thecut-forms Documentation

Release 0.5

The Cut Creative

May 23, 2017

Contents

Contents:

CHAPTER 1

Welcome to thecut-forms

Form rendering helpers.

Features

- Automatically add appropriate HTML5 `type`, `required` and `maxlength` attributes to form fields.
- Automatically add `date`, `time`, and `datetime` CSS classes to appropriate form fields.
- Easily add custom `placeholders` to form fields by editing a `dict`.
- Easily render forms in your templates in a well-designed standardised way that makes front-end development easier.

Documentation

The full documentation is at https://thecut-forms.readthedocs.org.

Quickstart

Install `thecut-forms` using the *Installation instructions*.

Use one of the many available `django.forms.Form` mixins on your `django.forms.Form`:

```
from django import forms
from thecut.forms import EmailTypeMixin, TimeClassMixin

class MyForm(EmailTypeMixin, TimeClassMixin, forms.Form):

    foo = forms.EmailField(required=True)
```

```
    bar = forms.TimeField(required=True)
```

Or use *thecut.forms.forms.FormMixin* to get them all at once:

```
from django import forms
from thecut.forms import FormMixin

class MyForm(FormMixin, forms.Form):

    foo = forms.CharField(required=True)
```

See *Form Mixins* for more information.

In your template, use the `forms/_form.html` snippet to easily render your forms:

```
{% include "forms/_form.html" %}
```

See *Templates* for more information.

Credits

See *Credits*.

CHAPTER 2

Installation instructions

1. Install via pip / pypi:

```
$ pip install thecut-forms
```

2. Add to your project's `INSTALLED_APPS` setting:

```
INSTALLED_APPS = [
    # ...
    'thecut.forms'
    # ...
]
```

3. Sync your project's migrations:

```
$ python manage.py migrate forms
```

CHAPTER 3

Usage

Form Mixins

The following mixins can be applied to any `Form`-type object.

`EmailTypeMixin`

class `thecut.forms.forms.`**`EmailTypeMixin`**(**args*, ***kwargs*)

A mixin for a `Form` that sets the HTML5 `email` input type on any child `EmailField` instances.

`RequiredMixin`

class `thecut.forms.forms.`**`RequiredMixin`**(**args*, ***kwargs*)

A mixin for a `Form` that sets the HTML5 `required` attribute on any child `Field` instances that is required.

This mixin does not apply the *required* attribute to fields using `RadioSelect` and `CheckboxSelectMultiple` as the HTML5 `required` attribute does not behave as (usually) expected on these widgets.

`MaxLengthMixin`

class `thecut.forms.forms.`**`MaxLengthMixin`**(**args*, ***kwargs*)

A mixin for a `Form` that sets the HTML5 `maxlength` attribute on any child `Field` instances using the `Textarea` widget.

A `max_length` must be specified on the `Field`.

PlaceholderMixin

class thecut.forms.forms.**PlaceholderMixin**(**args*, ***kwargs*)
A mixin for a Form that allows you to easily set the HTML5 placeholder widget on a child Field.

To add a placeholder to a Field, specify it in a placeholders dict on the Form's Meta class. For example:

```
class MyForm(forms.Form):

    foo = forms.CharField()

    class Meta(object):
        placeholders = {
            'foo': 'Enter some text here.'
        }
```

TimeClassMixin

class thecut.forms.forms.**TimeClassMixin**(**args*, ***kwargs*)
A mixin for a Form that adds a time CSS class on any child Field instances using the TimeInput widget..

DateClassMixin

class thecut.forms.forms.**DateClassMixin**(**args*, ***kwargs*)
A mixin for a Form that adds a date CSS class on any child Field instances using the DateInput widget..

DateTimeClassMixin

class thecut.forms.forms.**DateTimeClassMixin**(**args*, ***kwargs*)
A mixin for a Form that adds a datetime CSS class on any child Field instances using the DateTimeInput widget..

FormMixin

In order to make it easy to use all of the above mixins, we have provided *thecut.forms.forms.FormMixin* which inherits from all other mixins.

class thecut.forms.forms.**FormMixin**(**args*, ***kwargs*)
Bases: *thecut.forms.forms.DateTimeClassMixin*, *thecut.forms.forms.DateClassMixin*, *thecut.forms.forms.EmailTypeMixin*, *thecut.forms.forms.MaxLengthMixin*, *thecut.forms.forms.PlaceholderMixin*, *thecut.forms.forms.RequiredMixin*, *thecut.forms.forms.TimeClassMixin*

Form mixin.

Used to extend a standard Django Form class with useful/common behaviour.

