django-formfield Documentation

Release 0.4

Jose Soares

Apr 19, 2017

Contents

Contents:

CHAPTER 1

Installation

Installation is easy using `pip` or `easy_install`.

```
pip install django-formfield
```

or

```
easy_install django-formfield
```

Add to installed apps

```
INSTALLED_APPS = (
    ...
    formfield,
    ...
)
```

CHAPTER 2

Getting Started

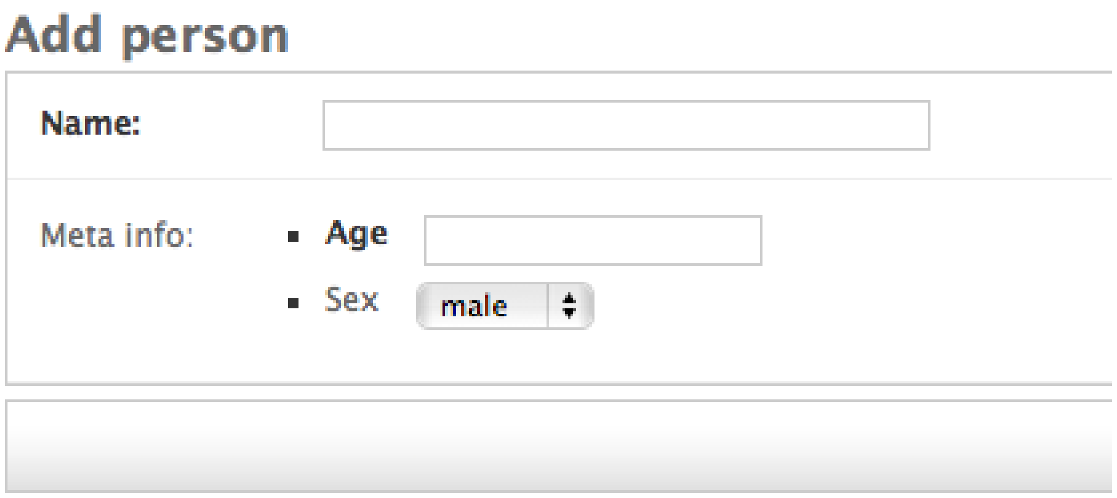
django-formfield is a form field that accepts a django form as its first argument, and validates as well as render's each form field as expected. Yes a form within a form, *within a dream*? There are two types of fields available, *FormField* and *ModelFormField*. For *ModelFormField* the data is stored in json. For *FormField* data is simply returned as a python dictionary (form.cleaned_data)

Example

```
from django.db import models
from django import forms

from formfield import ModelFormField

class PersonMetaForm(forms.Form):
    age = forms.IntegerField()
    sex = forms.ChoiceField(choices=((1, 'male'), (2, 'female')), required=False)


class Person(models.Model):
    name = CharField(max_length=200)

    meta_info = ModelFormField(form=PersonMetaForm)
```

Note: Changed in 0.3: You must supply the kwarg argument *form* to *ModelFormField*

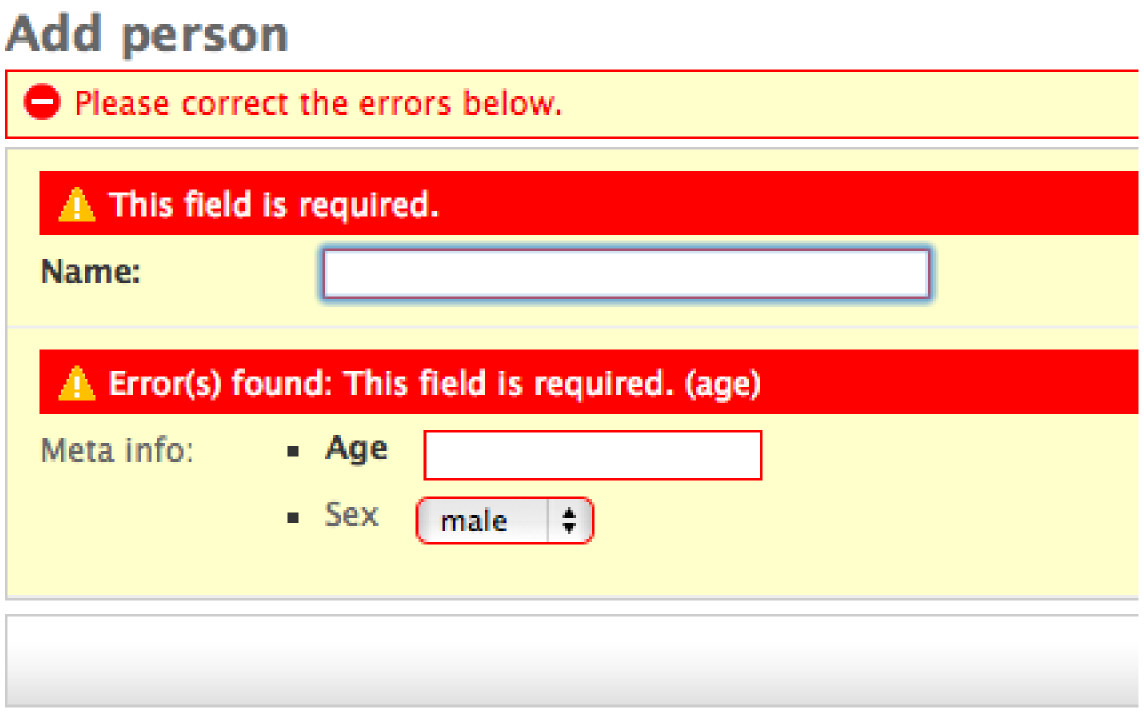
Which will result in something like this (using the admin)


