django-inplaceedit Documentation

Release 1.0.0

Pablo Martín

September 17, 2013

CONTENTS

CHAPTER

ONE

GETTING STARTED

1.1 Information

Inplace Edit Form is a Django application that allows you to inline edition of some data from the database

It is distributed under the terms of the GNU Lesser General Public License <http://www.gnu.org/licenses/lgpl.html>

1.2 Demo

Video Demo, of django-inplaceedit and Django-inlinetrans (Set full screen mode to view it correctly)

1.3 Usage

Calling inplace_edit tag you can inplace edit any data from the database. It is important to clarify that the inline edition is not only for text fields, although it may be the more common usage. You can also have inline edition for choices, boolean fields, date and datetime fields, foreing keys, many to many, file and image fields.

```
{% inplace_edit  "OBJ.FIELD_NAME" %}
{% inplace_edit  "OBJ.FIELD_NAME|FILTER1|FILTER2|...|FILTERN" %}
```

You only should change this:

```
<span>{{ user.first_name }}</span>
<span>{{ user.last_name }}</span>
```

to this:

```
<span>{% inplace_edit  "user.first_name" %}</span>
<span>{% inplace_edit  "user.last_name" %}</span>
```

CHAPTER

TWO

INSTALL

2.1 Requirements

- Django (>= 1.2, even works with 1.1 with some customizations in your project)
- jQuery (>=1.5.0)

2.2 In your base.html

Add

```
{% load inplace_edit %}
```

and wherever you load your static files, add either

```
{% inplace_toolbar %}
```

or

```
{% inplace_static %}
```

2.3 In your settings.py

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.admin',

    #.....................#

    'inplaceeditform',
)
```

And uncomment the request context processor:

```
TEMPLATE_CONTEXT_PROCESSORS = (
    #...#
    'django.core.context_processors.request',
```

```
    #...#
)
```

Optional:

```
INPLACEEDIT_EDIT_EMPTY_VALUE = 'Double click to edit'
INPLACEEDIT_AUTO_SAVE = True
INPLACEEDIT_EVENT = "dblclick"
INPLACEEDIT_DISABLE_CLICK = True  # For inplace edit text into a link tag
INPLACEEDIT_EDIT_MESSAGE_TRANSLATION = 'Write a translation' # transmeta option
INPLACEEDIT_SUCCESS_TEXT = 'Successfully saved'
INPLACEEDIT_UNSAVED_TEXT = 'You have unsaved changes'
INPLACE_ENABLE_CLASS = 'enable'
DEFAULT_INPLACE_EDIT_OPTIONS = {} # dictionnary of the optionals parameters that the templatetag can
DEFAULT_INPLACE_EDIT_OPTIONS_ONE_BY_ONE = True # modify the behavior of the DEFAULT_INPLACE_EDIT_OPT
ADAPTOR_INPLACEEDIT_EDIT = 'app_name.perms.MyAdaptorEditInline' # Explain in Permission Adaptor API
ADAPTOR_INPLACEEDIT = {'myadaptor': 'app_name.fields.MyAdaptor'} # Explain in Adaptor API
```

2.4 In your urls.py

```
urlpatterns = patterns('',

    #...#

    (r'^inplaceeditform/', include('inplaceeditform.urls')),

    #...#
)
```

If you use the date adaptor or datetime adaptor also:

```
js_info_dict = {
    'packages': ('django.conf',),
}

urlpatterns = patterns('',

    #...#

    (r'^inplaceeditform/', include('inplaceeditform.urls')),
    (r'^jsi18n$', 'django.views.i18n.javascript_catalog', js_info_dict),
)
```

CHAPTER

THREE

USAGE

Calling inplace_edit tag you can inplace edit any data from the database. It is important to clarify that the inline edition is not only for text fields, although it may be the more common usage. You can also have inline edition for choices, boolean fields, date and datetime fields, foreing keys, many to many, file and image fields.

