content = forms.CharFiend 를 사용해서 텍스트 필드를 정의를 한다.

- widget 속성은 폼 필드가 사용할 위젯을 지정할때 사용하는 속성이다.
    - widget = forms.Textarea() 는 여러줄의 텍스트 입력을 받을때 사용한다. 예를들어
    - 이메일을 입력받을때
        - forms.EmailField(widget=forms.EmailInput(attrs={'placeholder': 'Enter your email'}))
    - 비밀번호를 입력받을때
        - forms.CharField(widget=forms.PasswordInput())
          처럼 사용할 수 있다. Django 공식문서 보다는 GPT나 구글링 해서 찾아 사용하는 편이
          보기가 편할 것이다.

- 그리고 atters 는 매개변수로 HTML 속성을 지정할 때 사용된다.
    - 'class':'work' 라고 지정을 하면 이는, CSS 클래스가 work 로 설정이 된다.
    - 'placeholder':'할일을 등록해 주세요',는 말 그대로 플레이스 홀더 적용이다.

Create

자 이제 form을 완성 했으니 view 함수에 우리가 만들어 놓은 form을 적용시켜 보자.
create_todo 함수로 가보자.

views.py 에서 create_todo 수정

기존에 있는 코드에서 form을 적용하면 다음과 같이 적용 시킬 수 있다.

```
from . forms import TodoForm

def create_todo(request):
    if request.method == 'POST':
```

```
        form = TodoForm(request.POST)
        if form.is_valid():
            todo = form.save()
            return redirect('todos:detail', todo.pk)
        context = {
            'form': form
            }
        return render(request, 'todos/create_todo.html', context)

    else:
        form=TodoForm()
        context = {
            'form': form
        }
        return render(request, 'todos/create_todo.html',context)
```

- 가장 먼저, forms.py 에서 우리가 작성 해 놓은 TodoForm 클래스를 import 한다.
- 만약에 POST 요청일 경우
  - form 변수에 TodoForm 클래스의 내용을 담는다
  - form 을 django에서 제공하는 is_vaild() 함수를 통해 데이터가 유효한 데이터 라는 것이 검증이 되면, form을 저장과 동시에 todo라는 새로운객체에 저장을 한다.
  - todo 객체의 pk와 함께 해당 디테일 페이지로 redirect 한다.
- GET 요청일 경우
  - 작성 form을 template으로 넘겨 유저가 form에 입력 할 수 있도록 한다.

그러나 위 코드를 보면

```
  context = {
    'form': form
  }
  return render(request, 'todos/create_todo.html', context)
```

부분이 의미 없이 중복이 되고 있다.

중복되는 코드를 정리를 하면 아래와 같이 수정이 가능하다.

```
from . forms import TodoForm

def create_todo(request):
    if request.method == 'POST':       # POST 요청일 경우
        form = TodoForm(request.POST)   #유저가 작성한 내용을 form에 담은 후
        if form.is_valid():            #작성된 내용이 유효한 정보라면
            todo = form.save()         #Database에 저장
```

```
            return redirect('todos:detail', todo.pk)
    else:                   #GET 요청일 경우
        form = TodoForm()       #유저가 작성할 form을 foms.py에 정의해 놓은 form을 가져온다

    context = {
        'form': form          # post요청 또는 get요청으로 완성된 form을
    }
    return render(request, 'todos/create_todo.html', context) # create.html문서로 보낸다
```

[참고] 유효성 검사 method를 살펴보자.

`is_valid()` method

- 유효성 검사를 실행하고 데이터가 유효한지 여부를 True 또는 False로 반환한다.
- 데이터 유효성 검사를 보장하기 위한 많은 테스트에 대해 Django는 `is_valid()`를 제공하여 개발자의 편의를 도움을 준다. 예를들면, 필수 필드가 비어있지는 않는지? 정수를 입력받아야 하는데 문자열이 입력되지 않는지? 입력받는 값의 포맷(email URL 등)이 올바른 입력값인지 알아서 검사를 해준다.

