django-imagefield Documentation

Release 0.16.1

Feinheit AG

Sep 23, 2022

Contents

Heavily based on django-versatileimagefield, but with a few important differences:

- The amount of code is kept at a minimum. django-versatileimagefield has several times as much code (without tests).
- Generating images on-demand inside rendering code is made hard on purpose. Instead, images are generated when models are saved and also by running the management command `process_imagefields`.
- django-imagefield does not depend on a fast storage or a cache to be and stay fast, at least as long as the image width and height is saved in the database. An important part of this is never determining whether a processed image exists in the hot path at all (except if you `force` it).
- django-imagefield fails early when image data is incomplete or not processable by Pillow for some reason.
- django-imagefield allows adding width, height and PPOI (primary point of interest) fields to the model by adding `auto_add_fields=True` to the field instead of boringly and verbosingly adding them yourself.

Replacing existing uses of django-versatileimagefield requires the following steps:

- `from imagefield.fields import ImageField as VersatileImageField, PPOIField`
- Specify the image sizes by either providing `ImageField(formats=...)` or adding the `IMAGEFIELD_FORMATS` setting. The latter overrides the former if given.
- Convert template code to access the new properties (e.g. `instance.image.square` instead of `instance.image.crop.200x200` when using the `IMAGEFIELD_FORMATS` setting below).
- When using django-imagefield with a PPOI, make sure that the PPOI field is also added to `ModelAdmin` or `InlineModelAdmin` fieldsets, otherwise you'll just see the image, but no PPOI picker. Contrary to django-versatileimagefield the PPOI field is editable itself, which avoids apart from other complexities a pitfall with inline form change detection.
- Add `"imagefield"` to `INSTALLED_APPS`.

If you used e.g. `instance.image.crop.200x200` and `instance.image.thumbnail.800x500` before, you should add the following setting:

```
IMAGEFIELD_FORMATS = {
    # image field path, lowercase
    'yourapp.yourmodel.image': {
        'square': ['default', ('crop', (200, 200))],
        'full': ['default', ('thumbnail', (800, 500))],

        # The 'full' spec is equivalent to the following format
        # specification in terms of image file produced (the
        # resulting file name is different though):
        # 'full': [
        #     'autorotate', 'process_jpeg', 'process_png',
        #     'process_gif', 'autorotate',
        #     ('thumbnail', (800, 500)),
        # ],
        # Note that the exact list of default processors may
        # change in the future.
    },
}
```

After running `./manage.py process_imagefields` once you can now use use `instance.image.square` and `instance.image.thumbnail` in templates instead. Note that the properties on the `image` file do by design not check whether thumbs exist.

CHAPTER 1

Image processors

django-imagefield uses an image processing pipeline modelled after Django's middleware.

The following processors are available out of the box:

- `autorotate`: Autorotates an image by reading the EXIF data.
- `process_jpeg`: Converts non-RGB images to RGB, activates progressive encoding and sets quality to a higher value of 90.
- `process_png`: Converts PNG images with palette to RGBA.
- `process_gif`: Preserves transparency and palette data in resized images.
- `preserve_icc_profile`: As the name says.
- `thumbnail`: Resizes images to not exceed a bounding box.
- `crop`: Crops an image to the given dimensions, also takes the PPOI (primary point of interest) information into account if provided.
- `default`: The combination of `autorotate`, `process_jpeg`, `process_gif`, `process_png` and `preserve_icc_profile`. Additional default processors may be added in the future. It is recommended to use `default` instead of adding the processors one-by-one.

Processors can be specified either using their name alone, or if they take arguments, using a tuple where the first entry is the processors' name and the rest are positional arguments.

You can easily register your own processors or even override built-in processors if you want to:

```
from imagefield.processing import register

# You could also write a class with a __call__ method, but I really
# like the simplicity of functions.

@register
def my_processor(get_image, ...):
    def processor(image, context):
        # read some information from the image...
```

(continues on next page)

(continued from previous page)

```
        # or maybe modify it, but it's mostly recommended to modify
        # the image after calling get_image

        image = get_image(image, context)

        # modify the image, and return it...
        modified_image = ...
        # maybe modify the context...
        return modified_image
    return processor
```

The processor's name is taken directly from the registered object.

An example processor which converts images to grayscale would look as follows:

```
from PIL import ImageOps
from imagefield.processing import register

@register
def grayscale(get_image):
    def processor(image, context):
        image = get_image(image, context)
        return ImageOps.grayscale(image)
    return processor
```

Now include `"grayscale"` in the processing spec for the image where you want to use it.

