django-elements Documentation

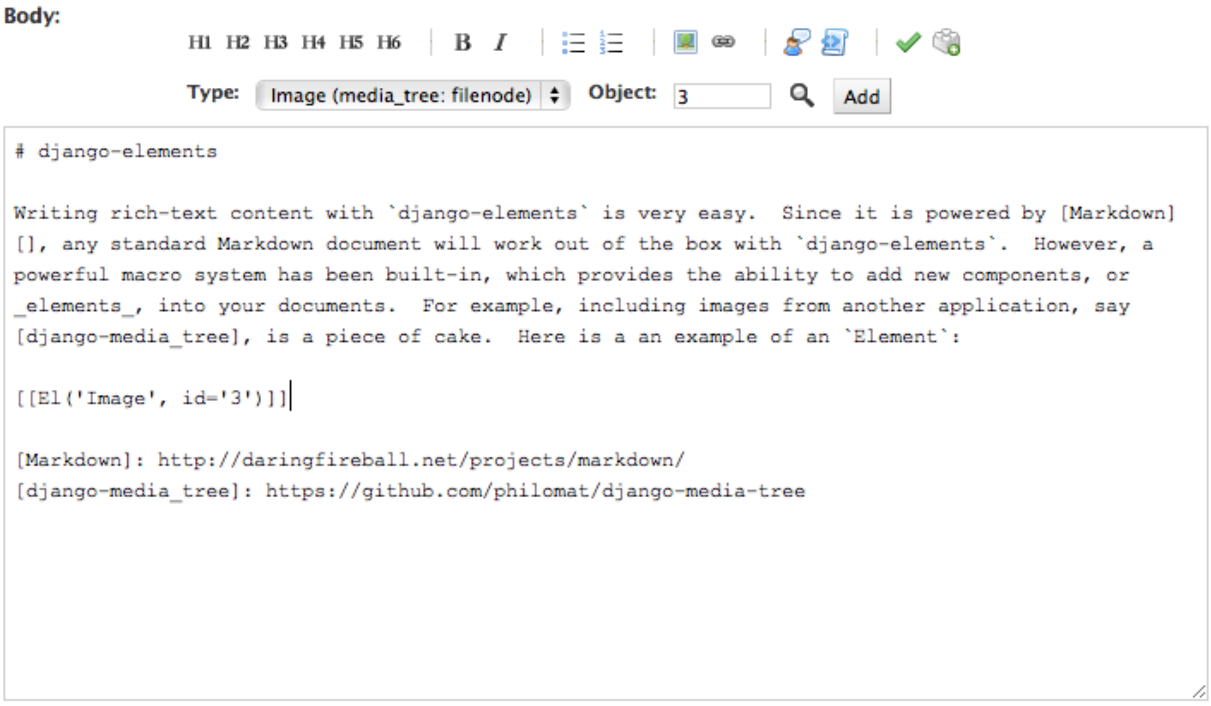
Release 0.1.0

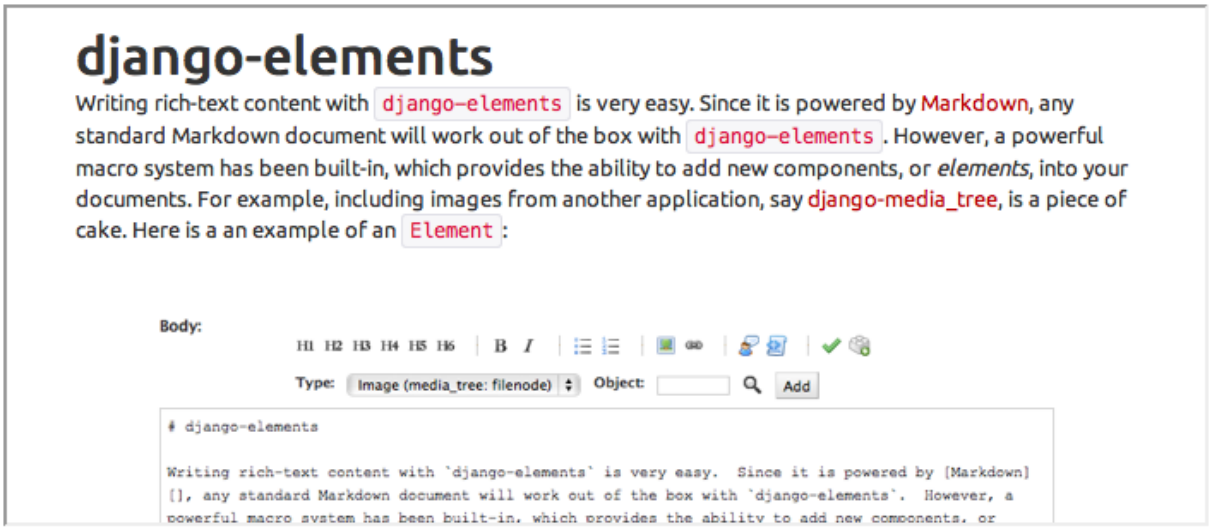
Weston Nielson

Aug 22, 2017

Contents

`django-elements` is a reusable application for Django projects that seeks to provide a way to make publishing rich-text content on the web easy for the end-user. Instead of requiring the user to write in HTML (or worse, use a WYSIWYG editor), `django-elements` uses a superset of the human-friendly Markdown syntax. The standard Markdown syntax has been extended with a powerful macro system (similar to the macro syntax used in Trac) via markdown-macros. Since a picture is worth 1000 words, here is an example of a document written with `django-elements`:





Contents:

CHAPTER 1

Installation

The first step is to add `elements` to your `INSTALLED_APPS` setting inside `settings.py`:

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

Then run `syncdb`:

```
python manange.py syncdb
```

That's it for the required steps.

