

Django Forms Advanced



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://softuni.bg>

sli.do

#python-web

1. Validating **Forms** in **Django**
 - Validation in **Forms** and **ModelForms**
 - Overriding **Error Messages**
2. Form **Class Methods**
3. Formsets
4. Styling **Forms**
5. Working with **Media Files** - Demo





Validating Forms in Django

Django Validators

- A **validator** is a **function** or **class** designed to **assess**
 - whether a given **value** meets specified **criteria**
- If the **value** satisfies all **criteria**, the validator does **not return anything**
- If any **criteria** are **not met**, it raises a **ValidationError**

validators.py

```
from django.core.exceptions import ValidationError

def validate_value(value):
    # If the value does not meet the criteria:
    raise ValidationError("Some Error Message")
```



Django Validators Reusability

- You can **efficiently reuse** validation logic across different types of fields in Django
- This reusability **extends** to
 - **Models:** apply the **same validation logic** to fields
 - **Forms:** **validation logic** can be **shared** among fields
 - **ModelForms:** **reuse** the **same validation logic** in the context of a Django ModelForm



Validating Forms

- **Form validation** occurs during the **data cleaning** process in Django
 - Raises **ValidationError** and provides relevant **information** about the **error**
 - The **cleaned** and **normalized** data is **returned** as a Python object
- Each **form field** is equipped with **custom validation** logic **tailored** to its specific requirements



Form Validators

- You can pass **additional validators** to a Form field

```
class NameForm(forms.Form):  
    name = forms.CharField(  
        validators=[validator_one, validator_two, ...]  
    )
```

- You can use both:
 - **Custom validators**
 - **Built-in Django validators**



ModelForm Validation

- In a Django **ModelForm**, you can implement validation in two key aspects
 - Validating the **Model**
 - Ensure that the **data** adheres to the **validation** rules specified in the **corresponding Django model**
 - Validating the **Form**
 - Implement **custom validation** logic specific to the form
 - Considering any **additional criteria** or **constraints beyond** those defined in the model



- Pass additional validators to the **Model** field

```
class Name(models.Model):  
    first_name = models.CharField(  
        max_length=20,  
        validators=[  
            validator_one,  
            validator_two,  
            ...  
        ]  
    )
```

Error Messages in Forms


- You can **customize** the **error messages** associated with an existing validator
 - by **overriding** the default **messages**
- Each validator has a list of **error message keys**
- Pass in a **dictionary** with **keys** and **error messages**

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



Error Messages in Models


- You can **override** the **error messages** in the **model**



```
class UserName(models.Model):
    username = models.CharField(
        max_length=50,
        unique=True,
        error_messages={
            "unique": "The name is already taken."
        })
```

Error Messages in ModelForms

- You can **override** the **error messages** in the **ModelForm**



```
class NameModelForm(forms.ModelForm):
    class Meta:
        ...
        error_messages = {
            'name': {
                'max_length': "The name is too long."
            }
        }
```



Form Class Methods

__init__() Method

- `__init__(self, *args, **kwargs)`
- This method is the **constructor** for the form class
- Used to **initialize** the form and can be **overridden** to perform any **setups**

```
from django import forms
```

```
class NameForm(forms.ModelForm):
```

```
    def __init__(self, *args, **kwargs):
```

```
        super().__init__(*args, **kwargs)
```

```
        # Custom initialization code
```

```
        for field_name in self.fields:
```

```
            self.fields[field_name].widget.attrs['readonly'] = 'readonly'
```

```
            self.fields[field_name].required = False
```

- **clean(self)**
- This method is used for overall form **cleaning/validation** that involves **multiple** fields
- It's called **after** all **individual** field clean methods

```
def clean(self):  
    cleaned_data = super(NameForm, self).clean()  
    # Custom form validation logic  
    if cleaned_data['first_name'] == cleaned_data['last_name']:  
        raise ValidationError('Incorrect names')  
    return cleaned_data
```


clean_<fieldname>() Method

- `clean_<fieldname>(self)`
- This method is used to implement **custom** cleaning/validation for a **specific** field
- It's **automatically** called during form **validation**

```
def clean_email(self):  
    email = self.cleaned_data.get('email')  
    # Custom cleaning/validation field-specific logic  
    if email and not email.endswith('@example.com'):  
        raise ValidationError('Wrong domain')  
    return email
```

