django-tapeforms

unknown

Feb 25, 2023

CONTENTS

CHAPTER

ONE

ALL CONTENTS

1.1 Installation

django-tapeforms supports Python 3 only and requires at least Django 2.2. No other dependencies are required.

To start, simply install the latest stable package using the command

```
$ pip install django-tapeforms
```

In addition, you have to add `'tapeforms'` to the `INSTALLED_APP` setting in your `settings.py`.

Thats it, now continue to the *Usage section* to learn how to render your forms to HTML.

1.2 Usage

Tapeforms provides a *Mixin* to add the required functionality to the Django forms which is required for rendering the whole form and its fields using Django templates.

```
from django import forms
from tapeforms.mixins import TapeformMixin

class ContactForm(TapeformMixin, forms.Form):
    name = forms.CharField()
    email = forms.EmailField()
    text = forms.CharField(widget=forms.Textarea)
```

Together with some template code you get some beautiful rendered forms.

```
{% load tapeforms %}

<form method="post" action=".">
    {% csrf_token %}
    {% form form %}
    <button type="submit">Submit</button>
</form>
```

By default, the form will be rendered using the `tapeforms/layouts/default.html`. To learn how to override the template which is used, please refer to the *Advanced usage section*.

If you don't want to override the template, you might also use the *as_tapeform* shortcut.

```
<form method="post" action=".">
    {% csrf_token %}
    {{ form.as_tapeform }}
    <button type="submit">Submit</button>
</form>
```

Note: For a full overview of the methods you might override to customize the rendering of your forms, go to the *API Reference for mixins*.

1.3 Advanced usage

Most of the advanced features are also demo'ed in the examples project you can find in the *django-tapeforms* codebase. Go to the `examples` directory.

Note: Most class properties have a corresponding method to access the value. This helps in cases where you might want to manipulate the response in a more dynamic way. If you're unsure what can be changed, refer to the *API Reference*.

1.3.1 Overriding layout templates

In the context of *django-tapeforms* the layout template is the outer part of a form which contains the non field errors and the loop over the fields to render.

If you want to override the template used for rendering the form layout, the best way is defining a property on the form class:

```
class MyForm(TapeformMixin, forms.Form):
    layout_template = 'another-form-template.html'

    field1 = forms.CharField()
```

You are done. The `form` *template tag* will pick the new defined template for rendering.

If you need to select the layout template on other things like the instance in `ModelForm`, you can overwrite the *`get_layout_template`* method.

1.3.2 Overriding field templates

In the context of *django-tapeforms* the field template is the part of a form which contains the label, widget, errors and help text for a certain field.

If you want to override the template used for rendering fields, there is more than one way.

When the template should be changed for all fields, you can simply set the `field_template` property.

```
class MyForm(TapeformMixin, forms.Form):
    field_template = 'my-field-template.html'
```

(continues on next page)

(continued from previous page)

```
    field1 = forms.CharField()
```

The `formfield` *template tag* will now pick the new defined template for rendering the fields.

You can define the templates used for field rendering in a more specific way.

```
class MyForm(TapeformMixin, forms.Form):
    field_template_overrides = {
        'field1': 'my-field1-template.html',
        forms.IntegerField: 'my-number-template.html',
    }

    field1 = forms.CharField()
    field2 = forms.IntegerField()
    field3 = forms.IntegerField()
```

As you can see, you can override the templates for fields based on the *field name* and also based on the *field class*.

If you need to select the field template depending on other things, you can overwrite the *get_field_template* method. It receives a `BoundField` instance for the template selection.

1.3.3 Overriding widget templates

In the context of *django-tapeforms* and *Django* itself, the widget template is used for the actual input element.

If you want to override the template used for rendering widgets, you can change the `template_name` by subclassing the widget classes but this requires much effort.

To make this easier, the *TapeformMixin* provided a helper to set the widget `template_name`. The matching is done using the field name and widget class.

```
class MyForm(TapeformMixin, forms.Form):
    widget_template_overrides = {
        'field1': 'my-field1-widget-template.html',
        forms.NumberInput: 'my-number-widget-template.html',
    }

    field1 = forms.CharField()
    field2 = forms.IntegerField()
    field3 = forms.IntegerField()
```

If you need to select the widget template based on other things, you can overwrite the *get_widget_template* method. It receives the field name as `str` and the `Field` instance.

