Django Image Tools Documentation

Release 0.7.b1

Bonsai Studio

May 05, 2017

Contents

Django Image Tools

Template Images management made easy

Contents:

CHAPTER 1

Quick Start

All you have to do is

```
pip install django-image-tools-2
```

and of course, add the app in your INSTALLED_APPS as

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

Configuration

The minimum required settings you should add in your settings.py are:

```
# Folder in which you want to store your images, in this example it's the sub folder
↪'media'
MEDIA_ROOT = os.path.join(PROJECT_ROOT, 'media')
# Temporary folder for upload, in this example is the subfolder 'upload'
UPLOAD_TO = os.path.join(PROJECT_ROOT, 'media/upload')
```

The folders will be created if they do not exist.

You should also add your filters and sizes in your settings file, like this:

```
DJANGO_IMAGE_TOOLS_SIZES = {
    'thumbnail': {
        'width': 30,
        'height': 30,
        'auto': None
    },
    'very_long': {
```

```
        'width': 200,
        'height': 30,
        'auto': None
    },
    'very_tall': {
        'width': 30,
        'height': 200,
        'auto': None
    },
    'huge': {
        'width': 2000,
        'height': 2000,
        'auto': None
    },
    'auto_width': {
        'width': 0,
        'height': 20,
        'auto': 'WIDTH'
    },
    'auto_height': {
        'width': 20,
        'height': 0,
        'auto': 'HEIGHT'
    },
}

DJANGO_IMAGE_TOOLS_FILTERS = {
    'grey_scaled': {
        'filter_type': 'GREYSCALE'
    },
    'blurred': {
        'filter_type': 'GAUSSIAN_BLUR',
        'value': 5
    }
}
```

That's it, you're good to go now. Let's create a couple of models containing images!

Example models

Let's say you have a "Person" model and you want to include an Avatar, or a profile pic. Here's how you do it

```
....
from django_image_tools.models import Image

class Person(models.Model):

    first_name = models.CharField(max_length=255)
    second_name = models.CharField(max_length=255)
    ...

    picture = models.ForeignKey(Image)
```

You can also (obviously) use a ManyToMany field, for example:

```
....
from django_image_tools.models import Image

class Slideshow(models.Model):
   title = models.CharField(max_length=255)
   description = models.TextField()
   ...

   slides = models.ManyToManyField(Image)
```

That's all you need to do when dealing with the models.

Then, in your templates, you can use them like this:

```
# Displaying a thumbnail
<img src={{ person.image.get__thumbnail }} />

# Displaying a blurred thumbnail
<img src={{ person.image.get__blurred__thumbnail }} />

# Displaying the original image
<img src={{ person.image.get__original }} />

# Displaying the blurred (original) image
<img src={{ person.image.get__blurred__original }} />
```

Including thumbnails in your admin

You can include thumbnail in your admin panel

Just look at this admin.py file

```
from __future__ import absolute_import
from django.contrib import admin
from .models import Person


class PersonAdmin(admin.ModelAdmin):
    list_display = ('thumbnail', 'first_name', 'second_name')

admin.site.register(Person, PersonAdmin)
```

You will also need to tweak your model so that it uses the `thumbnail` method of the correct image. Here's an example:

```
from django.db import models
from django_image_tools.models import Image


class Person(models.Model):
    first_name = models.CharField(max_length=255)
    second_name = models.CharField(max_length=255)

    picture = models.ForeignKey(Image)

    # Add the thumbnail method and grab the image thumbnail
```

```
    def thumbnail(self):
        return self.picture.thumbnail()

    # Now you only need to tell django that this thumbnail field is safe
    thumbnail.allow_tags = True
```

Please note that in this case we used the `picture.thumbnail()` method, and not the `picture.get__thumbnail` because this particular method is designed to output the whole 'img' tag for the django admin panel.

If you have any problem, make sure you followed Django's guide to serve static and user uploaded files !