CHAPTER 2

The processing context

The `context` is a namespace with the following attributes (feel free to add your own):

- `processors`: The list of processors.
- `name`: The name of the resulting image relative to its storages' root.
- `extension`: The extension of the source and target.
- `ppoi`: The primary point of interest as a list of two floats between 0 and 1.
- `save_kwargs`: A dictionary of keyword arguments to pass to `PIL.Image.save`.

The `ppoi`, `extension`, `processors` and `name` attributes cannot be modified when running processors anymore. Under some circumstances `extension` and `name` will not even be there.

If you want to modify the extension or file type, or create a different processing pipeline depending on facts not known when configuring settings you can use a callable instead of the list of processors. The callable will receive the fieldfile and the context instance and must at least set the context's `processors` attribute to something sensible. Just as an example here's an image field which always returns JPEG thumbnails:

```
from imagefield.processing import register

@register
def force_jpeg(get_image):
    def processor(image, context):
        image = get_image(image, context)
        context.save_kwargs["format"] = "JPEG"
        context.save_kwargs["quality"] = 90
        return image
    return processor

def jpeg_processor_spec(fieldfile, context):
    context.extension = ".jpg"
    context.processors = [
        "force_jpeg",
        "autorotate",
```

(continues on next page)

(continued from previous page)

```
        ("thumbnail", (200, 200)),
    ]

class Model(...):
    image = ImageField(..., formats={"thumb": jpeg_processor_spec})
```

Of course you can also access the model instance through the field file by way of its `fieldfile.instance` attribute and use those informations to customize the pipeline.

CHAPTER 3

Development

django-imagefield uses flake8 and black to keep the code clean and formatted. Run both using tox:

```
tox -e style
```

The easiest way to build the documentation and run the test suite is also by using tox:

```
tox -e docs  # Open docs/build/html/index.html
tox -e tests
```

CHAPTER 4

Change log

4.1 Next version

- Added a force-WEBP spec.
- Added Django 4.1.

4.2 0.16 (2022-05-04)

- Raised the minimum Pillow version to 9.0.
- Avoided a deprecation warning by using the `PIL.Image.Resampling` enum.

4.3 0.15 (2022-03-07)

- Dropped support for Python < 3.8, Django < 3.2.
- Added a simplistic workaround for `IOError` exceptions which still plague Pillow when saving some JPEG files.

4.4 0.14 (2021-12-23)

- Renamed the main branch to `main`.
- Reformatted the frontend code using prettier and checked it using ESLint.
- Fixed a crash which happened when the PPOI field contained an invalid value.
- Added saving files in their original format to `verified`. Previously, some images were accepted because they can be loaded but they could not be saved later.

- Added Python 3.10, Django 4.0 to the CI.
- Dropped support for Python 2.7, Django 1.11.
- Added a warning when using `ImageField(null=True)`.
- Started using pre-commit.

4.5 0.13 (2021-02-03)

- Started using `ImageOps.exif_transpose` introduced in Pillow 6.0 to autorotate images.
- Allowed overriding the preview image for the form field by adding a `"preview"` spec to the field.
- Added descriptions when raising `AttributeError` exceptions.
- Fixed the alignment of file uploads when imagefields with PPOI widgets are used within a fieldbox in the admin interface.
- Do not accept keys or attributes starting with underscores in the versatile image proxy.
- Disallowed image format names starting with an underscore.
- Added `IMAGEFIELD_VALIDATE_ON_SAVE` to skip image validation when using model-level methods. Only use this when you are absolutely 100% sure that the images you are adding can be processed by Pillow.
- Switched from Travis CI to GitHub actions.

4.6 0.12 (2020-07-24)

- **BACKWARDS INCOMPATIBLE**: Consolidated cache key generation in the versatile image proxy and the admin widget code. Already cached values will be checked again. Also, the cache timeout has been changed from infinite (in case of the versatile image proxy) and 30 days (in case of the admin widget) to a random value between 170 and 190 days. This can be overridden by specifying the timeout as `IMAGEFIELD_CACHE_TIMEOUT`. The setting may either be a value or a callable.
- Fixed a pickle/unpickle crash.
- Closed image files in more places to avoid resource warnings.
- Dropped Django 1.8 from the build matrix. Supporting it in the testsuite became annoying.
- Added verification of images even when not using forms.
- Ensured that configured fallbacks are also processed by `process_imagefields`.
- Silenced more warnings when running the testsuite and generally improved test coverage.
- Avoided setting the image field files' value too early when using fallbacks.
- Added a new `process_png` processor which converts PNGs using palettes to RGBA. This avoids ugly artefacts when resizing images.

4.7 0.11 (2020-01-27)

- Changed the fallback facility to a keyword argument to the `ImageField` instance.
- Changed processing context creation to assign the `name` field earlier.