1.3.4 Changing the applied CSS classes

When you render the form using *django-tapeforms* you can apply CSS classes to the field container, label and widget.

This is done using the properties `field_container_css_class`, `field_label_css_class` and `widget_css_class`.

For all CSS class properties, there are methods to override the applied CSS class per field. Please refer to the *API Reference* to learn what arguments are passed to the CSS class methods.

1.3.5 Adding CSS classes to invalid field

When you render the form using *django-tapeforms*, you can also apply additional CSS classes to the label and widget of a field which has errors.

This is done using the properties `field_label_invalid_css_class` and `widget_invalid_css_class`.

If you need to set more attributes to the widget when there are errors, you can overwrite the *apply_widget_invalid_options* method. It receives the field name as `str`.

1.4 Fieldsets

Learn how to organize your form fields in multiple fieldsets and have them rendered nicely.

1.4.1 Manual fieldsets

The first way to get fieldsets for organizing your form's fields is using the `TapeformFieldset` directly. This is also the low-level approach with full control of whats happening.

```
class MyForm(TapeformMixin, forms.Form):
    field1 = forms.CharField()
    field2 = forms.CharField()
    field3 = forms.CharField()
    field4 = forms.CharField(widget=forms.HiddenInput)

    def first_fieldset(self):
        return TapeformFieldset(self, fields=('field1', 'field2'), primary=True)

    def second_fieldset(self):
        return TapeformFieldset(self, exclude=('field1', 'field1'))
```

Note: You have to make sure that at least one fieldset is marked as your *primary* fieldsets. This is required because only the *primary* fieldset will render the non field errors and all hidden fields of the form.

The fieldsets can then be rendered using the `form` template tag just like classic forms.

```
<form action="." method="post" >
    {% csrf_token %}
    <fieldset>
        {% form form.first_fieldset %}
```

(continues on next page)

(continued from previous page)

```
    </fieldset>
    <fieldset>
        {% form form.second_fieldset %}
    </fieldset>
    <button type="submit">Submit</button>
</form>
```

1.4.2 Generated fieldsets

It might come to your mind that defining alle the fieldsets using methods is a bit to much boilerplate.

Because of this, django-tapeforms provides another *Mixin* you can use to make generation of fieldsets a lot easier.

Lets have a look on an example.

```
class MyForm(TapeformMixin, forms.Form):
    field1 = forms.CharField()
    field2 = forms.CharField()
    field3 = forms.CharField()
    field4 = forms.CharField(widget=forms.HiddenInput)

    fieldsets = [
        {'fields': ('field1', 'field2')},
        {'exclude': ('field1', 'field2')},  # Render all remaining fields
    ]
```

Also, the template is simpler now.

```
<form action="." method="post" >
    {% csrf_token %}
    {% for fieldset in form.get_fieldsets %}
        <fieldset>
            {% form fieldset %}
        </fieldset>
    {% endfor %}
    <button type="submit">Submit</button>
</form>
```

While the difference in code written might not be that big when rendering two fieldsets, imagine the difference when having lets say 7 oder 8 fieldsets.

As you can see, we don't have to care for the *primary* flag anymore. The `get_fieldsets` methods make sure that one fieldset is the primary fieldset (by default, the first fieldset is marked as *primary*).

There are many methods in the *TapeformFieldsetsMixin* you can override to get your hands on the generation process (like selection the right fieldset class or manipulating the data which is used to instantiate the fieldset).