create_todo template 수정

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

{% block content %}
  <h1>이곳에 할 일을 생성합니다.</h1>

  <form action="{% url 'todos:create_todo' %}" method="POST">
    {% csrf_token %}

    {{ form.as_p }}

    <input type="submit" value="제출하기" />
  </form>
{% endblock %}
```

여기서 `{{ form.as_p }}`는 폼의 모든 필드를 `<p>` 태그로 감싸서 렌더링하겠다는 의미이다.

`as_p` 메서드는 폼 필드의 기본 HTML을 생성하는 데 사용된다. 만약에 리스트의 형식으로 하겠다면 `as_ul()`로 작성 할 수도 있겠다.

Update

update view 수정

아래 주석이 100% 모두 이해가 되야 할 것이다.

```
def update_todo(request, todo_pk):
    todo = Todo.objects.get(pk=todo_pk)                # 수정 전 기록이 담긴 객체 (todo)
    if request.method == 'POST':
        form = TodoForm(request.POST, instance=todo)    # 새로운 기록이 담긴 객체 (form)
        if form.is_valid():
            todo = form.save()                         # todo객체에 새로운기록을 넣고 DB에 저장
            return redirect('todos:detail', todo.pk)
    else:
        form = TodoForm(instance=todo)                 # get요청일 경우 수정전 기록이 담긴 객체 (todo)
    context = {
        'todo': todo,
        'form': form                                   # (post라면) form에는 유효하지 않은정보 또는
    }                                                  # (get이라면) 빈 modleform이 전달 될 것임
    return render(request, 'todos/update_todo.html', context)
```

수정 전 기록이 담긴 객체 (todo)

새로운 기록이 담긴 객체 (form)

todo객체에 새로운기록을 넣고 DB에 저장

get요청일 경우 수정전 기록이 담긴 객체 (todo)

(post라면) form에는 유효하지 않은정보 또는

(get이라면) 빈 modleform이 전달 될 것임

- `request.POST` : 사용자가 form을 통해 전송한 데이터(새로운 데이터)
- `instance` : 수정이 되는 대상

update template 수정

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

{% block content %}
  <h1>이 곳에서 할 일을 수정합니다.</h1>

  <form action="{% url 'todos:update_todo' todo.pk %}" method="POST">
    {% csrf_token %}
    {{form.as_p}}
    <input type="submit" value="수정완료" />
  </form>
{% endblock %}
```

Django에서 제공하는 form은 2종류가 있고 앞에서 언급했다.

1. Django Form
2. Django ModelForm

이번에는 Django Form을 작성해 볼 할 것이다.
로그인 인증 기능 같은 것은 사용자가 입력한 데이터를 DB 따로 저장을 하지는 않을 것이다.
유효성 검사를 통해서 인증 절차만 밟기 때문이다.
그렇기 때문에 로그인 인증기능을 위해서 지금까지 연습했던 ModelForm을 사용하는 것이 아니라
그냥 Form을 사용할 것이다.

실제로 화면으로 확인하는 것은 차후 로그인 페이지를 구현 할 때 자세하게 할 것이다.
아직 진도를 나가지 않은 부분임으로 일단을 따라서 작성만 해보자.

accounts 앱에 들어가서 forms.py 파일을 생성 후 다음과 같이 작성해 보자.

```
from django import forms

class LoginForm(forms.Form):
    username = forms.CharField(max_length=12)
    password = forms.CharField(widget=forms.PasswordInput())
```

그다음 URL경로는 따로 변경사항이 없으므로
바로 views.py 로 넘어가자

```
from django.shortcuts import render
from .forms import LoginForm

# Create your views here.
def login(request):
    form = LoginForm()
    context = {
        'form': form
    }
    return render(request, 'accounts/login.html', context)
```

이제 login.html 쪽으로 넘어가자

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

{% block content %}
  <h1>로그인 페이지</h1>
  <form action="#" method="GET">
```

```
    {{ form.as_p }}
    <input type="submit" value="login" />
  </form>
{% endblock %}
```

서버를 켜서 화면이 잘 나오는지 확인만 해보자. http://127.0.0.1:8000/accounts/login/

끝 오늘도 수고 많으셨습니다.

[완성본]

test33.zip