django-bleach Documentation

Mark Walker

Apr 02, 2024

CONTENTS

bleach is a Python module that takes any HTML input, and returns valid, sanitised HTML that contains only an allowed subset of HTML tags, attributes and styles. django-bleach is a Django app that makes using bleach extremely easy.

Contents:

CHAPTER

ONE

SETUP

1. Get the source from the Git repository or install it from the Python Package Index by running `pip install django-bleach`.
2. Add `django_bleach` to the `INSTALLED_APPS` setting:

   ```
   INSTALLED_APPS += (
       'django_bleach',
   )
   ```

3. Configure `django_bleach`. It comes with some sensible defaults, but you will probably want to tweak the settings for your application. See the *Settings* page for more information
3. Add a `django_bleach.models.BleachField` to a model, a `django_bleach.forms.BleachField` to a form, or use the `bleach` template filter in your templates.

CHAPTER

TWO

USAGE

2.1 In your models

django-bleach provides three ways of creating bleached output. The simplest way of including user-editable HTML content that is automatically sanitised is by using the BleachField model field:

```
# in app/models.py

from django import models
from django_bleach.models import BleachField

class Post(models.Model):

    title = models.CharField()
    content = BleachField()
```

BleachField takes the following arguments, to customise the output of bleach.

See the bleach documentation for their use:

- allowed_tags
- allowed_attributes
- allowed_protocols
- strip_tags
- strip_comments
- css_sanitizer

The following argument will be deprecated in the near future:

- allowed_styles

In addition to the bleach-specific arguments, the BleachField model field accepts all of the normal field attributes. Behind the scenes, it is a TextField, and accepts all the same arguments as TextField.

The BleachField model field sanitises its value before it is saved to the database and is marked safe so it can be immediately rendered in a template without further intervention.

In model forms, BleachField model field are represented with the BleachField form field by default.

2.2 In your forms

A BleachField form field is provided. This field sanitises HTML input from the user, and presents safe, clean HTML to your Django application and the returned value is marked safe for immediate rendering.

Usually you will want to use a BleachField model field, as opposed to the form field, but if you want, you can just use the form field. One possible use case for this set up is to force user input to be bleached, but allow administrators to add any content they like via another form (e.g. the admin site):

```
# in app/forms.py

from django import forms
from django_bleach.forms import BleachField

from app.models import Post

class PostForm(forms.ModelForm):
    class Meta:
        model = Post

        fields = ['title', 'content']

    content = BleachField()
```

The BleachField form field takes exactly the same arguments as the BleachField model field above.

2.3 In your templates

If you have a piece of content from somewhere that needs to be printed in a template, you can use the bleach filter:

```
{% load bleach_tags %}

{{ some_unsafe_content|bleach }}
```

It uses the ALLOWED_TAGS setting in your application, or optionally, bleach can pass tags:

```
{% load bleach_tags %}

{{ some_unsafe_content|bleach:"p,span" }}
```

If you have content which doesn't contain HTML, but contains links or email addresses, you can also use the bleach_linkify filter to convert content to links:

```
{% load bleach_tags %}

{{ some_safe_content|bleach_linkify }}
```

CHAPTER

THREE

SETTINGS

3.1 Configuring bleach

You can configure how `bleach` acts for your whole project using the following settings. These settings map directly to the `bleach` parameters of the same name, so see the `bleach` *documentation* for more information. Each of these have a sensible default set by `bleach`, and each of these are completely optional:

```
# Which HTML tags are allowed
BLEACH_ALLOWED_TAGS = ['p', 'b', 'i', 'u', 'em', 'strong', 'a']

# Which HTML attributes are allowed
BLEACH_ALLOWED_ATTRIBUTES = ['href', 'title', 'style']

# Which CSS properties are allowed in 'style' attributes (assuming style is
# an allowed attribute)
BLEACH_ALLOWED_STYLES = [
    'font-family', 'font-weight', 'text-decoration', 'font-variant'
]

# Which protocols (and pseudo-protocols) are allowed in 'src' attributes
# (assuming src is an allowed attribute)
BLEACH_ALLOWED_PROTOCOLS = [
    'http', 'https', 'data'
]

# Strip unknown tags if True, replace with HTML escaped characters if False
BLEACH_STRIP_TAGS = True

# Strip HTML comments, or leave them in.
BLEACH_STRIP_COMMENTS = False
```

You can override each of these for individual `BleachField` form and model fields if you need to. Simply pass in one of the following settings you want to override as a named parameter to the `BleachField`:

- `allowed_tags`
- `allowed_attributes`
- `allowed_protocols`
- `strip_tags`
- `strip_comments`

- `css_sanitizer`

The following argument will be deprecated in the near future:

- `allowed_styles`

An example, where blog posts should be allowed to contain images and headings:

```
# in app/models.py

from django import models
from django_bleach.models import BleachField

class Post(models.Model):

    title = models.CharField()
    content = BleachField(allowed_tags=[
        'p', 'b', 'i', 'u', 'em', 'strong', 'a',
        'img', 'h3', 'h4', 'h5', 'h6'])
```

3.2 Default form widget

By default, a `BleachField` will use a `django.forms.Textarea` widget. This is obviously not great for users. You can override this to use a custom widget in your project. You will probably want to use a WYSIWYG editor, or something similar:

```
BLEACH_DEFAULT_WIDGET = 'wysiwyg.widgets.WysiwygWidget'
```

I use `django-ckeditor` in my projects, but what you use is up to you.