Django-Front Documentation

Release 0.5.6

Marco Bonetti

Apr 24, 2017

Contents

Django-front is a front-end editing application for Django

Fig. 1: Using django-front to edit a placholder with three of the supported editors.

CHAPTER 1

Contents:

Introduction

Django-front is a front-end editing application: placeholders can be defined in Django templates, which can then be edited on the front-end (in place) by authorized users.

Features

- Edit html content directly on the front-end of your site.
- Version history: jump back to a previous version of a content block at any time.
- Content blocks are persisted in the database, and in Django's cache (read: zero database queries)
- Content scope and inheritance: content blocks (placeholders) can be defined globally (i.e. edited once, displayed on every page), or on a single url, for a single language, ...
- Built-in support for several content editor plugins (WYSIWYG, html, Markdown, ...), adding a new editor is very simple.

Installation

Requirements

Python requirements are automatically installed when you install via pip.

- Django-front supports Django 1.8 through 1.11
- django-classy-tags
- Python 2.7+ or Python 3.5+
- jQuery is required in your template

Installing django-front

- `pip install django-front`
- Add `front` to your `INSTALLED_APPS`
- Add a line to your `urls.py`:

```
urlpatterns += [
    url(r'^front-edit/', include('front.urls')),
]
```

- `python manage.py migrate` (or syncdb if that's your dope)
- Note: the `django.core.context_processors.request` context processor must be enabled in your `TEMPLATE_CONTEXT_PROCESSORS` setting.

Testing

- `pip install tox && tox`

Setting it up

Template layout

We assume that your site uses a basic template hierarchy, having a base template and multiple "content" templates inheriting from the base one.

To set up *django-front*, you will need to include a few lines in your base template, then add content placeholders in the child templates.

Your base template...

First, load the *front_tags* module at the top of your base template:

```
{% load front_tags %}
```

Then include jQuery, followed by front-editing scripts somewhere towards the end of your `<body>`, e.g.:

```
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js"></script>
{% front_edit_scripts %}
```

Note: the Redactor editor needs a fairly recent version of jQuery (1.8+), but WYMeditor will need an older one (<= 1.7). Adapt as needed.

Defining placeholders in your child templates...

Once the `front_edit_scripts` scripts are injected (they are only rendered to users who can actually edit the content), you can start adding placeholders to your templates.

First load the tag library:

```
{% load front_tags %}
```

Then define a placeholder:

```
{% front_edit "placeholder_name" request.path request.LANGUAGE_CODE %}
    <p>Default when empty</p>
{% end_front_edit  %}
```

When no placeholder content is defined for this placeholder, the default content is displayed instead.

Placeholder scope

Any variable passed after the placeholder name will be evaluated. The full identifier (i.e. name and variables) will be hashed and will define the main identifier for this placeholder.

The scope (visibility) of the rendered content block is defined by the variable names used in the block definition: the content block in the previous example will be rendered only on the page at the current URL, and the current language.

The following example, on the other hand, would be rendered on every page using the template having this tag, regardless of the language and the URL:

```
{% front_edit "look ma, Im global!" %}
    <p>Default when empty</p>
{% end_front_edit  %}
```

Choosing an editor

The default editor just uses a plain `<textarea>` to edit the html code, not too fancy.

You can pass an argument to the `front_edit_scripts` tag added in the base template, to specify the editor to use.

Ace editor

Ace is an embeddable code editor written in JavaScript, it is used by GitHub and the Khan Academy, among others.

To use the Ace editor:

```
{% front_edit_scripts editor="ace" %}
```

Ace is loaded from a CDN, no extra installation is required.