CHAPTER 2

How to use it

Philosophy

There are many tools suitable for cutting, cropping, resizing and applying filters at runtime. but we noticed that many of these lacked a sense of 'DRY-ness' as almost all of them forced you to input the size and the filter options in the template itself.

Now, try to see this from a designer point of view. As a designer, I want all of the images in this template to have a specific effect, and maybe I want a different effect for the hover version. So what do I do? Do I write down the effect's parameters for each image in the template?

And what if I wanted to change the effect after the template is ready?

The same thing applies for image sizes, what if you suddenly needed your images to be bigger or smaller?

Django Image Tools allows you to create specific filters and sizes for your images, and bind those filters to your images through their name, so you can change the filter parameters and the image size from your settings file, and those changes will automatically be applied to all of the bound images in your templates. Now your templates will remain untouched even if you decide to change a filter, or even the size of your images.

We believe this will save some time looking at the docs of an app trying to figure out which custom filter file you should include, how to input your parameters, etc...

Include it in your models

Nothing more simple. Just import the 'Image' model in your models.py file and create a Foreign Key to it. Done!

Syntax

We tried to make the syntax as readable and intuitive as possible. Filter names should be applied before size names. Size names are mandatory.

To activate django_image_tools, you need to call a `get__` attribute on the image.

For example, if you want the path for a size named 'thumbnail', you should define a new Size called 'thumbnail' on your settings file. Then, in your templates you should use:

```
<img src="image.get__thumbnail" />
```

To improve readability of your templates, we advice you to name the filters as adjectives. For example, a Gaussian Blur could be called 'blurred'.

If you want to have a thumbnail with your Gaussian blur, now the template reads:

```
<img src="image.get__blurred__thumbnail" />
```

How intuitive is that? Note that filters and sizes are separated by a double '_', as django-esque as possible.

Obviously, you cannot add filters / sizes whose name contains a double '_'.

Size & Cropping

Sizes and filters are defined in your settings file. Previous versions of this app had them defined into the admin panel, but we soon discovered how this was a real pain, especially when trying to work on multiple environments.

Here is a simple settings definition:

‘‘‘

DJANGO_IMAGE_TOOLS_SIZES = {

'thumbnail': { 'width': 30, 'height': 30, 'auto': None

}, 'very_long': {

'width': 200, 'height': 30, 'auto': None

}, 'very_tall': {

'width': 30, 'height': 200, 'auto': None

}, 'huge': {

'width': 2000, 'height': 2000, 'auto': None

}, 'auto_width': {

'width': 0, 'height': 20, 'auto': 'WIDTH'

}, 'auto_height': {

'width': 20, 'height': 0, 'auto': 'HEIGHT'

},

}

‘‘‘

All of the sizes you create will be available for the images the user will upload through their name. For example, you can have a 'thumbnail' 250x250 size, and every image you upload will have the `get__thumbnail` method that will output the path for the image with the requested size.

Having 'auto' height, for exmaple, means that the image will be resized to match the given width, and keep the original aspect ratio (This is useful for example, if you want to create a *pinterest* board). The 'auto' attribute can be either None (or just not defined), 'WIDTH', or 'HEIGHT'.

In the template, to display an image field, all you have to do is:

```
<img src='{{ some_image.get__thumbnail }}' alt_text='{{ some_image.alt_text }}' />
```

Here's a list of all the fields for each image:

- **checksum** A md5 checksum of the image. Useful for checking the integrity of the files with the database.
- **filename** The current file name. Changing this will result in renaming the actual file (useful for SEO purposes).

  Attempting to rename with the name of an existing file will throw an exception
- **subject_horizontal_position** The horizontal position of the subject. This is currently one of the list (left, 1/3, center, 2/3, right).
- **subject_vertical_position** The vertical position of the subject. This is currently one of the list (top, 1/3, center, 2/3, bottom).

