Django Form

1. Form Class

- 지금까지 HTML form, input을 통해 사용자로부터 데이터를 받았으나, 정제된 데이터를 받는 데는 많은 노력이 필요함.
- Django form은 과중한 작업과 반복 코드 줄여줌으로 이 작업을 쉽게 만들어 준다.
    1. 렌더링을 위한 데이터 준비 및 재구성
    2. 데이터에 대한 HTML forms 작성
    3. 클라이언트로부터 받은 데이터 수신 및 처리
- **Form Class**
    - form내 field, field 배치, 디스플레이 widget, label, 초기값, 유효하지 않은 field에 관련된 에러 메세지를 결정

1.1 Form 선언하기

```
# articles/forms.py

from django import forms

class ArticleForm(forms.Form):
    title = forms.CharField(max_length=10)
    content = forms.CharField()
```

- forms 라이브러리에서 파생된 Form 클래스를 상속받는다.
- Model과 유사하나 다름
- import 뒤의 `forms`는 input을 가져다가 장고에서 만들어둔 형태로 쓰는 것이기 때문에 textField 같은 것 없음**(Model이랑은 다름!)**

1.2 Form 사용하기

```
# articles/views.py
from .forms import ArticleForm

def new(request):
    form = ArticleForm()
    context = {
        'form': form,
    }
    return render(request, 'articles/new.html', context)
```

```
<!--new.html-->
  <form actoion="..." method="POST" >
    ...
    {% csrf_token %}
    {{ form.as_p }}
    ...
  </form>
```

1. Form rendering options**

< label > & < input > 쌍에 대한 3가지 출력 옵션

1. as_p() : 각 필드가 p tag로 감싸져서 렌더링 됨
2. as_ul() : 각 필드가 li tag로 감싸져서 렌더링 됨. ul tag는 직접 작성
3. as_table : 각 필드가 tr tag로 감싸져서 렌더링 됨. table rag는 직접 작성

1.3 Django의 HTML 요소 표현 방법 2가지

1. Form fields

- input에 대한 유효성 검사 로직을 처리하며 템플릿에서 직접 사용됨

2. Widgets ☆

```
# articles/forms.py
class ArticleForm(forms.Form):
    ...
    content = forms.CharField(widget=forms.Textarea)
```

- 웹페이지의 HTML input 요소 렌더링
- GET/POST 딕셔너리에서 데이터 추출
- 하지만 widgets은 반드시 form fields에 할당됨
- Django의 html input element 표현
- html 렌더링을 처리

- 주의사항
    - form field와 혼동되어서는 안됨
    - Form Fields는 input의 유효성 검사를 처리
    - widgets은 웹페이지에서 input element의 단순 raw한 렌더링 처리

3. 응용

```
# articles/forms.py

class ArticleForm(forms.Form):
    REGION_A = 'sl'
    REGION_B = 'dj'
    ...
    REGION_CHOICES = [
        (REGION_A, '서울'),
        (REGION_B, '대전'),
        ...
    ]
    ...
    region = forms.ChoiceField(choices=REGION_CHOICES, widget=forms.Select())
```

- 주의 : choices 쓸 때만 위와 같은 형식 사용해야 한다

2. Model Form☆☆☆

- Article 모델이 있고 사용자가 게시글을 제출할 수 있는 양식을 만들고 싶은 경우 model에서 정의한 내용 form에서도 field 재정의 -> 중복 발생
- django는 model을 통해 Form Class 만들 수 있는 helper 제공(**ModelForm Class**)

- ModelForm Class : 일반 Form Class와 완전히 같은 방식(객체 생성)으로 view에서 사용가능

2.1 ModelForm 선언하기 : Meta class

- forms 라이브러리에서 파생된 ModelForm 클래스를 상속받음
- Class 안에 Meta Class 선언하고, 어떤 모델 기반으로 form 작성할 것인지 Meta class 안에 작성

```
# forms.py
...
from .models import Article

class ArticleForm(forms.ModelForm):
    class Meta:
        model = Article
        fields = '__all__'
```