NOTE: if the CDN solution doesn't work for you, download Ace and serve it locally and use the *ace-local* plugin:

```
<script src="{{STATIC_URL}}ace/src-min-noconflict/ace.js"></script>
{% front_edit_scripts editor="ace-local" %}
```

WYMeditor

WYMeditor is a web-based WYSIWYM (What You See Is What You Mean) XHTML editor

You'll have to install `django-wymeditor`: `pip install django-wymeditor`, then add `wymeditor` to your `INSTALLED_APPS`, then:

```
{% front_edit_scripts editor="wymeditor" %}
```

Redactor

Redactor is a commercial WYSIWYG html editor.

To use the Redactor editor:

```
{% front_edit_scripts editor="redactor" %}
```

Redactor being closed-source, it is not distributed with django-front: you'll have to download it and install it in your project:

1. Copy `redactor9xx` into a directory being served as static file
2. In the `head` of your master template, include the Redactor stylesheet:

   ```
   <link rel="stylesheet" type="text/css" href="{{STATIC_URL}}redactor9xx/redactor/
   ↪redactor.css">
   ```

3. In your master template, after the jQuery inclusion and before your `{% front_edit_scripts editor="redactor" %}` tag, include the Redactor JavaSript file:

   ```
   <script type="text/javascript" src="{{STATIC_URL}}redactor9xx/redactor/redactor.
   ↪min.js"></script>
   ```

(Replace `redactor9xx` with the build number you've downloaded)

CKEditor

CKEditor is a ready-for-use HTML text editor designed to simplify web content creation.

To use CKEditor editor, make sure that the `ckeditor.js` script is loaded in your base template, (or load it from a CDN: `<script src="//cdn.ckeditor.com/4.4.7/standard/ckeditor.js"></script>`), then:

```
{% front_edit_scripts editor="ckeditor" %}
```

EpicEditor

EpicEditor is an embeddable JavaScript Markdown editor.

To use EpicEditor:

```
{% front_edit_scripts editor="epiceditor" %}
```

The EpicEditor scripts are served directly from django-front's static folders, no need to include anything else in your base template.

Froala

Froala is a commercial WYSIWYG html editor. It is free to use for personal and non-profit projects.

Froala being closed-source, it is not distributed with django-front: you'll have to download and install it in your project. Alternatively it can be served from a CDN.

In your <head>:

```
<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/font-awesome/4.2.0/css/font-
↪awesome.min.css">
<link rel="stylesheet" href="//cdnjs.cloudflare.com/ajax/libs/froala-editor/1.2.6/css/
↪froala_editor.min.css">
<link rel="stylesheet" href="//cdnjs.cloudflare.com/ajax/libs/froala-editor/1.2.6/css/
↪themes/gray.min.css">
```

At the end of your <body>:

```
<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script src="//cdnjs.cloudflare.com/ajax/libs/froala-editor/1.2.6/js/froala_editor.
↪min.js"></script>
{% front_edit_scripts editor="froala" %}
```

Froala accepts options that can be passed to the editor via the DJANGO_FRONT_EDITOR_OPTIONS settings (see the next section).

Medium Editor

Medium Editor is a Medium.com WYSIWYG editor clone. Uses contenteditable API to implement a rich text solution.

In your <head>:

```
<link rel="stylesheet" href="//cdn.jsdelivr.net/medium-editor/latest/css/medium-
↪editor.min.css">
<link rel="stylesheet" href="//cdn.jsdelivr.net/medium-editor/latest/css/themes/
↪beagle.min.css">
```

At the end of your <body>:

```
<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script src="//cdn.jsdelivr.net/medium-editor/latest/js/medium-editor.min.js"></
↪script>
{% front_edit_scripts editor="medium" %}
```

The Medium Editor accepts setting options that can be passed to the editor via the DJANGO_FRONT_EDITOR_OPTIONS settings (see the next section).