Templates

We provide a Django template snippet that can be included in your template files to easily render forms. Among other things, it handles rendering:

- The CSRF token (if required).
- A honeypot field (if required).
- The parent `<form>` element.
- Non-field errors.
- Form fields (in a way that makes front-end styling much easier).
- A submit button.

Basuc usage

In order to render a basic form in your template, just include `` `forms/_form.html ``:

```
{% include "forms/_form.html" %}
```

You do *not* need to define a `<form>` element, call `{% csrf_token %}`.etc - the snippet will handle this for you.

Customising the form

The form template makes use of template context variables to allow you to customise some aspects of its functionality.

Variable	Description	Default	Example
`form`	The form to render.	`form`	`{% include "forms/_form.html" with form=my_form %}`
`form_action`	The URL to post the form to.	`request.path`	`{% url 'site:homepage' as my_url %} {% include "form/_form.html" with form_action=my_url %}`
`form_method`	The HTTP method to use.	POST	`{% include "forms/_form.html" with form_method="GET" %}`
`form_honeypot_field`	A honeypot field to include in the form.	•	`{% include "forms/_form.html" with form_honeypot_field=my_field %}`
`form_submit_value`	The value (label) for the form's submit button.	Submit	`{% include "forms/_form.html" with form_submit_value="Send" %}`

CHAPTER 4

Testing

Running unit tests

Using your system's Python / Django

You can perform basic testing against your system's Python / Django.

1. Install the test suite requirements:

```
$ pip install -r requirements-test.txt
```

2. Ensure a version of Django is installed:

```
$ pip install Django
```

3. Run the test runner:

```
$ python runtests.py
```

Using a virtualenv

You can use `virtualenv` to test without polluting your system's Python environment.

1. Install `virtualenv`:

```
$ pip install virtualenv
```

2. Create and activate a `virtualenv`:

```
$ cd thecut-forms
$ virtualenv .
$ source bin/activate
(thecut-forms) $
```

3. Follow 'Using your system's Python / Django' above.

Using tox

You can use tox to automatically test the application on a number of different Python and Django versions.

1. Install `tox`:

```
$ pip install -r requirements-test.txt
```

2. Run `tox`:

```
(thecut-forms) $ tox --recreate
```

Tox assumes that a number of different Python versions are available on your system. If you do not have all required versions of Python installed on your system, running the tests will fail. See `tox.ini` for a list of Python versions that are used during testing.

Test coverage

The included `tox` configuration automatically detects test code coverage with `coverage`:

```
$ coverage report
```

Available tests

TestEmailTypeMixin

class thecut.forms.tests.test_forms.**TestEmailTypeMixin**(*methodName='runTest'*)

Tests for the *thecut.forms.forms.EmailTypeMixin* class.

test_input_type_not_set_to_email_for_non_emailfield()

Test that the HTML `input` attribute is not set to `email` on a child `django.forms.Field` that is not `django.forms.EmailField`.

test_input_type_set_to_email_for_emailfield()

Test that the HTML `input` attribute is set to `email` on a child `django.forms.EmailField`.

TestRequiredMixin

class thecut.forms.tests.test_forms.**TestRequiredMixin**(*methodName='runTest'*)

Tests for the *thecut.forms.forms.RequiredMixin* class.

test_required_attribute_not_set_for_optional_field()

Test that the HTML5 `required` attribute is not set on a child `django.forms.Field` that does not have `required` set to `True`.

test_required_attribute_not_set_for_required_checkbox_widget()

Test that the HTML5 `required` attribute is not set on a child `django.forms.Field` that has `required` set to `True` and uses the `django.forms.CheckboxSelectMultiple` widget.

test_required_attribute_not_set_for_required_radio_widget()
Test that the HTML5 `required` attribute is not set on a child `django.forms.Field` that has `required` set to `True` and uses the `django.forms.RadioSelect` widget.

test_required_attribute_set_for_required_field()
Test that the HTML5 `required` attribute is set on a child `django.forms.Field` that has `required` set to `True`.

TestMaxLengthMixin

class thecut.forms.tests.test_forms.**TestMaxLengthMixin**(*methodName='runTest'*)
Tests for the *thecut.forms.forms.MaxLengthMixin* class.

test_correct_maxlength_set_for_textarea_with_max_length()
Test if the correct HTML5 `maxlength` attribute is set on a child `django.forms.Field` using `django.forms.Textarea`and with ``max_length` set.

test_no_maxlength_for_non_textarea_with_max_length()
Test if no HTML5 `maxlength` attribute is set on a child `django.forms.Field` not using `django.forms.Textarea` but with `max_length` set.

test_no_maxlength_for_non_textarea_with_no_max_length()
Test if no HTML5 `maxlength` attribute is set on a child `django.forms.Field` not using `django.forms.Textarea` and with no `max_length` set.

test_no_maxlength_for_textarea_with_no_max_length()
Test if no HTML5 `maxlength` attribute is set on a child `django.forms.Field` using `django.forms.Textarea` and with no `max_length` set.