- **is_valid(self)**
 - This method is used to **check** if the form data **is valid**
 - It **triggers** the **validation** of **all fields** and returns **True** if the form is valid, otherwise **False**
- **save(self, commit=True)**
 - If your form is a **ModelForm** (associated with a **Django model**), this method is used to
 - **save** the form **data** to the **database**

Form Instance Methods - Example

```
# forms.py
```

```
class MyModelForm(forms.ModelForm):  
    class Meta:  
        model = MyModel  
        ...
```

```
# views.py
```

```
...  
my_form = MyModelForm(request.POST)  
if my_form.is_valid():  
    instance = my_form.save()  
# This saves the form data to the database
```



Formsets

Django Formset



- **Formset** is a layer of **abstraction** to work with
 - **Multiple** forms on the **same** page
- It allows you to **manage** and **process** **multiple** forms **simultaneously**
- Django provides a **formset** class to handle this

Using Formsets

```
# models.py
from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()

# forms.py
from django import forms
from .models import Person

class PersonForm(forms.ModelForm):
    class Meta:
        model = Person
        fields = ['name', 'email']
```

```
# forms.py
```

```
from django.forms import modelformset_factory
```

Creating a formset for the
PersonForm

The formset will include forms
for two persons

```
PersonFormSet = modelformset_factory(Person,  
                                     form=PersonForm, extra=2)
```

```
# views.py
from django.shortcuts import render, redirect
from .forms import PersonFormSet

def manage_people(request):
    if request.method == 'POST':
        formset = PersonFormSet(request.POST, queryset=Person.objects.all())
        if formset.is_valid():
            formset.save()
            # Do something upon successful form submission
            return redirect('success_page')
    else:
        formset = PersonFormSet(queryset=Person.objects.all())

    return render(request, 'manage_people.html', {'formset': formset})
```



```
<!-- manage_people.html -->

<form method="post" action="">
  {% csrf_token %}
  {{ formset.management_form }}
  {% for form in formset %}
    {{ form.as_p }}
  {% endfor %}
  <input type="submit" value="Save">
</form>
```



Styling Forms

- Can be applied on a form or formset
 - `as_p`, `as_div`, `as_table`, `as_ul`

```
<!-- manage_people.html -->  
  
<form method="post" action="">  
    {% csrf_token %}  
    {{ formset.as_div }}  
    <input type="submit" value="Save">  
</form>
```

```
<form>
  <div class="form-group">
    <label for="email">Email
      address
    </label>
    <input type="email"
      class="form-control"
      id="email"
      placeholder="Enter email"
    >
  </div>
  <button type="submit"
    class="btn btn-
      primary">Submit</button>
</form>
```



```
from django import forms

class PersonForm(forms.ModelForm):
    def __init__(self, *args,
      **kwargs):
        super().__init__(*args,
      **kwargs)
        # Add a class attribute
        self.fields['email'].widget.attrs[
          'class'] = 'form-control'
        # Add a placeholder
        self.fields['email'].widget.attrs[
          'placeholder'] = 'Enter email'
        ...
```

- Install **Crispy Forms**

```
pip install django-crispy-forms
```

- Add **Crispy Forms** to your **installed Apps**

```
INSTALLED_APPS = (  
    ...  
    'crispy_forms',  
)
```

- You can use an **instance-level helper** to **customize** various aspects of the form's rendering

```
from crispy_forms.helper import FormHelper

class ExampleForm(forms.Form):
    ...
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.helper = FormHelper()
        self.helper.form_id = 'id-exampleForm'
        self.helper.form_class = 'blueForms'
        ...
```

```
<!-- Django template -->
```

```
{% load crispy_forms_tags %}
```

```
<form action="{% url 'url_name' %}"  
      class="my-class" method="post">
```

```
    {% crispy form %}
```

```
</form>
```

The `{% crispy form %}`
tag includes the CSRF
token by default



Working with Media Files

Live Demo

What are Media Files?



