django-wysiwyg Documentation

Release 0.5.1

Daniel Greenfeld, Chris Adams, and contributors

Jun 21, 2017

Contents

Contents:

CHAPTER 1

Installation

in settings.py:

```
INSTALLED_APPS = (
    ...
    'django_wysiwyg'
    ...
)
```

Other settings:

```
DJANGO_WYSIWYG_FLAVOR = 'yui'        # Default
DJANGO_WYSIWYG_FLAVOR = 'ckeditor'   # Requires you to also place the ckeditor files␣
↪here:
DJANGO_WYSIWYG_MEDIA_URL = STATIC_URL + "ckeditor/"
```

The following editors are supported out of the box:

- *ckeditor* - The CKEditor, formally known as FCKEditor.
- *redactor* - The Redactor editor (requires a license).
- *tinymce* - The TinyMCE editor, in simple mode.
- *tinymce_advanced* - The TinyMCE editor with many more toolbar buttons.
- *yui* - The YAHOO editor (the default)>
- *yui_advanced* - The YAHOO editor with more toolbar buttons.

Media sources

When you use one of the editors, you need to make sure that the editor distributables are also located in your project. By default *django-wysiwyg* looks for a *STATIC_URL/flavor* folder. You can also install django-ckeditor or django-tinymce to have the CKEditor and TinyMCE distributables respectively. *django-wysiwyg* will automatically find their sources if they are mentioned in the `INSTALLED_APPS`.

It's also possible to add new editors, see *extending django-wysiwyg*

CHAPTER 2

Examples

Simple template example:

```
{% load wysiwyg %}

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">

<html lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>basic_test</title>
    {% wysiwyg_setup  %}
</head>
<body>
    <textarea id="my_text">This is some text. Please edit it</textarea>
    {% wysiwyg_editor "my_text" %}
</body>
</html>
```

CHAPTER 3

Extending django-wysiwyg

The django-wysiwyg module can easily by extended with new editor types.

The editor switching in implemented by selecting templates based on the `DJANGO_WYSIWYG_FLAVOR` setting. Adding an extra editor simply requires these templates to be added:

- django_wysiwyg/*editorname*/includes.html
- django_wysiwyg/*editorname*/editor_instance.html

includes.html

The includes file will be added to the top of the page, to provide all required scripts. It is loaded by the `{% wysiwyg_setup %}` code. The template could contain something like:

```
<script type="text/javascript" src="{{ DJANGO_WYSIWYG_MEDIA_URL }}editor.js"></script>
<script type="text/javascript" src="{{ DJANGO_WYSIWYG_MEDIA_URL }}sample.css"></
↪script>
```

Secondly, the file has to provide a few JavaScript functions, to implement the *JavaScript API* This is used for Ajax environments, or interfaces which use a lot of DOM manipulation. The required API functions have the following structure:

```
var django_wysiwyg_editor_configs = [];   // allow custom settings per editor ID{%␣
↪block django_wysiwyg_editor_config %}
var django_wysiwyg_editor_config = {};
{% endblock %}


var django_wysiwyg =
{
    editors: {},  // where the editor object can be stored.

    is_loaded: function()
```

```
    {
        // ... some test to see if the scripts were loaded properly.
        return window.MY_EDITOR != null;
    },

    enable: function(editor_name, field_name, config)
    {
        if( !config ) {
            config = django_wysiwyg_editor_configs[field_id] || django_wysiwyg_editor_
↪config;
        }

        if( !this.editors[editor_name] ) {
            this.editors[editor_name] = // ... enable the editor for the field name
        }
    },

    disable: function(editor_name)
    {
        var editor = this.editors[editor_name];
        if( editor ) {
            editor.the_destroy_function();   // ... call the destroy function
            this.editors[editor_name] = null;
        }
    }
}
```

The `enable()` function should be able to deal with attempts to enable the editor twice. It should also store the created WYSIWYG editor instance in the `this.editors[editor_name]` variable. That allows the caller to access the object when it needs to.

For more inspiration, you can inspect the files in the `django_wysiwyg` template directory.

editor_instance.html

The editor-instance template is used to instantiate a single editor statically. It is loaded by the `{% wysiwyg_editor fieldname %}` line in the template. The contents of the template can look something like:

```
<script type="text/javascript">
    (function(){
        var config = {{ config }};
        django_wysiwyg.enable('{{ editor_name }}', '{{ field_id }}', config);
    })();
</script>
```

In most cases, this should be enough to instantiate the editor for a specific field.

Extending existing templates

Some templates also provide blocks, that allow them to be extended. For example, the *yui_advanced* editor, is implemented by extending the *yui* templates.

CHAPTER 4

JavaScript API

To support the needs of more advanced web interfaces, django-wysiwyg provides a JavaScript API. This can be used to enable or disable WYSIWYG editors on demand, for example, because the element was inserted dynamically though DOM manipulation.

Every editor type provides the following JavaScript functions:

- `django_wysiwyg.is_loaded()` - Whether the external scripts for the editor are loaded. This can be useful for offline-mode, to refrain from updating textarea fields.
- `django_wysiwyg.enable(editor_name, field_name)` - Enable an editor for a given field.
- `django_wysiwyg.disable(editor_name)` - Disable the editor.
- `django_wysiwyg.editors[editor_name]` - Access to the editor object (e.g. the CKEditor, or YUI Editor object).

Moving WYSIWYG editors in the DOM

Most WYSIWYG editors are not built for being moved around in the DOM, and keep referring to old elements in the event handlers. A solution to this problem, is disabling the editor first (which removes all WYSIWYG nodes), and enable it again after moving the containr.

CHAPTER 5

Utility functions

Django-wysiwyg provides a few utility functions to deal with HTML from WYSIWYG editors. Example:

```
from django_wysiwyg.utils import clean_html, sanitize_html

print clean_html("<b><i>test</b></i>")
print sanitize_html("<b><script>alert(1)</script></b>")
```

CHAPTER 6

Indices and tables

- genindex
- modindex
- search