```
{% inplace_edit  "OBJ.FIELD_NAME" %}
{% inplace_edit  "OBJ.FIELD_NAME|FILTER1|FILTER2|...|FILTERN" %}
```

3.1 Example

```
{% load inplace_edit %}
   <html>
   <head>
   ...
   <script src="{{ STATIC_URL }}js/jquery.min.js" type="text/javascript"></script>
   {% inplace_toolbar %}
   </head>
   <body>
       ...
       <div id="content">
           ...
           {% inplace_edit "content.name" %}
           ...
           <div class="description">
               {% inplace_edit "content.date_initial|date:'d m Y'" %}
               {% inplace_edit "content.description|safe" %}
           </div>
           <div class="body">
               {% inplace_edit "content.body|safe|truncatewords_html:15" %}
           </div>
       </div>
       ...
   </body>
   </html>
```

3.1.1 How to use it

- If you use inplace_static: Just pass the cursor above the field and double click (this is *customizable*), authenticated with a super user (this is also *customizable*)

- If you use inplace_toolbar: Enable a edit inline and just pass the cursor above the field and double click (this is *customizable*), authenticated with a super user (this is also *customizable*)

CHAPTER

FOUR

ADVANCED USAGE

Inplaceedit has some optionals parameters that the templatetag can receive to change its behavior:

- auto_height: Adapt the height's widget to the tag container.
- auto_width: Adapt the width's widget to the tag container.
- min_width: The minimum of the width's widget
- class_inplace: Add a class to edit inline form.
- tag_name_cover: The value is covered for a span. But it's possible to change it.
- filters_to_show: The server filters the value before to save. List separate for "|"
- loads: If you use some filter that need a load, you set this option. List separate for ":"
- edit_empty_value: The text to display when the field is empty

4.1 Examples

```
{% inplace_edit "content.description|safe" auto_height=1, auto_width=1 %}
{% inplace_edit "content.title" class_inplace="titleFormEditInline" %}
{% inplace_edit "content.description|safe" filters_to_show="safe|truncatewords_html:30", tag_name_cov
{% inplace_edit "content.description|my_filter" loads="my_template_tag" %}
{% inplace_edit "content.index" edit_empty_value="This is a editable content, now the value is none.
{% inplace_edit "content.amount" min_width="100" %}
```

CHAPTER

FIVE

CUSTOMIZING DJANGO-INPLACEEDIT

django-inplaceedit is generic, customizable and extensible.

5.1 Permission Adaptor API

By default you can inline edit a field if you are authenticated with a superuser. But it's customizable:

5.1.1 Overwriting the default permission adaptor

This package have two inplementations:

- SuperUserPermEditInline (by default): Only you can edit if you are super user
- AdminDjangoPermEditInline: Yo can edit the content if you have a permission edit for that model. If you want enabled this, write in your settings:

```
ADAPTOR_INPLACEEDIT_EDIT = 'inplace_edit.perms.AdminDjangoPermEditInline'
```

You can create a specify adaptor. MyAdaptorEditInline is a class with a single class method, this method receives a adaptor field

```
# in your settings

ADAPTOR_INPLACEEDIT_EDIT = 'app_name.perms.MyAdaptorEditInline'


# in app_name.perms

class MyAdaptorEditInline(object):

    @classmethod
    def can_edit(cls, adaptor_field):
       return True # All user can edit
```

5.1.2 Example

```
class MyAdaptorEditInline(object):

    @classmethod
    def can_edit(cls, adaptor_field):
```

```
    user = adaptor_field.request.user
    obj = adaptor_field.obj
    can_edit = False
    if user.is_anonymous():
        pass
    elif user.is_superuser:
        can_edit = True
    else:
       can_edit = has_permission(obj, user, 'edit')
    return can_edit
```

5.2 Creating an adaptor

You can create an adaptor to work with django-inplaceedit, the behavior is fully customizable. To default inplaceedit has 17 adaptors (AdaptorTextField, AdaptorTextAreaField, AdaptorChoicesField, AdaptorBooleanField, AdaptorDateTimeField, AdaptorForeingKeyField, AdaptorManyToManyField, AdaptorImageField etc). These use the api, overwriting some methods for them.

You can see four examples in django-inplaceedit-extra-fields project

5.2.1 First step

In your settings:

```
ADAPTOR_INPLACEEDIT = {'myadaptor': 'app_name.fields.MyAdaptor'}
```

In app_name.fields.MyAdaptor:

```
class MyAdaptor(BaseAdaptorField):

    @property
    def name(self):
        return 'myadaptor'
```

You can overwrite a default adaptor. To overwrite a adaptor add in your settings something like this:

