django-resckeditor Documentation

Release 1.0.0

abidibo

Jan 23, 2019

Contents

1 Getting started **3**
1.1 Installation and Configuration 3
1.2 Usage 4

This is a django app which provides and infrastructure you can use to include custom contents of other apps directly inside ckeditor.

It defines a custom CKEDITOR plugin which implements two dialog tabs: the first to select the resource, and the second to dynamically set options.

CHAPTER 1

Getting started

1.1 Installation and Configuration

1.1.1 Installation

1. Install from PyPI:

```
pip install django-resckeditor
```

2. Add `ckeditor` and `resckeditor` to your `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    # ...
    'ckeditor',
    'resckeditor',
    # ...
)
```

1.1.2 Configuration

Configuring django-resckeditor is quite simple, you just need to define a list of resources made available to the editor. Each resource should define a function listing all its items (i.e. the news available for insertion, the media galleries ready to be included, ...) and a function responsible for generating the html output of a single item:

```
RESCKEDITOR_CONFIG = {
    'RESOURCES': [
        {
            'list': 'news.ckeditor.resources',
            'output': 'news.ckeditor.resource_output',
            'label': 'News'
        },
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

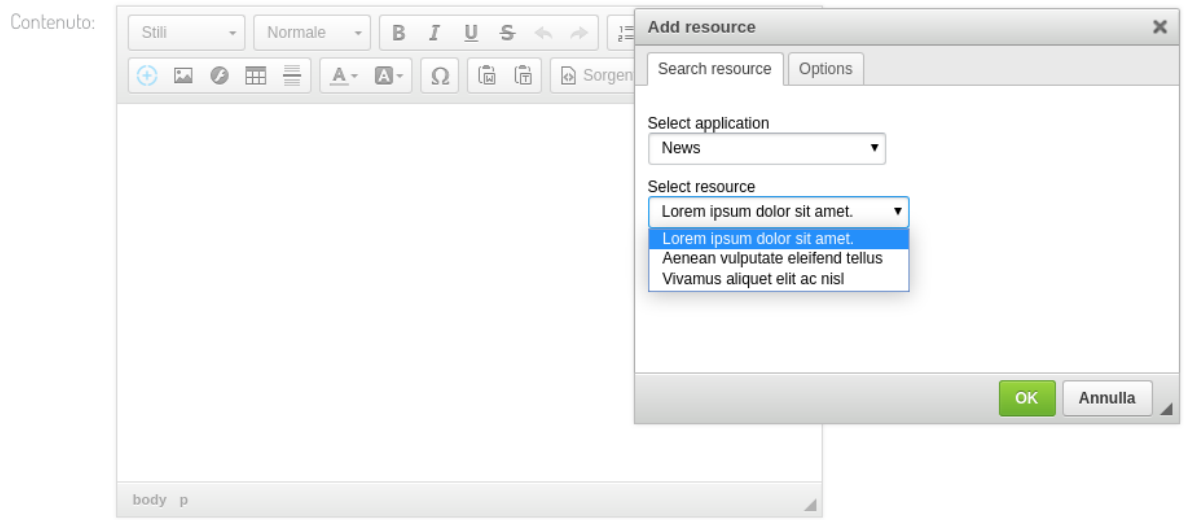
The label is used in the dropdown of the ckeditor plugin dialog window to select the desired resource.

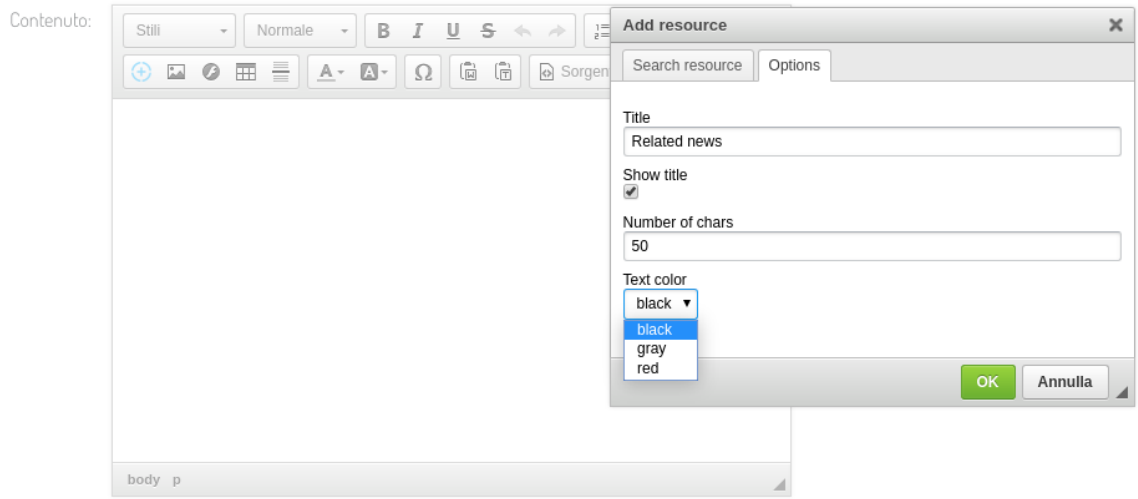
1.2 Usage

Imagine you want to allow an user to insert the extract of one of the last 5 news inside the editor. First, you can configure django-resckeditor as seen in the previous section.

So the *news* app must define a ckeditor module containing two functions: *resources* and *resource_output*.

The *list* function is invoked without arguments and should return a dictionary listing all the available items of the resource, and eventually the representation of the available options for the single resource. Such options will be available to the user in a tab of the plugin window. Currently only 4 types of options are supported: checkbox, text, number and select, but should be sufficient for almost all use cases.





news/ckeditor.py:

```
from .models import News

def resources():
    res = []
    for n in News.objects.published().order_by('-date')[:5]:
       res.append({
           'label': n.title,
           'id': n.id
       })
    return {
        'resources': res,
        'options': [
             {
                 'type': 'text',
                 'name': 'title',
                 'label': 'Title',
                 'default': 'Related news'
             },
             {
                 'type': 'checkbox',
                 'name': 'show-title',
                 'label': 'Show title',
                 'default': True
             },
             {
                 'type': 'number',
                 'name': 'num-chars',
                 'label': 'Number of chars',
                 'default': 50
             },
             {
                 'type': 'select',
                 'name': 'txt-color',
                 'label': 'Text color',
                 'data': [
                     {'label': 'black', 'value': '#000'},
                     {'label': 'gray', 'value': '#666'},
                     {'label': 'red', 'value': '#900'},
                 ]
             }
        ]
    }
```

The *output* function will receive the selected *id* and the dictionary of the provided *options* and should return the html output of the resource that will be inserted in the editor

news/ckeditor.py:

```
# this is just an example, use templates here
def resource_output(id, options):
   news = News.objects.get(id=id)
   if news:
      title = '<h2>' + options.get('title', 'Related news') + '</h2>'
      text = ''
      if options.get('show-title', False):
         text += '<h3>' + news.title + '</h3>'
         text += '<p style="color: ' + options.get('txt-color', '#000') + '">' + news.
↪text[:int(options.get('num-chars', 50))] + '</p>'
```

(continues on next page)

(continued from previous page)

```
    return '<section>' + title + text + '</section>'
  else:
    return ''
```