It is also possible to generate the fieldsets configuration on the fly by overriding the `get_fieldsets` method and pass a config to your super call.

```
class MyForm(TapeformMixin, forms.Form):
    field1 = forms.CharField()
    field2 = forms.CharField()
```

(continues on next page)

(continued from previous page)

```
    field3 = forms.CharField()
    field4 = forms.CharField(widget=forms.HiddenInput)

    def get_fieldsets(self):
        # Geneate a fieldset for every form field. Why would one do that?
        return super().get_fieldsets([
            {'fields': (field.name,)}
            for field in self.visible_fields()
        ])
```

1.4.3 Passing around additional data

In more compex setups you might want to pass around additional data. In our example we assume that we require a css class added to the fieldset element.

```
class MyForm(TapeformMixin, forms.Form):
    field1 = forms.CharField()
    field2 = forms.CharField()
    field3 = forms.CharField()
    field4 = forms.CharField(widget=forms.HiddenInput)

    fieldsets = [{
        'fields': ('field1', 'field2'),
        'extra': {'css_class': 'my-class-foo'}
    }, {
        'exclude': ('field1', 'field2'),
        'extra': {'css_class': 'my-class-bar'}
    }]
```

```
<form action="." method="post" >
    {% csrf_token %}
    {% for fieldset in form.get_fieldsets %}
        <fieldset class="{{ fieldset.extra.css_class }}">
            {% form fieldset %}
        </fieldset>
    {% endfor %}
    <button type="submit">Submit</button>
</form>
```

The extra key in the fieldset configuration is not checked in any way. Its just passed around. You might use it to carry things in a `dic` like in the example or push a model instance to the template for further use.

1.4.4 Advanced usage

For a full overview of the methods TapeformFieldset and TapeformFieldsetsMixin provide, go to the *API Reference for fieldsets*.

1.5 Contrib

Extra, optional, features of *django-tapeforms*.

1.5.1 Bootstrap mixin

You can use the `tapeforms.contrib.bootstrap.Bootstrap4TapeformMixin` to render forms with a Bootstrap 4 compatible HTML layout / CSS classes, or `tapeforms.contrib.bootstrap.Bootstrap5TapeformMixin` for Bootstrap 5.

This alternative mixin makes sure that the rendered widgets, fields and labels have the correct css classes assigned.

In addition, the mixin uses a different template for the fields because Bootstrap requires that the ordering of label and widget inside a field is swapped (widget first, label second).

1.5.2 Foundation mixin

You can use the `tapeforms.contrib.foundation.FoundationTapeformMixin` to render forms with a Foundation compatible HTML layout / css classes.

This alternative mixin makes sure that the rendered widgets, fields and labels have the correct CSS classes assigned especially in case of errors.

In addition, the mixin uses a different template for the fields. It is required to swap the ordering of label and widget when rendering a checkbox, and also to wrap mulitple inputs - e.g. a group of checkboxes or radio buttons - in a `fieldset` element, as Foundation documentation suggests.

1.6 API Reference

1.6.1 Mixins

class tapeforms.mixins.**TapeformLayoutMixin**

Bases: `object`

Mixin to render a form of fieldset as HTML.

layout_template = None

Layout template to use when rendering the form. Optional.

get_layout_template(*template_name=None*)

Returns the layout template to use when rendering the form to HTML.

Preference of template selection:

1. Provided method argument *template_name*
2. Form class property *layout_template*

3. Globally defined default template from *defaults.LAYOUT_DEFAULT_TEMPLATE*

Parameters
template_name – Optional template to use instead of other configurations.

Returns
Template name to use when rendering the form.

`get_layout_context()`

Returns the context which is used when rendering the form to HTML.

The generated template context will contain the following variables:

- form: *Form* instance
- errors: *ErrorList* instance with non field errors and hidden field errors
- hidden_fields: All hidden fields to render.
- visible_fields: All visible fields to render.

Returns
Template context for form rendering.

`as_tapeform()`

Shortcut to render the form as a "tapeform" without including the tapeforms templatetags. Behaves similar to *as_p* and *as_table*.

`class tapeforms.mixins.TapeformMixin(*args, **kwargs)`

Bases: `TapeformLayoutMixin`

Mixin to extend the forms capability to render itself as HTML output. (using the template tags provided by *tapeforms*).

`field_template = None`

Field template to use when rendering a bound form-field. Optional.

`field_template_overrides = None`

A dictionary of form-field names and/or form-field classes to override the field template which is used when rendering a certain form-field. Optional.

`field_container_css_class = 'form-field'`

The CSS class to apply to the form-field container element.

