31 December 2019

# Django Forms

As well as its <input> elements, a form must specify two things:

• *where*: the URL to which the data corresponding to the user's input should be returned ( using the action attribute )

• *how*: the HTTP method the data should be returned by (using the method attribute)

At the basic level we have Django forms class

```
from django import forms

class NameForm(forms.Form):
        your_name = forms.CharField(label='Your name', max_length=100)
```

The label represents a human friendly of the field.

A form instance has an is_valid() method which runs validation for all its fields.

**This is how it will be rendered**

```
<label for="your_name">Your name: </label>
<input id="your_name" type="text" name="your_name" maxlength="100" required>
```

Handling it in our view

```
# create a form instance and populate it with data from the request:
form = NameForm(request.POST)

# check whether it's valid:

if form.is_valid():
```

Whatever the data submitted with a form, once it has been successfully validated by calling is_valid() (and is_valid() has returned True), the validated form data will be in the form.cleaned_data dictionary. This data will have been nicely converted into Python types for you.

So before accessing form values use is_valid() and then access it from the .cleaned_data dictionary.

Django allows us to render fields manually

```
{{ form.non_field_errors }} <div class="fieldWrapper">

{{ form.subject.errors }}
<label for="{{ form.subject.id_for_label }}">Email subject:</label> {{ form.subject }}

</div>
<div class="fieldWrapper">

{{ form.message.errors }}
<label for="{{ form.message.id_for_label }}">Your message:</label> {{ form.message }}

</div>
<div class="fieldWrapper">

{{ form.sender.errors }}
<label for="{{ form.sender.id_for_label }}">Your email address:</label> {{ form.sender }}

</div>
<div class="fieldWrapper">
```

## Creating forms from models

ModelForm

```
# Create the form class.

>>> class ArticleForm(ModelForm):

...

...

class Meta:
model = Article

fields = ['pub_date', 'headline', 'content', 'reporter'] # Creating a form to add an article.

>>> form = ArticleForm()
```

# Creating a form to change an existing article.

>>> article = Article.objects.get(pk=1) >>> form = ArticleForm(instance=article)

To reflect the changes in the database use form.save()

If we pass commit=False as params to save then the form instance won't be saved and instead would return model instance.

Issue arises if we want to save m2m field and commit = False.

Selecting fields to use

fields = '__all__'

exclude = ['title']

For updates, the user may not provide all the values. Then we need to populate the field with database values.

author = Author(title='Mr')
form = PartialAuthorForm(request.POST, instance=author) form.save()

## A FORMSET IS USED TO DEAL WITH MULTIPLE FORMS IN A SINGLE PAGE