Passing arguments to the editor

You can pass extra initialization arguments to some of the editors, to e.g. handle file uploads or load plugins. See: DJANGO_FRONT_EDITOR_OPTIONS under *Settings*

Add your own editor

To add support for a new editor type (say "foo"):

1. Add ['foo',] to DJANGO_FRONT_ALLOWED_EDITORS in your settings. See: *Settings*
2. Add a /static/front/js/front-edit.foo.js file, you'll need to provide the following function prototype (here as an example for the default editor, see more examples in static/front/js)

```
(function(jQuery){
    window.front_edit_plugin = {

        target: null,

        // Returns the html that will contain the editor
        get_container_html: function(element_id, front_edit_options) {
            return '<textarea class="front-edit-container" id="edit-'+ element_id
↪+'"></textarea>';
        },

        // initializes the editor on the target element, with the given html code
        set_html: function(target, html, front_edit_options) {
            this.target = target;
            this.target.find('.front-edit-container').html(html);
        },

        // returns the edited html code
        get_html: function(front_edit_options) {
            return this.target.find('.front-edit-container').val();
        },

        // destroy the editor
        destroy_editor: function() {
            self.target = null;
        }
    };
})(jQuery);
```

3. Maybe submit a pull request?

Using django-front

Editing content

1. As an authorized (and authenticated) user, hover your mouse pointer over a placeholder, it'll get a blue border.
2. Double-click to start editing.
3. After you're done editing:
   (a) Click "save" to commit the changes to the database and get back to the default view, or
   (b) Click "cancel" to revert to the previously saved version, or
   (c) Click "history" to fetch the change history from the back-end. The button will turn into a select-box, displaying all the saved timestamps. Selecting a previous version will load the content from that version into the editor.

Security considerations

By default, only authenticated users having the staff flag can edit placeholder content.

You can specify who can edit content in the settings: see `DJANGO_FRONT_PERMISSION` under *Settings*.

Django-front will render its JavaScript object only to authenticated users with editing permissions.

Performance

- The first time a placeholder tag is rendered, its content fetched from the database and stored in Django's cache. Successive hits on that placeholder will get the content from the cache.
- Each time a placeholder is saved, its cache key is invalidated. A `PlaceholderHistory` object is also saved (if the content was changed).

Warning: To avoid hitting the database for each placeholder it is critical to use a proper cache back-end, such as Memcached.

Caveats

For users allowed to edit the content, `django-front` will wrap the rendered placeholder in a div, to mark it up as editable, e.g.:

```
<body>
    <div class="editable" id="269cd2452ec9b17252d19eaa20719eedebec86a9">
        <h1>aaa</h1>
        <p>Lorem ipsum dolor sit amet, ...</p>
    </div>
</body>
```

(whitespace added for emphasis)

To non-authenticated users, the same placeholder will be rendered without the wrapping div:

```
<body>
    <h1>aaa</h1>
    <p>Lorem ipsum dolor sit amet, ...</p>
</body>
```

As a consequence, CSS child selectors (e.g. `body > h1`) will behave differently to authenticated users than non-authenticated ones. As a workaround, just keep in mind to extend the child selector to cover this case, e.g.:

```
body > h1,
body > .editable > h1 {
    ...
}
```

Settings

These settings are defined, and can be overridden in your project settings

- `DJANGO_FRONT_PERMISSION`: a callable that gets passed a user object, and returns a boolean specifying whether or not the user is allowed to do front-end editing. Defaults to `lambda u:  u and u.is_staff`
- `DJANGO_FRONT_EDIT_MODE`: specifies whether the editor should be opened in a lightbox (default) or inline (over the edited element). Valid values are `'inline'` and `'lightbox'`.
- `DJANGO_FRONT_EDITOR_OPTIONS`: allows for options to be passed on to the editors (works with WYMeditor, Redactor, EpicEditor, CKEditor, Froala, Medium). This dictionary will be serialized as JSON and merged with the editor's base options. Defaults to `{}`. Example, to handle image uploads in Redactor:

```
DJANGO_FRONT_EDITOR_OPTIONS = {
    'imageUpload': '/path/to/image/handling/view/'
}
```

