formidable Documentation

Release 0.1.0

Guillaume Gérard Guillaume Camera

Jun 28, 2021

Contents

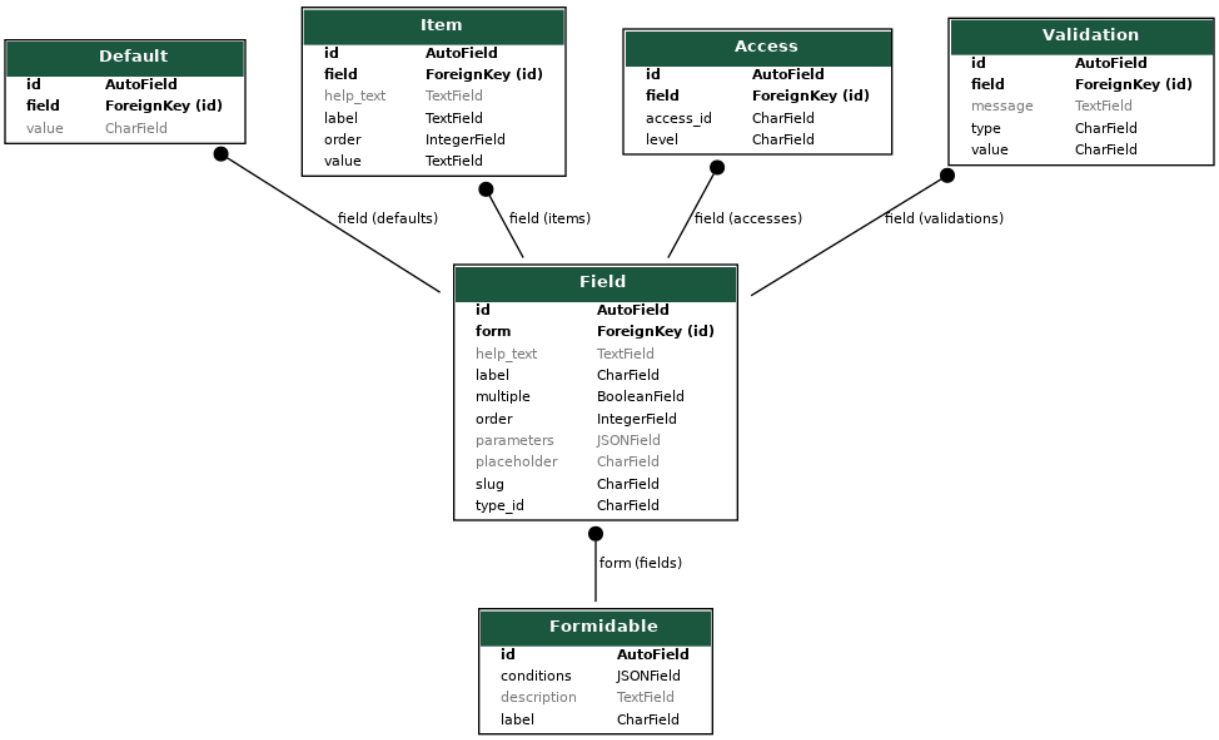
Contents:

CHAPTER 1

Introduction

`django-formidable` allows your users to create/edit custom Django forms. `django-formidable` provides a RESTful API, which can be used in conjunction with a front-end application. A number of endpoints are provided in order to enable building forms via the API.

1.1 Django model schema



CHAPTER 2

Install

2.1 Install the app

2.1.1 From PyPI

```
$ pip install django-formidable
```

2.1.2 From Github

You can also install `django-formidable` via GitHub:

```
pip install git+https://github.com/peopledoc/django-formidable.git
```

2.2 Configure the app

Before you can use the app, some things need to be configured in order to get it fully operational. `django-formidable` has the ability to handle different roles and accesses on a per form basis. This is useful when you have multiple types of user accessing the same form. If you don't need multiple roles, just create a single unique role, this will be enough.

2.2.1 Configure access-rights

First of all, you need to declare all available roles inside your application. To do this, create an `formidable.accesses.AccessObject` per role needed.

```
from formidable.accesses import AccessObject

jedi = AccessObject(id='jedi', label='Jedi')
padawan = AccessObject(id='padawan', label='Padawan')
```

Once your roles are defined, you will need to create a function to return them, in your projects (for the purposes of this example, we're assuming the function will be created in the module `yourproject.access_rights`):

```
def get_access_rights():
    return [jedi, padawan]
```

The main idea is to create a function which can be called by `django-formidable` to get the declared roles you defined previously. To tell `django-formidable` where the function is located, you need to add `FORMIDABLE_ACCESS_RIGHTS_LOADER` to your settings:

```
FORMIDABLE_ACCESS_RIGHTS_LOADER = 'yourproject.access_rights.get_access_rights'
```

2.2.2 Fetch the context

When the content of a contextualised form are required, e.g. to render it in a JavaScript front-end, `django-formidable` needs to know which context to fetch in order to render the correct fields with the right permissions.

