
بسم الله

DJANGO FORMS

HOSSEIN FORGHANI

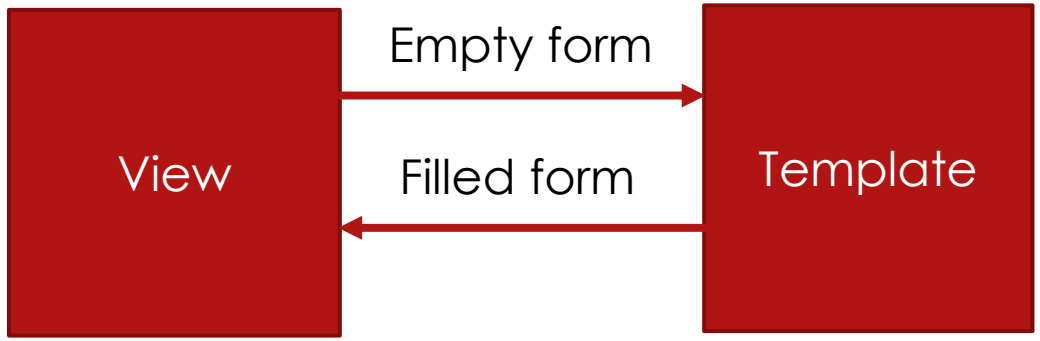
MAKTAB SHARIF

Contents

- How to use forms
- How to use multiple forms: formset
- How to use a form to fill data of an instance of model: ModelForm

Django's role in forms

- preparing and restructuring data to make it ready for rendering
- creating HTML forms for the data
- receiving and processing submitted forms and data from the client



Building a form in Django

forms.py

```
from django import forms

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

Validates at both client and server sides

View:

views.py

```python
from django.http import HttpResponseRedirect
from django.shortcuts import render

from .forms import NameForm

def get_name(request):
    # if this is a POST request we need to process the form
data
    if request.method == 'POST':
        # 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():
            # process the data in form.cleaned_data as
required
            # ...
            # redirect to a new URL:
            return HttpResponseRedirect('/thanks/')

    # if a GET (or any other method) we'll create a blank form
    else:
        form = NameForm()

    return render(request, 'name.html', {'form': form})
```

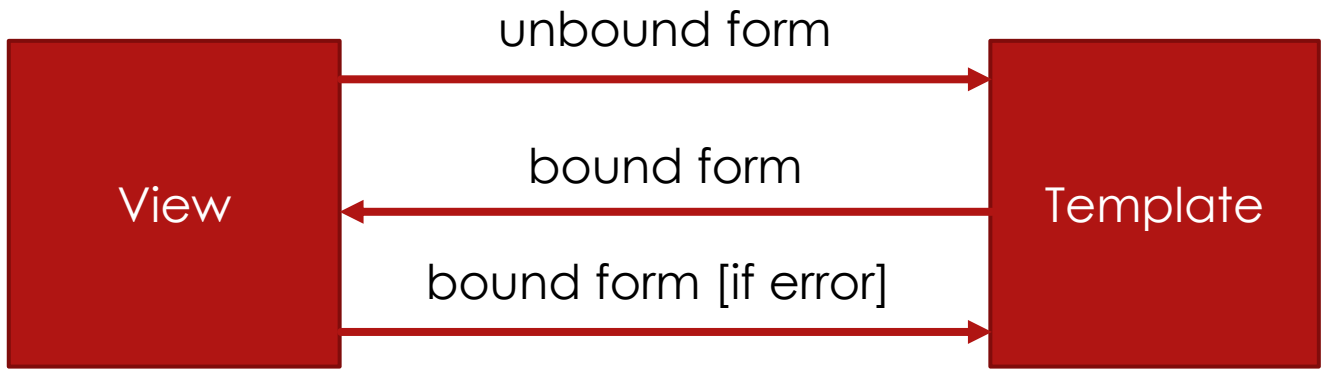
Template

```
<form action="/your-name/" method="post">
    {% csrf_token %}
    {{ form }}
    <input type="submit" value="Submit">
</form>
```

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

Bound and Unbound form instances

- unbound form has no data associated with it
- bound form has submitted data, and hence can be used to tell if that data is valid



More on Fields

forms.py

```python
from django import forms

class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
    message = forms.CharField(widget=forms.Textarea)
    sender = forms.EmailField()
    cc_myself = forms.BooleanField(required=False)
```

Other Field Options

- error_messages:

```
name = forms.CharField(error_messages={'required': 'Please enter your name'})
```

- validators: Just like model field valiators

Field Data

- Once it has been successfully validated by calling is_valid() the validated form data will be in the form.cleaned_data dictionary

views.py