Optional Steps

If you want to make editing in the admin easier, then you'll probably want to install `django-markitup`:

```
pip install django-markitup==tip
```

`django-elements` ships with an extended Markdown syntax set for the javascript-based `MarkItUp` editor provided in `django-markitup`. To enable it, add the following to your `settings.py`:

```
MARKITUP_FILTER = ('elements.markup.convert', {})
MARKITUP_SET = 'elements/markitup/markdown'
```

Then run `collectstatic`:

```
python manage.py collectstatic
```

CHAPTER 2

Configuration

Admin Integration

If you installed django-markitup, and you'd like to have the `MarkItUp` editor enabled for text areas in the admin, then you need to create special templates. For example, let's say you'd like to add the editor to Django's `flatpages` application. In your templates directory, create the following file: `admin/flatpages/flatpage/change_form.html`, and add the following content:

```
{% extends "admin/change_form.html" %}

{% load markitup_tags %}

{% block extrahead %}
  {{ block.super }}
  {% markitup_media %}
{% endblock %}


{% block content %}
  {{ block.super }}
  {% markitup_editor "id_content" %}
{% endblock %}
```

You can repeat this for any other text area you'd like to add the editor to.

Settings

There are a few settings that let you control the behavior of `django-elements`.

ELEMENTS_MARKDOWN_EXT

The `ELEMENTS_MARKDOWN_EXT` directive allows you to define extra python-markdown extensions to use in the Markdown rendering. The default is:

```
ELEMENTS_MARKDOWN_EXT = (
    'toc',
    'tables',
    'abbr',
    'footnotes',
    'def_list',
    'headerid',
    'meta',
    'codehilite'
)
```

For a complete list of available extenstions, see this page. Additionally, if you'd like to take advantage of the `codehilite` extension, you'll need to install pygments:

```
pip install pygments
```

ELEMENTS_MARKDOWN_EXT_CONFIGS

This directive allows you to pass extra extension-specific config options to to Markdown processor. The default is:

```
ELEMENTS_MARKDOWN_EXT_CONFIGS = {}
```

CHAPTER 3

Usage

To convert a Markdown document with `Elements` contained within, you need to first load the template tags:

```
{% load elements_markup %}
```

Currently there is both a tag and a filter that you can use to render the Markdown content as HTML. The more powerful option is the template tag and its usage is best explained with an example. Let's pretend we've created a `FlatPage` who's `content` contains text written in Markdown with elements contained within. We can render the page as HTML, assuming the `FlatPage` object is referenced as `flatpage`, like so:

```
{% markup_elements flatpage content %}
```

This template tag is more powerful than the filter described below, because each element contained within this `FlatPage` is "aware" of the context in which is is being rendered. Again, this concept is more easily explained with and example.

Let's consider a very simple application, which we'll call `media`, that has a very simple model:

```
from django.db import models

class MediaItem(models.Model):
    title = models.CharField(max_length=255)
    file = models.FileField()
    caption = models.TextField(blank=True)
```

Now let's also pretend that we've created an `ElementType` for this model, titled "Image". This means that we can now add an "Image" into a `FlatPage` like so, assuming we've uploaded an image with `pk=1`:

```
Here is some flatpage content.  Let's go ahead an insert an image:

[[El('Image', id=1)]]
```

To recap what we've got currently:

- A `FlatPage` with `pk=2` and `content` shown above

- An `ElementType` titled "Image"
- An `MediaItem` with `pk=1` from the `media` application

Now we can control how the `Image` above is actually rendered into HTML by defining various templates. Here is the order in which templates will be searched:

- `elements/flatpages/flatpage/2/media_mediaitem_1.html`
- `elements/flatpages/flatpage/2/media_mediaitem-image.html`
- `elements/flatpages/flatpage/2/media_mediaitem.html`
- `elements/flatpages/flatpage/2/media-image.html`
- `elements/flatpages/flatpage/2/media.html`
- `elements/flatpages/flatpage/2/default.html`
- `elements/flatpages/flatpage/media_mediaitem_1.html`
- `elements/flatpages/flatpage/media_mediaitem-image.html`
- `elements/flatpages/flatpage/media_mediaitem.html`
- `elements/flatpages/flatpage/media-image.html`
- `elements/flatpages/flatpage/media.html`
- `elements/flatpages/flatpage/default.html`
- `elements/flatpages/media_mediaitem_1.html`
- `elements/flatpages/media_mediaitem-image.html`
- `elements/flatpages/media_mediaitem.html`
- `elements/flatpages/media-image.html`
- `elements/flatpages/media.html`
- `elements/media_mediaitem-image.html`
- `elements/media_mediaitem.html`
- `elements/default.html`

This means we can define a template that will dictate how this "Image" element will render for this `FlatPage` (and only this particular `FlatPage`) via the `elements/flatpages/flatpage/2/media_mediaitem_1.html`, or we can simply define a more generic template that will define how to render any "Image" for any `FlatPage` via the `elements/flatpages/flatpage/media_mediaitem-image.html`.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search