Meta Class :

- Model의 정보를 작성하는 곳
- models.py에서 `content = models.TextField()` content를 TextField로 정의하고 있기 때문에 알아서 text field로 바꿈
- 데이터 유효성 검사는 아직 안 되고 있는 상태
- `# fields = ('title', 'content')`로 개별로 정의하거나, 상기처럼 all로 하고 제외할 것을 `# exclude = ('title',)` 할 수 있음

2.2. 데이터 유효성 검사

```
# articles/views.py

def create(request):
    form = ArticleForm(request.POST)

    # 유효성 검사
    if form. is_valid():
        article = form.save()
        return redirect('articles:detail', article.pk)
    return redirect('articles:new')
```

1. **is_valid method**

- 유효성 검사를 실행하고, 유효한지 여부를 boolean으로 반환
- `article = form.save()`: 괄호가 있다는 것은 함수라는 뜻이므로, return 값 있음

2. **save() method** : create/ update 끝내고 다시 보기

- form instance 에서 사용하는 method
- form에 바인딩 된 데이터에서 데이터 베이스 객체를 만들고 저장(반환값 有)
- ModelForm의 하위 클래스는 기존 모델 인스턴스를 키워드 인자instance 로 받아들일 수 있음
    - 제공되면 save()는 해당 인스턴트 업데이트(update)
    - 안되면 새 인스턴스 만듦(create)

- form의 유효성이 확인되지 않은 경우 save() 호출하면 form.errors를 확인하여 에러 확인 가능

```
# create a firn unstance frin POST data.
form = ArticleForm(request.POST)

# CREATE : Save a new ARticle object from the form's data.
new_article = f.save()

# UPDATE : Create a form to edit an existing Article, but use POST data to
populate the form.
article = Article.objects.get(pk=1)
form = ArtileForm(request.POST, instance=article)
form.save()
```

2.3 create view 함수 구조 변경

```
# articles/views.py
def create(request):
    # 이전 create
    if request.method == 'POST':
        form = ArticleForm(request.POST)
        if form.is_valid():
            article = form.save()
            return redirect('articles:detail', article.pk)
    # 이전 new
    else:
        form = ArticleForm()
    context = {
        'form': form,
    }
    return render(request, 'articles/create.html', context)
```

- new view 함수, url path 삭제

```
<!-- create.html -->
...
  <h1 class="text-center">CREATE</h1>
  <form action="{% url 'articles:create' %}" method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit">
  </form>
  <hr>
  <a href="{% url 'articles:index' %}">[back]</a>
{% endblock  %}
```

- new.html -> create.html로 이름 변경
- action 값 없어도 동작

```
<!-- index.html -->
  <h1>Articles</h1>
  <a href="{% url 'articles:create' %}">[CREATE]</a>
  <hr>
  ...
```

- index.html에 create 페이지로 링크 작성
- input칸에 공백 데이터 넣으면 에러메시지 "필수 항목입니다." 나오는 것 확인

1.

2. GET 먼저 분기했을 때

3. widget 적용하기

- django의 html input element 표현
- HTML 렌더링을 처리
- 2가지 작성 방식을 가짐

아래는 두 번째 방식.

```
# forms.py

class ArticleForm(forms.ModelForm):
    title = forms.CharField(
        label ='제목',
        widget = forms.TextInput(
            attrs={
                'class': 'my_title',
                'placeholder': 'Enter the title',
                'maxlength': 10,
            }
        )
    )
    content = forms.CharField(
        label ='내용',
        widget = forms.Textarea(
            attrs={
                'class': 'my_content',
                'placeholder': 'Enter the content',
                'rows': 5,
                'cols': 50,
            }
        )
    )
    class Meta:
        ...
```