```
from django.core.mail import send_mail

if form.is_valid():
    subject = form.cleaned_data['subject']
    message = form.cleaned_data['message']
    sender = form.cleaned_data['sender']
    cc_myself = form.cleaned_data['cc_myself']

    recipients = ['info@example.com']
    if cc_myself:
        recipients.append(sender)

    send_mail(subject, message, sender, recipients)
    return HttpResponseRedirect('/thanks/')
```

Working with form templates

- {{ form }} will render its <label> and <input> simply
- {{ form.as_table }} will render them as table cells wrapped in <tr> tags
- {{ form.as_p }} will render them wrapped in <p> tags
- {{ form.as_ul }} will render them wrapped in <li> tags

Rendering 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">
    {{ form.cc_myself.errors }}
    <label for="{{ form.cc_myself.id_for_label }}">CC yourself?
</label>
    {{ form.cc_myself }}
</div>
```

Can be replaced with

```
{{ form.subject.label_tag }}
```

Further Customize

```
{{ form.subject.errors }}
```

```
{% if form.subject.errors %}
    <ol>
    {% for error in form.subject.errors %}
        <li><strong>{{ error|escape }}</strong></li>
    {% endfor %}
    </ol>
{% endif %}
```

Looping Over the Form's Fields

```
{% for field in form %}
    <div class="fieldWrapper">
        {{ field.errors }}
        {{ field.label_tag }} {{ field }}
        {% if field.help_text %}
        <p class="help">{{ field.help_text|safe }}</p>
        {% endif %}
    </div>
{% endfor %}
```

Looping over Hidden and Visible Fields

```
{# Include the hidden fields #}
{% for hidden in form.hidden_fields %}
{{ hidden }}
{% endfor %}
{# Include the visible fields #}
{% for field in form.visible_fields %}
    <div class="fieldWrapper">
        {{ field.errors }}
        {{ field.label_tag }} {{ field }}
    </div>
{% endfor %}
```

Basic File Upload

```
<form enctype="multipart/form-data" method="post" action="/foo/">
```

forms.py

```
from django import forms

class UploadFileForm(forms.Form):
    title = forms.CharField(max_length=50)
    file = forms.FileField()
```

views.py

```python
from django.http import HttpResponseRedirect
from django.shortcuts import render
from .forms import UploadFileForm

# Imaginary function to handle an uploaded file.
from somewhere import handle_uploaded_file

def upload_file(request):
    if request.method == 'POST':
        form = UploadFileForm(request.POST, request.FILES)
        if form.is_valid():
            handle_uploaded_file(request.FILES['file'])
            return HttpResponseRedirect('/success/url/')
    else:
        form = UploadFileForm()
    return render(request, 'upload.html', {'form': form})
```

Basic File Upload – cont.

```
def handle_uploaded_file(f):
    with open('some/file/name.txt', 'wb+') as destination:
        for chunk in f.chunks():
            destination.write(chunk)
```

Formsets

What is Formset?

- A formset is a layer of abstraction to work with multiple forms on the same page

Change person

History

Name

FirstName: Thor

LastName: Thad

Contact info

Email: thor.thad@xyz.com

PhoneNumber: 1-555-192-1231

Attribute values

Attribute	IsConsumer	Value	Delete?
AttributeValue object			
Requisition	☐	Medicine	☐
---------	☐		
---------	☐		
---------	☐		

Add another Attribute Value

Creating Formset

- Let's say you have the following form:

```
>>> from django import forms
>>> class ArticleForm(forms.Form):
...     title = forms.CharField()
...     pub_date = forms.DateField()
```

Creating Formset – cont.

- Then:

```
>>> from django.forms import formset_factory
>>> ArticleFormSet = formset_factory(ArticleForm)

>>> formset = ArticleFormSet()
>>> for form in formset:
...     print(form.as_table())
<tr><th><label for="id_form-0-title">Title:</label></th><td>
<input type="text" name="form-0-title" id="id_form-0-title"></td>
</tr>
<tr><th><label for="id_form-0-pub_date">Pub date:</label></th>
<td><input type="text" name="form-0-pub_date" id="id_form-0-
pub_date"></td></tr>
```

Number of Empty Forms

```
>>> ArticleFormSet = formset_factory(ArticleForm, extra=2)
```