`field_label_css_class = None`

CSS class to append to the rendered field label tag. Optional.

`field_label_invalid_css_class = None`

An additional CSS class to append to the rendered field label tag when the field has errors. Optional.

`widget_template_overrides = None`

A dictionary of form-field names and/or widget classes to override the widget template which is used when rendering a certain form-field. Optional.

`widget_css_class = None`

CSS class to append to the widget attributes. Optional.

`widget_invalid_css_class = None`

An additional CSS class to append to the widget attributes when the field has errors. Optional.

`__init__`(**args*, ***kwargs*)

The init method is overwritten to apply widget templates and CSS classes.

`full_clean`(**args*, ***kwargs*)

The full_clean method is hijacked to apply special treatment to invalid field inputs. For example adding extra options/classes to widgets.

`get_field_template`(*bound_field*, *template_name=None*)

Returns the field template to use when rendering a form field to HTML.

Preference of template selection:

1. Provided method argument *template_name*
2. Template from *field_template_overrides* selected by field name
3. Template from *field_template_overrides* selected by field class
4. Form class property *field_template*
5. Globally defined default template from *defaults.LAYOUT_FIELD_TEMPLATE*

Parameters

- `bound_field` – *BoundField* instance to select a template for.
- `template_name` – Optional template to use instead of other configurations.

Returns

Template name to use when rendering the form field.

`get_field_container_css_class`(*bound_field*)

Returns the container CSS class to use when rendering a field template.

By default, returns the Form class property *field_container_css_class*.

Parameters

`bound_field` – *BoundField* instance to return CSS class for.

Returns

A CSS class string.

`get_field_label_css_class`(*bound_field*)

Returns the optional label CSS class to use when rendering a field template.

By default, returns the Form class property *field_label_css_class*. If the field has errors and the Form class property *field_label_invalid_css_class* is defined, its value is appended to the CSS class.

Parameters

`bound_field` – *BoundField* instance to return CSS class for.

Returns

A CSS class string or *None*

`get_field_context`(*bound_field*)

Returns the context which is used when rendering a form field to HTML.

The generated template context will contain the following variables:

- form: *Form* instance
- field: *BoundField* instance of the field
- field_id: Field ID to use in *<label for="..">*

- field_name: Name of the form field to render
- errors: *ErrorList* instance with errors of the field
- required: Boolean flag to signal if the field is required or not
- label: The label text of the field
- label_css_class: The optional label CSS class, might be *None*
- help_text: Optional help text for the form field. Might be *None*
- container_css_class: The CSS class for the field container.
- widget_class_name: Lowercased version of the widget class name (e.g. 'textinput')
- widget_input_type: *input_type* property of the widget instance, falls back to *widget_class_name* if not available.

Returns
Template context for field rendering.

`apply_widget_options`(*field_name*)

Applies additional widget options like changing the input type of DateInput and TimeInput to "date" / "time" to enable Browser date pickers or other attributes/properties.

`apply_widget_template`(*field_name*)

Applies widget template overrides if available.

The method uses the *get_widget_template* method to determine if the widget template should be exchanged. If a template is available, the template_name property of the widget instance is updated.

Parameters
`field_name` – A field name of the form.

`get_widget_template`(*field_name*, *field*)

Returns the optional widget template to use when rendering the widget for a form field.

Preference of template selection:

1. Template from *widget_template_overrides* selected by field name
2. Template from *widget_template_overrides* selected by widget class

By default, returns *None* which means "use Django's default widget template".

Parameters

- `field_name` – The field name to select a widget template for.
- `field` – *Field* instance to return a widget template.

Returns
Template name to use when rendering the widget or *None*

`apply_widget_css_class`(*field_name*)

Applies CSS classes to widgets if available.

The method uses the *get_widget_css_class* method to determine if the widget CSS class should be changed. If a CSS class is returned, it is appended to the current value of the class property of the widget instance.

Parameters
`field_name` – A field name of the form.

`get_widget_css_class`(*field_name*, *field*)

Returns the optional widget CSS class to use when rendering the form's field widget.

By default, returns *None* which means "no CSS class / no change".