- Media files encompass a variety of **digital content**
 - **pictures, music, audio, videos, and documents**
- These files are typically **encoded** during the saving process allowing computer programs or applications
 - to **read** and **work** with them
- For example, **document formats** can be easily read and edited in word-processing programs

- **Image** file formats: JPEG, GIF, TIFF, BMP
- **Audio** file formats: AAC, MP3, WAV, WMA, DOLBY DIGITAL, DTS
 - Other available **audio** file formats: AIFF, ASF, FLAC, ADPCM, DSD, LPCM, OGG
- **Video** file formats: MPEG-1, MPEG-2, MPEG-4, AVI, MOV, AVCHD, H.264, H.265
 - Other available **video** formats: DivX and DivX HD, Xvid HD, MKV, RMVB, WMV9, TS/TP/M2T, WMV

Pillow - Python Imaging Library



- The **P**ython **I**maging **L**ibrary, often abbreviated as **PIL** (and known as **Pillow** in newer versions), is a free and open-source library
- It **enhances** Python's **capabilities** by providing **support** for opening, manipulating, and saving a wide range of **image file formats**
- **Compatible** with Windows, Mac OS X, and Linux
- Among the **supported formats** are PPM, PNG, JPEG, GIF, TIFF, and BMP

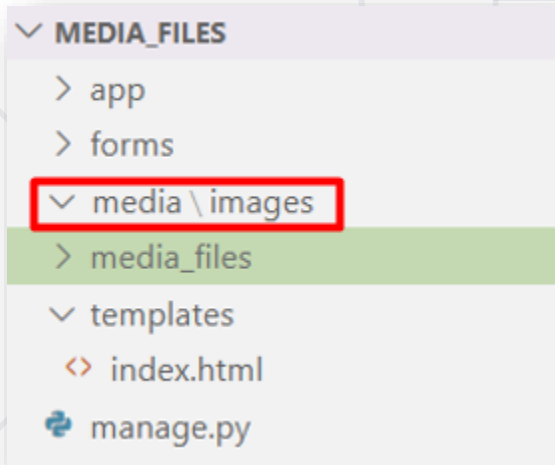
- To install **Pillow**, we can use the Python package manager (pip)

```
pip install pillow
```

- Warnings
 - Pillow and PIL **cannot co-exist** in the same environment
 - Before installing Pillow, please **uninstall PIL**

Configuring Media Folder

- Create a **media** folder and configure it in the **settings.py** file

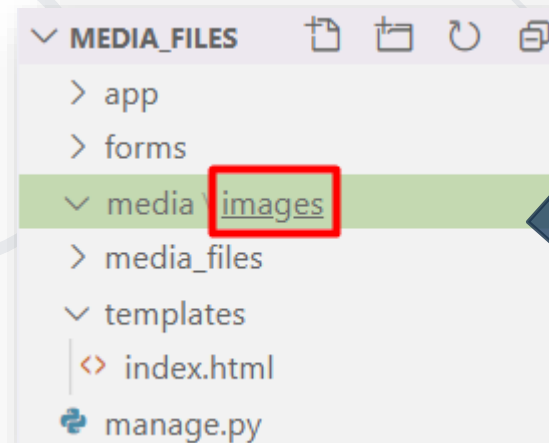


```
117
118     USE_L10N = True
119
120     USE_TZ = True
121
122
123     # Static files (CSS, JavaScript, Images)
124     # https://docs.djangoproject.com/en/3.0/howto/static-files/
125
126     STATIC_URL = '/static/'
127
128     MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
129     MEDIA_URL = '/media/'
130
```

Create an Image Field in a Model

```
app > models.py > Image
1 from django.db import models
2
3 # Create your models here.
4 class Image(models.Model):
5     image = models.ImageField(upload_to="images")
```

Name of the folder where
the images will be stored



Create a Model Form

```
forms > image_form.py > ImageForm > Meta
1 from django.forms import ModelForm
2 from app.models import Image
3
4 class ImageForm(ModelForm):
5     class Meta:
6         model = Image
7         fields = '__all__'
```

Add new cat image

Image: No file chosen

```
<form enctype="multipart/form-data", method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit">
</form>
```