To do this, we'll need to write some code which will be called by `django-formidable`.

Let's assume your user model has a `user_type` attribute on it. In this case, you could write the following function:

```
def get_context(request, kwargs):
    return request.user.user_type
```

The `request` is a standard Django request, as found in any view. Likewise, `kwargs` is a standard dictionary of keyword arguments. Of course, the user type should correspond to the `id` of the AccessObject

Next fill the setting key `FORMIDABLE_CONTEXT_LOADER`:

```
FORMIDABLE_CONTEXT_LOADER = 'yourproject.access_rights.get_context'
```

2.2.3 Formidable's URLs

URLs are defined in `formidable.urls`. You can load them with the following line:

```
url(r'^api/', include('formidable.urls', namespace='formidable'))
```

2.2.4 URLs accesses

The `Formidable` views are built with `djangorestframework` and use the related permissions in order to handle accesses. So, you can write your own permissions with `djangorestframework` and use it in `django-formidable` views.

By default, a restrictive permission is applied on all API views if nothing is specified in django settings.

You can specified a list of permissions classes to all the API views by providing the configuration key `FORMIDABLE_DEFAULT_PERMISSION`:

```
FORMIDABLE_DEFAULT_PERMISSION = ['rest_framework.permissions.AllowAll']
```

There are two kinds of views,

1. views which allow to create or edit forms (handled by `FORMIDABLE_PERMISSION_BUILDER`) 2. views to use the form previously defined (handled by. `FORMIDABLE_PERMISSION_USING`).

You can provide any permissions you want.

2.2.5 CSRF

If you're dealing with logged-in users (you surely do), you're going to need to provide a CSRF Token when validating a creation or an edit form. If you don't provide it *or* if your CSRF is misconfigured, you'll receive a `403` error when trying to save your forms.

In order to do so, you'll have to use a code similar to this:

```
function setupCRSFToken(csrftoken) {
  $.ajaxSetup({
    beforeSend: function(xhr, settings) {
      if (!/^(GET|HEAD|OPTIONS|TRACE)$/.test(settings.type) && !this.crossDomain) {
        xhr.setRequestHeader("X-CSRFToken", csrftoken);
      }
    }
  });
}
```

Warning: you'll have to make sure that your CSRF configuration is properly set (middlewares, context managers, etc).

Then in your templates, those that'll have to display and handle the form editor, you'll have to call this function like this:

```
<script src="{% static "assets/csrftoken.js" %}"></script>
<script type="text/javascript">
    $(document).ready(function() {
        setupCRSFToken('{{ csrf_token }}');
    });
</script>
```

This way, every AJAX call coming from this template will provide a token that'll fit Django's (and Django REST Framework) requirements.

CHAPTER 3

Forms

The main purpose of this app is to handle Forms. Of course, the app provides an API to Create and Edit forms, but it's not the only option: `django-formidable` also provides a full python builder in order to create forms. `django-formidable` also provides a method to retrieve a standard django form class which can then be used just like an ordinary django form.

3.1 Formidable object

The main class is *formidable.models.Formidable*. This class is a classic django model which defines a representation of a dynamic form.

class formidable.models.**Formidable**(*id*, *label*, *description*, *conditions*)

exception DoesNotExist

exception MultipleObjectsReturned

static from_json(*definition_schema*, ***kwargs*)

Proxy static method to create an instance of `Formidable` more easily with a given `definition_schema`.

Params definition_schema Schema in JSON/dict

```
>>> Formidable.from_json(definition_schema)
<Formidable: Formidable object>
```

get_django_form_class(*role=None*, *field_factory=None*)

Return the django form class associated with the formidable definition. If no role_id is provided all the fields are fetched with an `EDITABLE` access-right. :params role: Fetch defined access for the specified role. :params field_factory: Instance of Custom field factory if needed. :params field_map: Custom Field Builder used by the field_factory.

get_next_field_order()

Get the next order to set on the field to arrive. Try to avoid using this method for performance reasons.

This is the main object which is used to create or edit dynamic forms through the RESTful API or directly in Python/Django.

3.1.1 Django form class

One of the main feature is to provide a standard django form class built from the definition stored as Formidable object. The django form class is accessible throught the *formidable.models.Formidable.get_django_form_class()*.

```
>>> formidable = Formidable.objects.get(pk=42)
>>> form_class = formidable.get_django_form_class()
```

This form class can be manipulated as all django form class, you can build an instance to validate data:

```
>>> form = form_class(data={'first_name': 'Obiwan'})
>>> form.is_valid()
False
>>> form.errors
{'last_name': ['This field is required.']}
>>> form = form_class(data={'first_name': 'Obiwan', 'last_name': 'Kenobi'})
>>> form.is_valid()
True
```

Or to render it:

```
{{ form.as_p }}
```

When a standard mechanism is implemented, you have a method to custom the final objec we get. `django-formidable` provides a way in order to custom the form class you get.