Parameters

- **field_name** – The field name of the corresponding field for the widget.
- **field** – *Field* instance to return CSS class for.

Returns
A CSS class string or *None*

`apply_widget_invalid_options`(*field_name*)

Applies additional widget options for an invalid field.

This method is called when there is some error on a field to apply additional options on its widget. It does the following:

- Sets the aria-invalid property of the widget for accessibility.
- Adds an invalid CSS class, which is determined by the returned value of *get_widget_invalid_css_class* method. If a CSS class is returned, it is appended to the current value of the class property of the widget.

Parameters
field_name – A field name of the form.

`get_widget_invalid_css_class`(*field_name*, *field*)

Returns the optional widget CSS class to append when rendering the form's field widget in case of error.

By default, returns *None* which means "no CSS class / no change".

Parameters

- **field_name** – The field name of the corresponding field for the widget.
- **field** – *Field* instance to return CSS class for.

Returns
A CSS class string or *None*

1.6.2 Fieldsets

`class tapeforms.fieldsets.TapeformFieldset`(*form*, *fields=None*, *exclude=None*, *primary=False*, *template=None*, *extra=None*)

Bases: *TapeformLayoutMixin*, `object`

Class to render a subset of a form's fields. From a template perspective, a fieldset looks quite similar to a form (and can use the same template tag to render: `form`.

`__init__`(*form*, *fields=None*, *exclude=None*, *primary=False*, *template=None*, *extra=None*)

Initializes a fieldset instance to be used like a form in a template.

Just like in ModelForm Meta, you have to provide at least a list of fields to render in this fieldset or a list of fields to exclude. If you provide both, exclusions have a higher priority.

Parameters

- **form** – The form instance to take fields from.

- **fields** – A list of visible field names to include in this fieldset.
- **exclude** – A list of visible fields to _not_ include in this fieldset.
- **primary** – If the fieldset is *primary*, this fieldset is responsible for rendering the hidden fields and non field errors.
- **template** – You can provide an alternative layout template to use.
- **extra** – This argument is carried around with the fieldset and is also available in the template. Useful to pass some special arguments for rendering around (like a fieldset headline.

Returns
A configured fieldset instance.

`hidden_fields()`

Returns the hidden fields of the form for rendering of the fieldset is marked as the primary fieldset.

Returns
List of bound field instances or empty tuple.

`non_field_errors()`

Returns all non-field errors of the form for rendering of the fieldset is marked as the primary fieldset.

Returns
ErrorList instance with non field errors or empty ErrorList.

`visible_fields()`

Returns the reduced set of visible fields to output from the form.

This method respects the provided `fields` configuration _and_ exlcudes all fields from the `exclude` configuration.

If no `fields` where provided when configuring this fieldset, all visible fields minus the excluded fields will be returned.

Returns
List of bound field instances or empty tuple.

`class tapeforms.fieldsets.TapeformFieldsetsMixin`

Bases: `object`

Mixin to generate fieldsets based on the *fieldsets* property of a `TapeformFieldsetsMixin` enabled form.

`fieldset_class`

Default fieldset class to use when instantiating a fieldset.

alias of *TapeformFieldset*

`fieldsets = None`

List/tuple of kwargs as *dict* ` to generate fieldsets for.

`get_fieldset_class(**fieldset_kwargs)`

Returns the fieldset class to use when generating the fieldset using the passed fieldset kwargs.

Parameters
fieldset_kwargs – `dict` with the fieldset config from `fieldsets`

Returns
Class to use when instantiating the fieldset.

get_fieldset(***fieldset_kwargs*)

Returns a fieldset instance for the passed `fieldset_kwargs`.

Parameters
fieldset_kwargs – `dict` with the fieldset config from `fieldsets`

Returns
Fieldset instance

get_fieldsets(*fieldsets=None*)

This method returns a generator which yields fieldset instances.

The method uses the optional fieldsets argument to generate fieldsets for. If no fieldsets argument is passed, the class property `fieldsets` is used.

When generating the fieldsets, the method ensures that at least one fielset will be the primary fieldset which is responsible for rendering the non field errors and hidden fields.