Using initial data with a formset

```
>>> import datetime
>>> from django.forms import formset_factory
>>> from myapp.forms import ArticleForm
>>> ArticleFormSet = formset_factory(ArticleForm, extra=2)
>>> formset = ArticleFormSet(initial=[
...     {'title': 'Django is now open source',
...      'pub_date': datetime.date.today(),}
... ])

>>> for form in formset:
...     print(form.as_table())
<tr><th><label for="id_form-0-title">Title:</label></th><td><input
type="text" name="form-0-title" value="Django is now open source"
id="id_form-0-title"></td></tr>
<tr><th><label for="id_form-0-pub_date">Pub date:</label></th><td>
<input type="text" name="form-0-pub_date" value="2008-05-12"
id="id_form-0-pub_date"></td></tr>
<tr><th><label for="id_form-1-title">Title:</label></th><td><input
type="text" name="form-1-title" id="id_form-1-title"></td></tr>
<tr><th><label for="id_form-1-pub_date">Pub date:</label></th><td>
<input type="text" name="form-1-pub_date" id="id_form-1-pub_date">
</td></tr>
<tr><th><label for="id_form-2-title">Title:</label></th><td><input
type="text" name="form-2-title" id="id_form-2-title"></td></tr>
<tr><th><label for="id_form-2-pub_date">Pub date:</label></th><td>
<input type="text" name="form-2-pub_date" id="id_form-2-pub_date">
</td></tr>
```

Processing Back from Template

- If you use an initial for displaying a formset, you should pass the same initial when processing that formset's submission so that the formset can detect which forms were changed by the user:

```
ArticleFormSet(request.POST, initial=[...])
```

Limiting the maximum number of forms

```
>>> from django.forms import formset_factory
>>> from myapp.forms import ArticleForm
>>> ArticleFormSet = formset_factory(ArticleForm, extra=2,
max_num=1)
>>> formset = ArticleFormSet()
>>> for form in formset:
...     print(form.as_table())
<tr><th><label for="id_form-0-title">Title:</label></th><td>
<input type="text" name="form-0-title" id="id_form-0-title"></td>
</tr>
<tr><th><label for="id_form-0-pub_date">Pub date:</label></th>
<td><input type="text" name="form-0-pub_date" id="id_form-0-
pub_date"></td></tr>
```

Formset validation

- Identical to a regular Form: `formset.is_valid()`
- The formset is smart enough to ignore extra forms that were not changed

Ordering and Deletion of Forms

- Lets you create a formset with the ability to order:

```
ArticleFormSet = formset_factory(ArticleForm, can_order=True)
```

- Lets you create a formset with the ability to select forms for deletion:

```
ArticleFormSet = formset_factory(ArticleForm, can_delete=True)
```

Using a formset in views

```
from django.forms import formset_factory
from django.shortcuts import render
from myapp.forms import ArticleForm

def manage_articles(request):
    ArticleFormSet = formset_factory(ArticleForm)
    if request.method == 'POST':
        formset = ArticleFormSet(request.POST, request.FILES)
        if formset.is_valid():
            # do something with the formset.cleaned_data
            pass
    else:
        formset = ArticleFormSet()
    return render(request, 'manage_articles.html', {'formset':
formset})
```

Using Formset in Template

```
<form method="post">
    {{ formset.management_form }}
    <table>
        {% for form in formset %}
        {{ form }}
        {% endfor %}
    </table>
</form>
```

OR

```
<form method="post">
    <table>
        {{ formset }}
    </table>
</form>
```

This form is used by the formset to manage the collection of forms contained in the formset

Using more than one formset in a view

```
from django.forms import formset_factory
from django.shortcuts import render
from myapp.forms import ArticleForm, BookForm

def manage_articles(request):
    ArticleFormSet = formset_factory(ArticleForm)
    BookFormSet = formset_factory(BookForm)
    if request.method == 'POST':
        article_formset = ArticleFormSet(request.POST,
request.FILES, prefix='articles')
        book_formset = BookFormSet(request.POST, request.FILES,
prefix='books')
        if article_formset.is_valid() and book_formset.is_valid():
            # do something with the cleaned_data on the formsets.
            pass
    else:
        article_formset = ArticleFormSet(prefix='articles')
        book_formset = BookFormSet(prefix='books')
    return render(request, 'manage_articles.html', {
        'article_formset': article_formset,
        'book_formset': book_formset,
    })
```

Model Forms

When to Use Model Form?