  If the aspect ratio of the resized image doesn't match the original ratio, the image will be cropped around this point
- **was_upscaled** Flag that notices if the image was used somewhere with a size bigger than its own (resulting in an upscaling). Useful for letting the user know that it should replace this image with a higher-resolution version.
- **title** A title field for the image
- **caption** Caption of the image
- **alt_text** For blind people and SEO
- **credit** Credit field

Filters

Django Image tools also works great for applying filters to your images. To define a filter, just add it in your settings file. Here's an example.

‘‘‘ DJANGO_IMAGE_TOOLS_FILTERS = {

‘grey_scaled’: { ‘filter_type’: ‘GREYSCALE’

}, ‘blurred’: {

‘filter_type’: ‘GAUSSIAN_BLUR’, ‘value’: 5

}

}

For example, let's say you defined a filter named ‘blurred’ with a Gaussian Blur and you want a blurred thumbnail of your image. This should be the image tag.

Currently these are the only two filters available, we will add them if the projects becomes more popular and / or we'll need them.

```
<img src="{{ some_image.get__blurred__thumbnail }}" />
```

Note that when using a filter, the image size is mandatory. If you want to apply a filter to an image with its original size, use ‘original’ as size

```
<img src="{{ some_image.get__blurred__original }}" />
```

CHAPTER 3

How it works

The app only creates the image the first time it is requested. So don't worry about the system clogging because you have 10.0000 images and you create a new size on the fly, or having your server filled up with cached images that are never used. The images are also cached, so you should never experience a notable lag, unless you request a bunch of images of a size that was never processed. The app uses an md5 checksum to check if the image was changed. This way, it can detect even if the image was replaced by some other app, (or the user) and reprocess the various sizes of that image on request.

CHAPTER 4

More features

The images will all be available in the admin panel. You can browse and search through all of them. Sometimes, the users will upload a 'small' image (You know users right?) and then they'll complain that the image doesn't scale well, or it's too jpegged. The app will automatically flag all images for which an upscaled version was requested, by flagging them with the 'was_upscaled' flag (if you're using django_suit, the background of the row will also be displayed red). You can use

> the filter in the app to see which one were upscaled, and delete them, or replace them with a higher-res version.

The original images will never be touched, unless the user wants to rename them. The cached image folder can be changed in the system settings, through the settings variable 'DJANGO_IMAGE_TOOLS_CACHE_DIR'. This will always be a sub dir of the 'MEDIA' dir, though I might change this in the future. I strongly advice you to use the 'raw_id_fields' for the image fields, as it will allow the user to search for a previously submitted image or input a new one with a nice popup menu. This will decrease the number of duplicates. If there is a 'thumbnail' size, the app will display images of that size for the admin panel, otherwise it will fall back on the original. You can fetch the original image path by requesting 'image.get__original'.

CHAPTER 5

Testing

Often times you will find yourself having images required in your models, and testing these models can be a real pain in the donkey as you will have to create images just for that.

We want to make things simple for you, so you can import our method 'create_dummy_image' to easily create a dummy image for your tests!

```
create_dummy_image(filename=u'Test_image', title=u'Title', caption=u'Caption', alt_
↪text=u'Alt Text',
                   credit=u'Credit'):
```

This will create a new dummy entry in the database, so all you have to do is to assign it to your model's Foreign Key.

Remember to call

```
image.delete()
```

In your tearDown.

Also, django_image_tools will never delete your images, so you will have to delete them yourself. Just kidding, we made a script for that too.

```
delete_image(image)
```

So, here's a complete script.

```
def setUp(self):
    partnerImage = create_dummy_image()
    model_with_image = Model(name=u'Coca cola', image=partnerImage)
    partner.save()

def testInsert(self):
    self.assertEqual(Model.objects.all()[0].name, 'Coca cola')

def tearDown(self):
    model_with_image = Model.objects.all()[0]
```

```
    delete_image(model_with_image.image)
    model_with_image.delete()
```

Support

Django Image Tools uses Travis CI to test the integration with several versions of Python and Django. You can see the list of currently supported combinations on our Travis CI page.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search