2.4 DELETE 로직

```
def delete(request, pk):
    article = Article.objects.get(pk=pk)
    if request.method == 'POST':
        article.delete()
        return redirect('articles:index')
    return redirect('articles:detao;', article.pk)
```

달라진 것 없음...! (나중에 `get_object_or_404` 쓰면 달라짐)

2.5 UPDATE view 함수 구조 변경하기

instance = article ☆

원래는 data =request.POST 써야되지만, 첫 번째 인자여서 data 생략됨.

```
def update(request, pk):
    article = Article.objects.get(pk=pk)
    # update
    if request.method == 'POST':
        form = ArticleForm(request.POST, instance=article)
        if form.is_valid():
            form.save()
            return redirect('articles:detail', article.pk)
    # 이전 edit
    else:
        form = ArticleForm(instance=article)
    context = {
        'article': article,
        'form': form,
    }
    return render(request, 'articles/edit.html', context)
```

- `instance=article` 을 안 넣을 경우, 계속 새로운 것 생성됨. 없는 경우, 기존 내역 볼 수 없음
- update view함수 작성

```
<!-- update.html -->
  ...
  <h1>EDIT</h1>
  <form action="{% url 'articles:update' article.pk %}" method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit">
  </form>
  ...
```

- edit -> update.html 파일명 변경

2.6 Form & ModelForm

1. Form

- 어떤 model에 저장해야하는지 알 수 없으므로 유효성 검사 이후 딕셔너리 생성
- 데이터 가져온 후 .save() 호출해야 함
- model에 연관되지 않은 데이터를 받을 때 사용

2. ModelForm

- django가 해당 model에서 양식에 필요한 대부분의 정보를 이미 정의
- 어떤 레코드를 만들어야 할 지 알고 있으므로 바로 .save() 호출 가능
- Model과 연관된 데이터 받을 때 도움을 주기 위한 helper의 역할

3. Rendering fields manually

3.1 Rendering fields manually

```
<!-- articles/create.html-->
...
<form action"" method='GET'>
    {{ form.subject.errors }}
</form>
```

3.2 Django Bootstrap Liabrary

4. Handling HTTP request

- Django에서 HTTP 요청을 처리하는 방법
    1. Django shortcut funtions

4.1 Django shortcut functions

1. get_object_or_404

- 모델 manager objects 에서 get()을 호출하지만, 해당 객체가 없을 경우 DoesNotExist 예외 대신 Http 404를 raise
- get() 메서드의 경우 조건에 맞는 데이터가 없을 경우 에러 발생하고 500으로 인식

```
# views.py 에서 get object 할 수 있는 것들은 다 바꿀 수 있음
# detail, delete, update 다 바꿈!
def detail(request, pk):
    # article = Article.objects.get(pk=pk)
    article= get_object_or_404(Article, pk=pk)
    context = {
        'article': article,
    }
    return render(request, 'articles/detail.html', context)
```

2. Django View decorators

- Allowed HTTP methods
    - 요청 충족시키지 못하면 HttpResponseNotAllowed(405) return
    1. require_http_methods() : `@require_http_methods(['GET', 'POST'])`
    2. require_POST(): `@require_POST` : delete에서 필요하고, if 문 삭제가능
- required_http_methods()
    - view함수가 특정한 method요청에 대해서만 허용하도록 하는 데코레이터
    - 예시에서 detail / create 함수에 적용함

```
@require_http_methods(['GET', 'POST'])
def detail(request, pk):
    # article = Article.objects.get(pk=pk)
    article= get_object_or_404(Article, pk=pk)
    context = {
        'article': article,
    }
    return render(request, 'articles/detail.html', context)
```

- require_POST()
    - view함수가 POST method요청만 승인하도록 하는 데코레이터

```
@require_POST
def delete(request, pk):
    # article = Article.objects.get(pk=pk)
    article= get_object_or_404(Article, pk=pk)
    article.delete()
    return redirect('articles:index')
```