- If you have forms that map closely to Django models
- In this case, it would be redundant to define the field types in your form, because you've already defined the fields in your model

Creating Model Form

```
>>> from django.forms import ModelForm
>>> from myapp.models import Article

# 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)
```

A Full Example

```python
from django.db import models
from django.forms import ModelForm

TITLE_CHOICES = [
    ('MR', 'Mr.'),
    ('MRS', 'Mrs.'),
    ('MS', 'Ms.'),
]

class Author(models.Model):
    name = models.CharField(max_length=100)
    title = models.CharField(max_length=3, choices=TITLE_CHOICES)
    birth_date = models.DateField(blank=True, null=True)

    def __str__(self):
        return self.name

class Book(models.Model):
    name = models.CharField(max_length=100)
    authors = models.ManyToManyField(Author)
```

A Full Example

```python
class AuthorForm(ModelForm):
    class Meta:
        model = Author
        fields = ['name', 'title', 'birth_date']

class BookForm(ModelForm):
    class Meta:
        model = Book
        fields = ['name', 'authors']
```

Equivalent to

```python
from django import forms

class AuthorForm(forms.Form):
    name = forms.CharField(max_length=100)
    title = forms.CharField(
        max_length=3,
        widget=forms.Select(choices=TITLE_CHOICES),
    )
    birth_date = forms.DateField(required=False)

class BookForm(forms.Form):
    name = forms.CharField(max_length=100)
    authors =
forms.ModelMultipleChoiceField(queryset=Author.objects.all())
```

Validation on a ModelForm

- Just like normal form validation: calling is_valid() or accessing errors
- There are two main steps involved in validating a ModelForm:
  - Validating the form
  - Validating the model instance

The save() method

- Updates or Creates a new instance
- if the form hasn't been validated, calling save() will do so

```
>>> from myapp.models import Article
>>> from myapp.forms import ArticleForm

# Create a form instance from POST data.
>>> f = ArticleForm(request.POST)

# Save a new Article object from the form's data.
>>> new_article = f.save()

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

Commit Argument

- If you call save() with commit=False, then it will return an object that hasn't yet been saved to the database
- In order to modify the instance before saving
- If your model has a many-to-many relation and you specify commit=False when you save a form, you must call save_m2m() after saving the instance

save_m2m()

```
# Create a form instance with POST data.
>>> f = AuthorForm(request.POST)

# Create, but don't save the new author instance.
>>> new_author = f.save(commit=False)

# Modify the author in some way.
>>> new_author.some_field = 'some_value'

# Save the new instance.
>>> new_author.save()

# Now, save the many-to-many data for the form.
>>> f.save_m2m()
```

Selecting the fields to use

```python
from django.forms import ModelForm

class AuthorForm(ModelForm):
    class Meta:
        model = Author
        fields = '__all__'
```

```python
class PartialAuthorForm(ModelForm):
    class Meta:
        model = Author
        exclude = ['title']
```

Overriding the default fields

```
from django.forms import ModelForm, Textarea
from myapp.models import Author

class AuthorForm(ModelForm):
    class Meta:
        model = Author
        fields = ('name', 'title', 'birth_date')
        widgets = {
            'name': Textarea(attrs={'cols': 80, 'rows': 20}),
        }
```

Overriding the default fields

```python
from django.utils.translation import gettext_lazy as _

class AuthorForm(ModelForm):
    class Meta:
        model = Author
        fields = ('name', 'title', 'birth_date')
        labels = {
            'name': _('Writer'),
        }
        help_texts = {
            'name': _('Some useful help text.'),
        }
        error_messages = {
            'name': {
                'max_length': _("This writer's name is too
long."),
            },
        }
```

Specify a ModelField's validators

```
from django.forms import CharField, ModelForm
from myapp.models import Article

class ArticleForm(ModelForm):
    slug = CharField(validators=[validate_slug])

    class Meta:
        model = Article
        fields = ['pub_date', 'headline', 'content', 'reporter',
'slug']
```

Another Way of Customizing Validation

- clean_<field_name> accesses cleaned_data
- Raises ValidationError in cases of invalid data, otherwise the value

```
class AuthorForm(forms.ModelForm):
    class Meta:
        model = Author
        fields = ('name', 'title')

    def clean_name(self):
        # custom validation for the name field
        ...
```

Handling uploaded files with a model

```
from django.http import HttpResponseRedirect
from django.shortcuts import render
from .forms import ModelFormWithFileField

def upload_file(request):
    if request.method == 'POST':
        form = ModelFormWithFileField(request.POST,
request.FILES)
        if form.is_valid():
            # file is saved
            form.save()
            return HttpResponseRedirect('/success/url/')
    else:
        form = ModelFormWithFileField()
    return render(request, 'upload.html', {'form': form})
```

References

- https://docs.djangoproject.com/en/3.1/topics/forms/
- https://docs.djangoproject.com/en/3.1/topics/forms/formsets/
- https://docs.djangoproject.com/en/3.1/topics/forms/modelforms/
- https://docs.djangoproject.com/en/3.1/topics/http/file-uploads/

Any Question?