Handling the POST Request

```
11 elif req.method == "POST"
12     form = ImageForm(req.POST, req.FILES)
13     if form.is_valid():
14         image = form.save()
15         image.save()
16     return render(req, 'index.html', {'images': images})
```

```
3 from django.conf import settings
4 from django.conf.urls.static import static
5
6
7 urlpatterns = [
8     path('', index, name="index")
9 ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

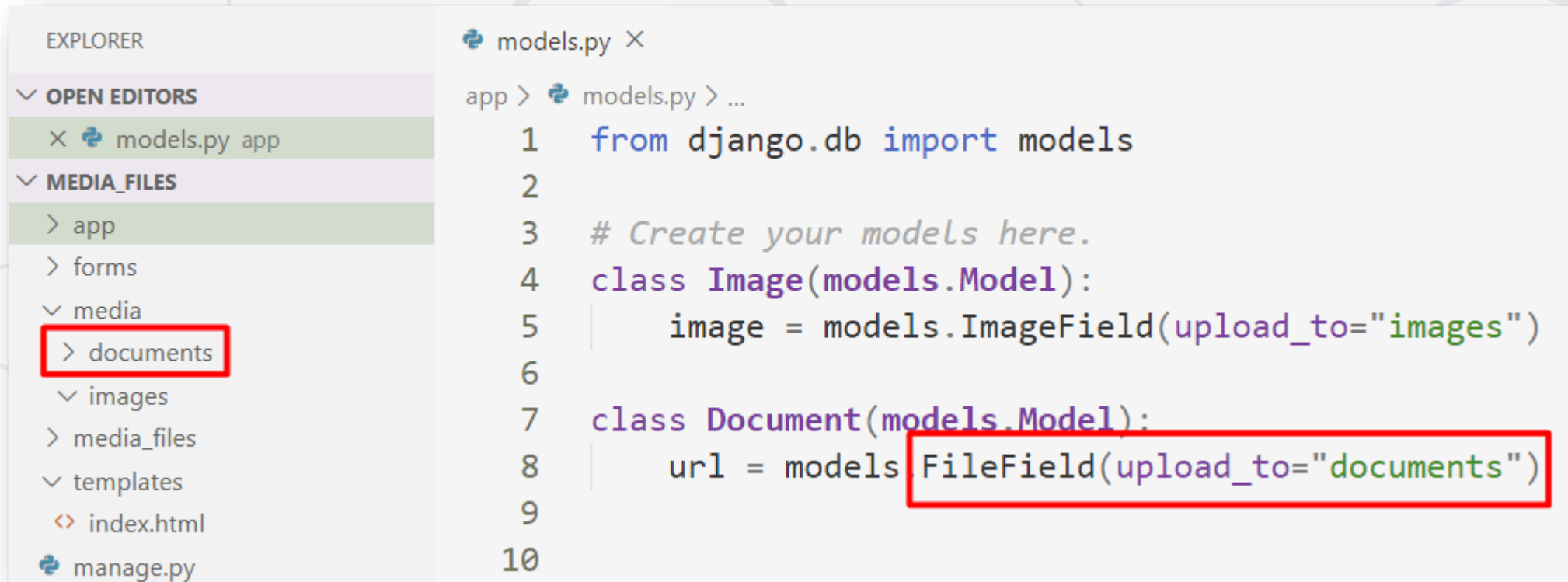
Configure the
URLs for media

Displaying the Image

```
1 <div class="card m-3" style="width: 18rem;">  
2     
3 </div>
```



Create a Documents Folder



The screenshot shows a code editor interface with two main panels. The left panel, titled 'EXPLORER', displays a file tree. Under the 'MEDIA_FILES' section, the 'documents' folder is highlighted with a red rectangle. The right panel, titled 'models.py', shows the following Python code:

```
app > models.py > ...  
1  from django.db import models  
2  
3  # Create your models here.  
4  class Image(models.Model):  
5      image = models.ImageField(upload_to="images")  
6  
7  class Document(models.Model):  
8      url = models.FileField(upload_to="documents")  
9  
10
```

In the code, the `upload_to="documents"` parameter in the `FileField` definition is highlighted with a red rectangle.

Create Model Form

document_form.py X

forms > document_form.py > ...