4.8 0.10 (2020-01-24)

- Added an experimental fallback facility for optional image fields.
- Allowed processor specs to return another processor spec in turn. This allows layering processor specs.
- Changed the image field to set image file's extensions depending on their image type. For example, a GIF uploaded as `example.png` will automatically be saved as `example.gif`.
- Improved test coverage a bit.

4.9 0.9 (2020-01-22)

- Fixed crashes because of image fields with `None` values.
- Fixed a case where an unsupported image was not detected early enough.
- Added a `IMAGEFIELD_SILENTFAILURE` setting for silent failure when processing images crashes. The default value of this setting is obviously `False`. This is mostly useful when adding `django-imagefield` to a project which already has images (which may not be processible by Pillow).
- Fixed the image verification to accept CMYK images again.
- Added Django 3.0 to the test matrix.
- Removed Python 3.4 from the test matrix.
- Ensure that `icc_profile` isn't passed if it is falsy. The WebP encoder didn't like `icc_profile=None`.
- Stopped including image fields of swapped models in `IMAGEFIELDS`.
- Replaced `ugettext*` with `gettext*`.
- Added an experimental websafe processor spec.

4.10 0.8 (2019-06-21)

- **BACKWARDS INCOMPATIBLE**: Changed processing to pass additional processors' arguments as positional arguments instead of as a single list. This change only affects custom processors, no changes are necessary for users of the library, except if for example you passed arguments to processors such as `default`, `autorotate` etc.
- Fixed a test to assume less about the error message for corrupt images.
- Localize the corrupt image validation errors.
- Stopped calling the storage's `delete()` method for non-existing images.
- Made the field resilient against NULL values from the database.

4.11 0.7 (2018-10-18)

- Made error reporting in `process_imagefields` include more info.
- Made image field validation catch errors while determining the image dimension too.

- Fixed a problem where older versions of Django didn't allow specifying the chunk size for iterating over querysets.
- Modified django-imagefield's internals to allow changing the type and extension of generated images by way of dynamically specifying the processing pipeline.
- Changed the API of the `get_image` callable in processors to only return the image without the context (since the context is mutable and available already).

4.12 0.6 (2018-09-13)

- Fixed a crash where unpickling image fields would fail.
- Changed `process_imagefields` to skip exclude model instances with an empty image field.
- Changed the `thumbnail` processor to not upscale images.
- Made `process_imagefields` not load the whole queryset at once to avoid massive slowdowns while determining the width and height of images (if those fields aren't filled in yet).
- Added housekeeping options to `process_imagefields`. The only method implemented right now is `--housekeep blank-on-failure` which empties image fields where processing fails.
- Changed `process_imagefields` to process items in a deterministic order.
- Clarified the processors spec documentation a bit and added an example how to write a processor of your own.

4.13 0.5 (2018-08-15)

- Dropped support for using image fields without associated height and width fields, because it is almost (?) always a really bad idea performance-wise.
- Fixed a bug where processed image names on Python 2 were different than those generated using Python 3. This bug affects only installations still using Python 2. Rerun `./manage.py process_imagefields --all` after upgrading.

4.14 0.4 (2018-08-13)

- Added compatibility with Django 1.8 for prehistoric projects.
- Polished tests and docs a bit.

4.15 0.3 (2018-05-29)

- **BACKWARDS INCOMPATIBLE**: Changed the filename generation method to preserve the filename part of the original file for SEO purposes etc. You should run `./manage.py process_imagefields --all`, and optionally empty the `__processed__` folder before doing that if you do not want to keep old images around.
- Improved progress reporting in `process_imagefields`.
- Added a call to `instance.save()` in `process_imagefields` so that width and height fields are saved (if any).

- Added `accept="image/*"` attribute to the file upload widget.
- Replaced the full image in the admin widget with an ad-hoc thumbnail.
- Fixed a bug where blank imagefields would not work correctly in the administration interface.
- Switched the preferred quote to `"` and started using black to automatically format Python code.

4.16 0.2 (2018-03-28)

- Rename management command to `process_imagefields`, and add `--all` option to process all imagefields.
- Fixed a bug where not all image fields from base classes were picked up for processing by `process_imagefields`.
- Added the `IMAGEFIELD_AUTOGENERATE` setting, which can be set to a list of image fields (in `app.model.field` notation, lowercased) to only activate automatic processing of images upon model creation and update for a few specific fields, or to `False` to disable this functionality for all fields.
- Added system checks which warn when `width_field` and `height_field` are not used.
- Changed `process_imagefields` to process image fields in alphabetic order. Also, made cosmetic changes to the progress output.
- Added a test which verifies that generating processed image URLs is not slowed down by potentially slow storages (e.g. cloud storage)
- Fixed the PPOI JavaScript to not crash when some imagefields have no corresponding PPOI input.

4.17 0.1 (2018-03-27)

- First release that should be fit for public consumption.