TestPlaceholderMixin

class thecut.forms.tests.test_forms.**TestPlaceholderMixin**(*methodName='runTest'*)
Tests for the *thecut.forms.forms.PlaceholderMixin* class.

test_placeholder_not_set_when_not_defiend()
Test if the correct HTML5 `placeholder` attribute is not set on a py:class:*django.forms.Field* when no appropriate entry is added to the `placeholders` dict.

test_placeholder_set_when_defiend()
Test if the correct HTML5 `placeholder` attribute is set on a field when an appropriate entry is added to the `placeholders` dict.

TestTimeClassMixin

class thecut.forms.tests.test_forms.**TestTimeClassMixin**(*methodName='runTest'*)
Tests for the *thecut.forms.forms.TimeClassMixin* class.

test_time_class_added_for_timefield()
Test if the `time` CSS class is applied to a child py:class:*django.forms.Field* using the `django.forms.TimeInput` widget.

test_time_class_not_added_for_nontimefield()
Test if the `time` CSS class is not applied to a child py:class:*django.forms.Field* not using the `django.forms.TimeInput` widget.

TestDateClassMixin

class thecut.forms.tests.test_forms.**TestDateClassMixin**(*methodName='runTest'*)
Tests for the *thecut.forms.forms.DateClassMixin* class.

test_date_class_added_for_datefield()
Test if the `date` CSS class is applied to a child py:class:*django.forms.Field* using the `django.forms.DateInput` widget.

test_date_class_not_added_for_nondatefield()
Test if the `date` CSS class is not applied to a child py:class:*django.forms.Field* not using the `django.forms.DateInput` widget.

TestFormMixin

class thecut.forms.tests.test_forms.**TestFormMixin**(*methodName='runTest'*)
Tests for the *thecut.forms.forms.EmailTypeMixin* class.

CHAPTER 5

History

0.4 (2016-07-06)

- Added Sphinx documentation environment.
- Added installation / usage / testing documentation.
- Moved some plain-text documentation over to reStructuredText.
- Added `tox`-based testing environment.
- Added unit tests for a majority of functionality.
- Added continuous integration with Travis.
- Added code coverage with codecov.
- Improved `setup.py`.

0.3.11 (2016-02-28)

- Improved unicode support in `version.py`.
- Added support for rendering honeypot field whilst using the in-built form rendering templates.

0.3.10 (2015-12-09)

- Gracefully handle situations where Meta `placeholders` attribute does not exist.

0.3.9 (2015-02-24)

- Created `PlaceholderMixin` to allow easy addition of custom placeholder text.
- Added `PlaceholderMixin` to `FormMixin`.

0.3.8 (2015-01-12)

- Added `LICENSE`, `AUTHORS`, `README`.

0.3.7 (2014-12-12)

- Bugfix: in form rendering template, render hidden fields.

0.3.6 (2014-12-03)

- In form rendering template, add class to field `<li>` wrapper with input type.

0.3.5 (2014-12-03)

- Redesigned form rendering template to allow for easier styling.
- Updated `version.py` to work with Python 3.

0.3.4 (2014-09-03)

- Added `DateTimeTimezoneMixin`.

0.3.3 (2014-06-30)

- In form rendering templates, separate hidden fields and visible fields.

0.3.2 (2014-04-15)

- Added missing template files to `MANIFEST.in`.

0.3.1 (2014-03-19)

- Added form rendering templates to improve rendering of forms in templates.
- Removed `distribute` from application requirements.

0.3 (2014-02-03)

- Apply HTML5 `maxlength` attribute to `Textarea` widgets when a maximum length has been specified on the field.

0.2 (2014-01-30)

- Do not appy `required` attribute to certain widgets (`forms.CheckboxSelectMultiple` or `forms.RadioSelect`) as the HTML5 `required` attribute does not behave correctly on the resulting HTML fields.

0.1 (2013-09-10)

- Initial release
- Use appropriate HTML5 `type` fields for email, time, date, and datetime fields / widgets.
- Apply HTML5 `required` attribute to required fields.

CHAPTER 6

Credits

- Matt Austin <matt.austin@thecut.net.au>
- Josh Crompton <josh.crompton@thecut.net.au>
- Mark Lockett <mark.lockett@thecut.net.au>
- Kye Russell <kye.russell@thecut.net.au>
- Elena Williams <elena.williams@thecut.net.au>

Index