- `DJANGO_FRONT_ALLOWED_EDITORS`: list of allowed editor plugins. Add to this list if you plan on adding a new editor type. Defaults to `['ace', 'ace-local', 'wymeditor', 'redactor', 'epiceditor', 'ckeditor', 'froala', 'medium', 'default']`

Version history

Version 0.5.6

- Missing static folder (Issue #14, thanks @sekiroh)

Version 0.5.5

- Added support for Medium Editor

Version 0.5.4

- Test against Django 1.11

Version 0.5.3

- Test against Django 1.10b1

Version 0.5.2

- Fixes a possible unicode decode error on funky input

Version 0.5.1

- Support for running tests via setuptools

Version 0.5.0

- Supported Django versions are now 1.7, 1.8 and 1.9

Version 0.4.9

- Wrap all JavaScript plugins in their distinct scope receiving a local jQuery

Version 0.4.8

- Support for the Froala editor

Version 0.4.7

- Upgraded the CDN and bundled versions of ACE, EPIC and CKEditor

Version 0.4.6

- Test against Django 1.8

Version 0.4.5

- Fixed editor history on RedactorJS > 10.0
- Fixed documentation
- Generate documentation during tox tests

Version 0.4.4

- Added a missing migration
- Test against Django 1.8a
- Switched to tox

Version 0.4.3

- Added an API allowing copying content from one Placeholder instance to another (e.g. same name, different arguments)

Version 0.4.2

- Support for RedactorJS v10 API

Version 0.4.1 Version 0.4.0 ========== * Destroy editor before removing its container. Issues #6 and #7, thanks @syabro

Version 0.3.9

- Test against Django 1.7 final
- Use event delegation instead of direct binding on .editable blocks

Version 0.3.8

- Support both South and Django 1.7 native migrations, inspired by https://github.com/SmileyChris/easy-thumbnails

Version 0.3.7

- Test against Django 1.7RC1

Version 0.3.6

- Refactored JavaScript files to use "jQuery" instead of the shortcut ("$")

Version 0.3.5

- Missing image in the EpicEditor static. (Issue #5, thanks @twined)

Version 0.3.3

- Support for CKEditor

Version 0.3.2

- Shipping with documentation

Version 0.3.0

- History of content, possibility to move back to a previous version of the placeholder
- Massive rework of front-end side, modularization of editor plugins

Version 0.2.6

- Added an "ace-local" plugin, for when Ace is served locally

Version 0.2.4

- Add an extra class to the container, when the placeholder will be rendered empty
- Add a min-height on empty placeholders

Version 0.2.3

- Make sure the urlconf entry was added properly
- Set a min-height on Redactor
- New DJANGO_FRONT_EDITOR_OPTIONS settings allows for options to be passed on to the editors (works with WYMeditor, Redactor, EpicEditor)

Version 0.2.2

- Added support for the EpicEditor (thanks @daikeren - Issue #2)

Version 0.2.1

- Clarified the installation section of the README (mentioned that django.core.context_processors.request needs to be enabled in TEMPLATE_CONTEXT_PROCESSORS)
- Added the test project to the settings, so that it's easier to run tests

Version 0.2.0

- Test against Django 1.6b1

Version 0.1.9

- Python 3.3 support on Django 1.5+

Version 0.1.8

- Namespaced the layer and dialog CSS classes

Version 0.1.7

- Editing mode (lightbox or inline)

Version 0.1.6

- Support for Redactor 9 beta

Version 0.1.5

- Support for the Redactor editor

Version 0.1.4

- Include the Django Wymeditor theme, because django-wymeditor doesn't by default
- Push the STATIC_URL to the JavaScript context so that we don't have to assume it's /static/

Version 0.1.3

- Basic test cases

Version 0.1.2

- Support for WYMeditor (see note in README about installing django-wymeditor)

Version 0.1.1

- Settings (permissions)
- Cleanups

Version 0.1.0

- First release