Parameters
fieldsets – Alternative set of fieldset kwargs. If passed this set is prevered of the `fieldsets` property of the form.

Returns
generator which yields fieldset instances.

1.6.3 Templatetags

tapeforms.templatetags.tapeforms.**form**(*context*, *form*, ***kwargs*)

The *form* template tag will render a tape-form enabled form using the template provided by *get_layout_template* method of the form using the context generated by *get_layout_context* method of the form.

Usage:

```
{% load tapeforms %}
{% form my_form %}
```

You can override the used layout template using the keyword argument *using*:

```
{% load tapeforms %}
{% form my_form using='other_form_layout_template.html' %}
```

Parameters
form – The Django form to render.

Returns
Rendered form (errors + hidden fields + fields) as HTML.

tapeforms.templatetags.tapeforms.**formfield**(*context*, *bound_field*, ***kwargs*)

The *formfield* template tag will render a form field of a tape-form enabled form using the template provided by *get_field_template* method of the form together with the context generated by *get_field_context* method of the form.

Usage:

```
{% load tapeforms %}
{% formfield my_form.my_field %}
```

You can override the used field template using the keyword argument *using*:

```
{% load tapeforms %}
{% formfield my_form.my_field using='other_field_template.html' %}
```

Parameters
bound_field – The *BoundField* from a Django form to render.

Returns
Rendered field (label + widget + other stuff) as HTML.

1.6.4 Contrib

Bootstrap mixin

class tapeforms.contrib.bootstrap.**Bootstrap4TapeformMixin**(**args*, ***kwargs*)

Bases: *TapeformMixin*

Tapeform Mixin to render Bootstrap v4 compatible forms. (using the template tags provided by *tapeforms*).

layout_template = 'tapeforms/layouts/bootstrap.html'

Use a special layout template for Bootstrap compatible forms.

field_template = 'tapeforms/fields/bootstrap.html'

Use a special field template for Bootstrap compatible forms.

field_container_css_class = 'form-group'

All form field containers need a CSS class "form-group".

widget_css_class = 'form-control'

Almost all widgets need a CSS class "form-control".

widget_invalid_css_class = 'is-invalid'

Use a special class to invalid field's widget.

widget_template_overrides = {<class 'django.forms.widgets.SelectDateWidget'>: 'tapeforms/widgets/bootstrap_multiwidget.html', <class 'django.forms.widgets.SplitDateTimeWidget'>: 'tapeforms/widgets/bootstrap_multiwidget.html', <class 'django.forms.widgets.RadioSelect'>: 'tapeforms/widgets/bootstrap_multipleinput.html', <class 'django.forms.widgets.CheckboxSelectMultiple'>: 'tapeforms/widgets/bootstrap_multipleinput.html'}

Widgets with multiple inputs require some extra care (don't use ul, etc.)

get_field_container_css_class(*bound_field*)

Returns "form-check" if widget is CheckboxInput in addition of the default value from the form property ("form-group") - which is returned for all other fields.

get_field_label_css_class(*bound_field*)

Returns "form-check-label" if widget is CheckboxInput. For all other fields, no CSS class is added.

get_widget_css_class(*field_name*, *field*)

Returns "form-check-input" if input widget is checkable, or "form-control-file" if widget is FileInput. For all other fields, returns the default value from the form property ("form-control").

class tapeforms.contrib.bootstrap.**Bootstrap5TapeformMixin**(**args*, ***kwargs*)

Bases: *Bootstrap4TapeformMixin*

Tapeform Mixin to render Bootstrap v5 compatible forms. (using the template tags provided by *tapeforms*).

field_container_css_class = 'mb-3'

Apply the CSS class "mb-3" to add spacing between the form fields.

field_label_css_class = 'form-label'

Almost all labels need a CSS class "form-label".

widget_template_overrides = {<class 'django.forms.widgets.SelectDateWidget'>: 'tapeforms/widgets/bootstrap5_multiwidget.html', <class 'django.forms.widgets.SplitDateTimeWidget'>: 'tapeforms/widgets/bootstrap5_multiwidget.html', <class 'django.forms.widgets.RadioSelect'>: 'tapeforms/widgets/bootstrap_multipleinput.html', <class 'django.forms.widgets.CheckboxSelectMultiple'>: 'tapeforms/widgets/bootstrap_multipleinput.html'}