```
1 from django.forms import ModelForm
2 from app.models import Document
3
4 class DocumentForm(ModelForm):
5     class Meta:
6         model = Document
7         fields = '__all__'
```

Add new document

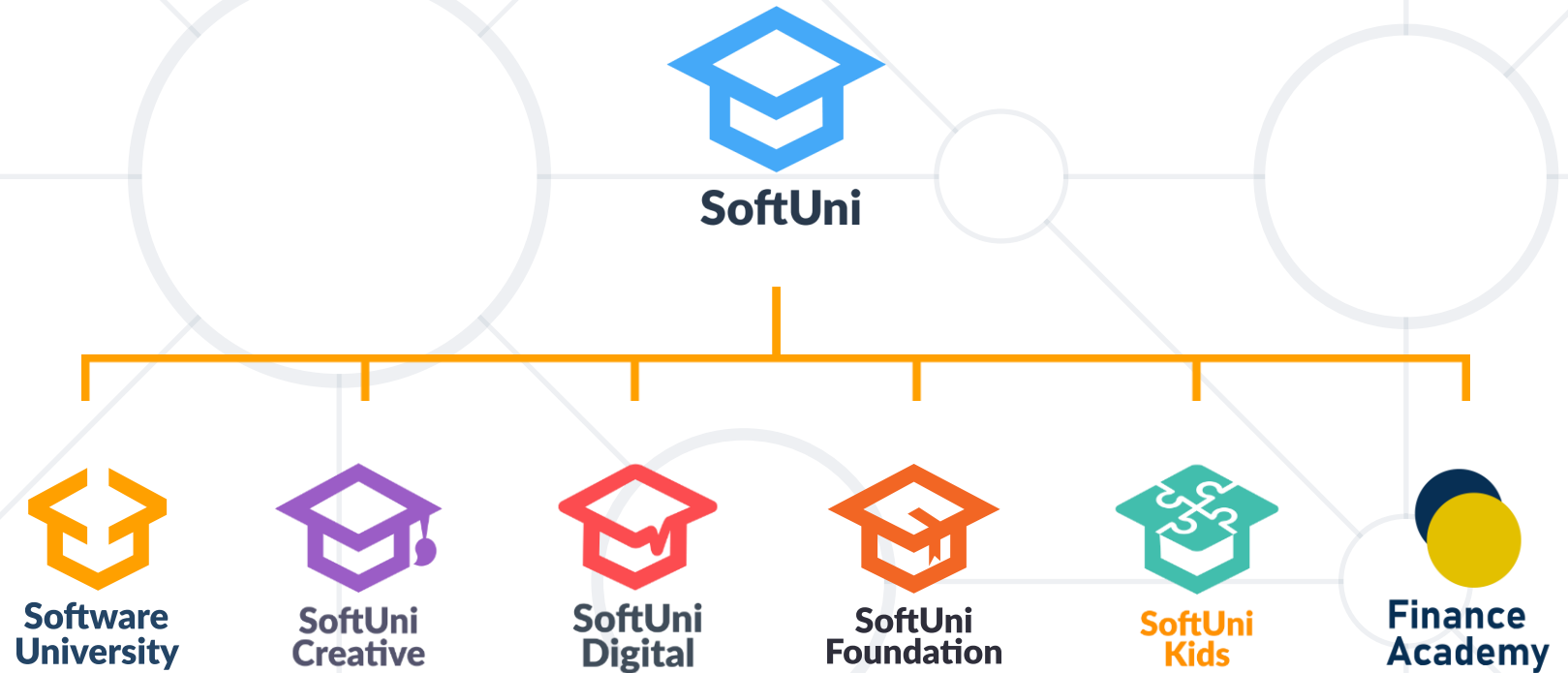
Url: No file chosen

```
10 <form enctype="multipart/form-data", method="POST">
11     {% csrf_token %}
12     {{ form.as_p }}
13     <input type="submit">
14 </form>
```

- **Validating** Django **Forms**
 - Error Messages
- Form **Class Methods**
- Formsets
- **Styling** Forms
 - Bootstrap Forms, Crispy Forms
- **Media** Files
 - Pillow



Questions?



SoftUni Diamond Partners



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg, softuni.org

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity

- Software University Forums

- forum.softuni.bg



- This course (slides, examples, demos, exercises, homework, documents, videos, and other assets) is **copyrighted content**
- Unauthorized copy, reproduction, or use is illegal
- © SoftUni – <https://softuni.org>
- © Software University – <https://softuni.bg>