```
ADAPTOR_INPLACEEDIT = {'text': 'app_name.fields.MyAdaptorText'}
```

For this case you overwrite the AdaptorText with MyAdaptorText.

5.2.2 Python API

- loads_to_post: It returns the value of the request (normally request.POST)
- classes: Classes of tag cover. By default "inplaceedit" and "myadaptorinplaceedit"
- get_config: Preprocessed of the configuration. By default, it does nothing.
- get_form_class: It returns the form class.
- get_form: It returns a instace of form class.
- get_field: It returns a field of instance of form class.
- render_value: It returns the render of the value. If you write {% inplace_edit "obj.name|filter1" %} it returns something like this {{ obj.name|filter1 }}.

- render_value_edit: It returns the render value if you can edit. It returns by default the same of "render_value", but if the value is None call to empty_value
- empty_value: It returns an empty value for this adaptor. By default, 'Dobleclick to edit'.
- render_field: It returns the render of form, with a field.
- render_media_field: It returns the media (scripts and css) of the field.
- render_config: It returns the render of config.
- can_edit: It returns a boolean that indicate if this user can edit inline this content or not.
- get_value_editor: It returns a clean value to be saved in DB.
- save: Save the value in DB.
- get_auto_height: Returned if the field rendered with auto height
- get_auto_width: Returned if the field rendered with auto width
- treatment_height: Special treatment to widget's height.
- treatment_width: Special treatment to widget's width.

```
If you want to use own options in your adaptor, you can do it. These options will be in self.config i
{% inplace_edit "obj.field_name" my_opt1="value1", my_opt2="value2" %}
```

5.2.3 JavaScript API

You can change the javascript behaviour by adding or overriding methods from the original implementation by adding the special file jquery.inplaceeditform.hooks.js to your project. $.inplaceeditform.extend takes an object with the new or replacement methods.

```
$.inplaceeditform.extend(
    {
        inplaceApplySuccessShowMessage: function(inplace_span) {
            var self = $.inplaceeditform;
            if (self.opts.successText) {
                var modal = $('#inplaceedit-modal');
                var body = modal.find('div.modal-body p');
                body.html(self.opts.successText);

                setTimeout(function () {
                    modal.fadeOut(function () {
                        $(this).remove();
                    });
                }, 2000);
            }
            modal.show();
        }
    }
);
```

Additionally there are four hooks,

- getValue: if the value is componing for various widgets, you can set the function getValue, to these DOM elements. Something like this:

```
<script type="text/javascript">
    (function($){
        $(document).ready(function () {
            function myGetValue(form, field_id) {
                return ""Something"";
            }
            $(".applyMyAdaptor").data("getValue", myGetValue);
    });
    })(jQuery);
</script>
```

- applyFinish: if you need/want to do some action after the value be saved. Something like this:

```
<script type="text/javascript">
    (function($){
        $(document).ready(function () {
            function myApplyFinish() {
                return ""Something"";
            }
            $(".applyMyAdaptor").data("applyFinish", myApplyFinish);
    });
    })(jQuery);
</script>
```

- cancelFinish: if you need/want to do some action after the cancel the edit. Something like this:

```
<script type="text/javascript">
    (function($){
        $(document).ready(function () {
            function myCancelFinish() {
                return ""Something"";
            }
            $(".cancelMyAdaptor").data("cancelFinish", myCancelFinish);
    });
    })(jQuery);
</script>
```

- extraConfig: if you need/want add something to the config in the ajax request to print the field

```
<script type="text/javascript">
    (function($){
        $(document).ready(function () {
            function myExtraConfig(data) {
                return data + ""Something"";
            }
            $(".configMyAdaptor").data("extraConfig", myExtraConfig);
    });
    })(jQuery);
</script>
```

For example the adaptor datetime use these hooks.

CHAPTER

SIX

TESTING

This django application has been tested on severals browsers: Firefox, Google Chrome, Opera, Safari and Internet Explorer on versions 7, 8, 9 and 10 to check javascript actions. **Attention**, jQuery 2 does not support IE 6/7/8. If you want to test in these browsers, please use jQuery 1 (recomended the last release 1.10.2)

Exists a testing django project. This project can use as demo project, because django-inplaceedit is totally adapted to it.

Also you can use the demo project of django-inplaceedit-bootstrap

The backend has unit test, to check it.:

```
cd testing
python run_tests.py
```

You can see this django-inplaceedit works (without changes) from django 1.2, and even in previous versions of Django (1.1 and 1.0) works with some customizations.

CHAPTER

SEVEN

OTHER PACKAGES RELATED

7.1 Transmeta

This egg is compatible with Transmeta But it is not a requirement

7.2 Django Inplace Edit Extra Field

If you want to get more download Django Inplace Edit Extra Field

7.3 Django Inplace Edit Bootstrap

If you want to get more download Django Inplace Edit Bootstrap

CHAPTER

EIGHT

DEVELOPMENT

You can get the last bleeding edge version of inplaceedit by doing a checkout of its git repository:

```
git clone git://github.com/Yaco-Sistemas/django-inplaceedit.git
```