Add person

Name:

Meta info:
- Age
- Sex male

The *ModelFormField* is automatically set to *null=True*, *blank=True*, this is because validation is done on the inner form. As a result you will see something like the following if we hit save on the change form:


Add person

Please correct the errors below.

This field is required.

Name:

Error(s) found: This field is required. (age)

Meta info:
- Age
- Sex male

If we supply the change for valid data you should get a python dictionary when retrieving the data:

```
>>> person = Person.objects.get(pk=1)
>>> person.meta_info
{u'age': 12, u'sex': u'1'}
```

The form is the only thing forcing valid input, behind the scenes the data is being serialized into json. Therefore on the python level we can supply meta_info any valid json::

```
>>> from sample_app.models import Person
>>> data = {'some': 'thing', 'is': 'wrong', 'here': 'help!'}
>>> p = Person.objects.create(name="Joan", meta_info=data)
>>> p.meta_info
{'is': 'wrong', 'some': 'thing', 'here': 'help!'}
```

Note: If the form field is being made available via a change form, such as the admin, any unexpected value will be overridden by what the form returns . For example, the *PersonMetaForm* above only expects *age* and *sex*, so none of the values above (‘is’, ‘some’ and ‘here’) match and will be overridden when the form submitted.

We can however, make the field hidden or readonly and use it to supply any valid json, but its not really the intension of this app.

CHAPTER 3

Reference

API

FormField

A form field which accepts a *django.forms.Form*, a string in the format 'path.to.Class' or a callable that returns a *django.forms.Form* as the first argument. *FormFieldWidget* is used as this fields widget.

Example Usage:

```
from django import forms
from formfield import FormField

class OtherInfoForm(forms.Form):
    other_name = forms.CharField()


class MyForm(forms.Form):

    name = forms.CharField()

    other_info = FormField(form=OtherInfoForm)
```

Note: Changed in 0.3: You must supply the kwarg argument *form* to *ModelFormField*

Seems odd to have a form within a form yea? Its more useful when using it with a model where the data is serialized to json.

ModelFormField

A model form field which accepts a *django.forms.Form*, a string in the format 'path.to.Class' or a callable that returns a *django.forms.Form* as the first argument. *FormField* is used as form field.

Example Usage:

```
from django.db import models
from django import forms
from formfield import ModelFormField

class MetaDataForm(forms.Form):
    alias = forms.CharField(required=False)
    phone = forms.CharField(required=False)
    email = forms.EmailField(required=False)


class Contact(models.Model):

    name = models.CharField(max_length=200)

    meta_data = ModelFormField(form=MetaDataForm)
```

FormFieldWidget

This is the widget used to render the output in a user friendly way. We added some methods to help render the output. The main method to override is the normal *format_output*, here is the default code:

```
ret = ['<ul class="formfield">']
for i, field in enumerate(self.fields):
    label = self.format_label(field, i)
    help_text = self.format_help_text(field, i)
    ret.append('<li>%s %s %s</li>' % (
        label, rendered_widgets[i], field.help_text and help_text))

ret.append('</ul>')
return six.u('').join(ret)
```

It simply wraps the entire form in a <ul> tag with a css class of *formfield*, you can override this for more control.

Extra methods

If you don't want to override the entire method you can override *format_label* and *format_help_text* as well. These methods accept to arguments, the bound field and a counter

FormFieldWidget.format_label

```
def format_label(self, field, counter):
    return '<label for="id_formfield_%s" %s>%s</label>' % (
        counter, field.field.required and 'class="required"', field.label)
```

FormFieldWidget.format_help_text

```
def format_help_text(self, field, counter):
    return '<p class="help">%s</p>' % field.help_text
```

CHAPTER 4

Contributing

- Source
- Issues

CHAPTER 5

Indices and tables

- genindex
- modindex
- search