Widgets with multiple inputs require some extra care (don't use ul, etc.)

get_widget_css_class(*field_name*, *field*)

Returns "form-check-input" if input widget is checkable, or "form-select" if widget is Select or a subclass. For all other fields, returns the default value from the form property ("form-control").

tapeforms.contrib.bootstrap.**BootstrapTapeformMixin**

This alias is for backward compatibility only. It could be deprecated and removed at some time, you should use *Bootstrap4TapeformMixin* or *Bootstrap5TapeformMixin* instead.

Foundation mixin

class tapeforms.contrib.foundation.**FoundationTapeformMixin**(**args*, ***kwargs*)

Bases: *TapeformMixin*

Tapeform Mixin to render Foundation compatible forms. (using the template tags provided by *tapeforms*).

layout_template = 'tapeforms/layouts/foundation.html'

Use a special layout template for Foundation compatible forms.

field_template = 'tapeforms/fields/foundation.html'

Use a special field template for Foundation compatible forms.

field_label_invalid_css_class = 'is-invalid-label'

Use a special class to invalid field's label.

widget_invalid_css_class = 'is-invalid-input'

Use a special class to invalid field's widget.

widget_template_overrides = {<class 'django.forms.widgets.RadioSelect'>: 'tapeforms/widgets/foundation_multipleinput.html', <class 'django.forms.widgets.CheckboxSelectMultiple'>: 'tapeforms/widgets/foundation_multipleinput.html'}

Widgets with multiple inputs require some extra care (don't use ul, etc.)

get_field_template(*bound_field*, *template_name=None*)

Uses a special field template for widget with multiple inputs. It only applies if no other template than the default one has been defined.

1.7 Changelog

1.7.1 1.2.0 - 2023-02-25

- Replace removed '.form-row' by grid system in Bootstrap v5 multiple widgets
- Add '.is-invalid' to multiple widget container in Bootstrap mixins to display the '.invalid-feedback' elements
- Drop "official" support for Python < 3.8, add support for Django 4.1

1.7.2 1.1.0 - 2021-09-13

- Rename Bootstrap mixin to `Bootstrap4TapeformMixin`, an alias is kept for backward compatibility but it will be deprecated and removed at some time
- Remove 'small' CSS class from the help text in Boostrap mixin
- Use 'form-field-' as prefix for container CSS class instead of the value returned by `get_field_container_css_class`
- Add a new `Bootstrap5TapeformMixin` mixin for Bootstrap v5
- Add 'form-group' CSS class to checkbox container in Bootstrap mixin

1.7.3 1.0.1 - 2021-04-28

- Add support for Django 3

1.7.4 1.0.0 - 2021-03-15

- Add support for Python 3.7 and 3.8
- Remove support for Django 2.0 and 2.1
- Fix duplicated class issue in bootstrap template

1.7.5 0.2.1 - 2019-02-06

- Change the way extra options are applied to invalid widgets

1.7.6 0.2.0 - 2018-12-12

- Add support for Foundation flavored forms
- Improve support for applying css classes to invalid fields

1.7.7 0.1.1 - 2018-08-27

- Fix styling of invalid inputs in Bootstrap forms by adding the correct css class

1.7.8 0.1.0 - 2018-08-17

- Add `as_tapeform` method to Forms to render forms without the need to call the `form` template tag
- Add hook to update widget options `apply_widget_options`
- DateInput, TimeInput and SplitDateTimeWidget get a proper input type to activate Browser's datepicker.
- Fix invalid help text display if html tags are part of the help text

1.7.9 0.0.4 - 2018-03-22

- Bugfix release (invalid template path)

1.7.10 0.0.3 - 2018-03-22

- Improved Bootstrap 4 support

1.7.11 0.0.2 - 2018-03-11

- Allow ModelForms in *form* template tag.

1.7.12 0.0.1 - 2018-02-27

- Initial release of *django-tapeforms*

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

Symbols

A

B

F

G