Each kind of field is built with an associated FieldBuilder:

slug	Field / Widgets	FieldBuilder
text	CharField / TextInput	`formidable.forms.field_builder.TextFieldBuilder`
paragraph	CharField / TextArea	`formidable.forms.field_builder.ParagraphFieldBuilder`
dropdown	ChoiceField / Select	`formidable.forms.field_builder.DropdownFieldBuilder`
checkbox	ChoiceField / CheckboxInput	`formidable.forms.field_builder.CheckboxFieldBuilder`
radios	ChoiceField / RadioSelect	`formidable.forms.field_builder.RadioFieldBuilder`
checkboxes	ChoiceField / CheckboxSelectMultiple	`formidable.forms.field_builder.CheckboxesFieldBuilder`
email	EmailField / TextInput	`formidable.forms.field_builder.EmailFieldBuilder`
date	DateField	`formidable.forms.field_builder.DateFieldBuilder`
number	IntegerField	`formidable.forms.field_builder.IntegerFieldBuilder`

So, as describe in django document (https://docs.djangoproject.com/en/1.9/topics/forms/media/#assets-as-a-static-definition), if you want add a CalendarWidget on the date field on your form, you can write your own field builder.

```
from django import forms

from formidable.forms.field_builder import DateFieldBuilder, FormFieldFactory

class CalendarWidget(forms.TextInput):

    class Media:
        css = {
            'all': ('pretty.css',)
        }
        js = ('animations.js', 'actions.js')


class CalendarDateFieldBuilder(DateFieldBuilder):
    widget_class = CalendarWidget


class MyFormFieldFactory(FormFieldFactory):
    field_map = FormFieldFactory.field_map.copy()
    field_map['date'] = CalendarDateFieldBuilder
```

With this definition you can call:

```
>>> formidable.get_django_form_class(field_factory=MyFormFieldFactory)
```

3.2 Roles and access-rights

3.2.1 Roles

One of the main features of `formidable` is to set up different access-rights for the same form. This way, you can create a form with certain fields that are only accessible to a specific group of users, for example.

For the moment, `formidable` is not designed to work without roles, so even if you don't need to handle multiple roles or access-rights inside your application, you will still have to define a default role for `formidable` to work properly.

All roles must be declared through a `formidable.accesses.AccessObject` instance. This class must be instantiated with an `id` and a `label`. The `id` has to be unique, it's up to you to maintain this constraint. The `label` serves as a human readable value. You can set this to any string you like.

```
from formidable.accesses import AccessObject

padawan = AccessObject(id='padawan', label='Padawan')
jedi = AccessObject(id='jedi', label='Jedi')
sith = AccessObject(id='sith', label='Bad Guy')
```

`django-formidable` needs to know how to get all declared instances. To do so, you will need to create a function which returns the correct instances:

```
def get_accesses():
    return [padawan, jedi, sith]
```

Once this function is defined, you will need to fill the settings key `FORMIDABLE_ACCESS_RIGHTS_LOADER`.

```
FORMIDABLE_ACCESS_RIGHTS_LOADER = 'myapp.accesses.get_accesses'
```

Once this is done, `django-formidable` will know which roles have been defined, so it can create or check access-rights as necessary.

3.2.2 Fetch context

Occasionally, `django-formidable` will require access to the web request's context, e.g. to find out which kind of user is accessing the current form.

For this reason, you must define a function to fetch the context of the current request. The function takes as parameters the request object of the view (`self.request`) and the view kwargs (`self.kwargs`).

The function must return an access id which is defined in one of the AccessObject instances returned by the method configured in `FORMIDABLE_ACCESS_RIGHTS_LOADER`.

If the user's role is defined as an attribute, you can just return it directly:

```
def fetch_context(request, kwargs):
    return request.user.role
```

Then, set `FORMIDABLE_CONTEXT_LOADER` in your settings:

```
FORMIDABLE_CONTEXT_LOADER = myapp.accesses.fetch_context
```

3.2.3 Available access-rights

For each field of a form, and for each role you have defined, you can define a specific access-right. There are four different available access-rights:

- `EDITABLE`, the user may fill-in the field but there is no obligation to do so.
- `REQUIRED`, the user must fill-in the field in order to submit the form.
- `READONLY`, this will render the field as disabled, allowing the user to view but not modify its contents.
- `HIDDEN`, the field will not be available to the user, preventing the user from either viewing or modifying its contents.

All the value are defined in `formidable.constants`

3.3 Conditions

Important: As of `1.4.0`, it is allowed to have several conditional display rules that target a common field. In case of "conflict" between these rules, priority goes to the *display*, rather than the *hide* action.

e.g.:

- Rule 1 says: "if checkbox-1 is checked, then display field X"
- Rule 2 says: "if checkbox-2 is checked, then display field X and Y"

if only checkbox-1 is checked, field X will be displayed, even if checkbox-2 is unchecked, and vice-versa. If both are checked, fields X and Y will be displayed. If none is checked, fields X and Y will be hidden.

3.3.1 Types for conditional rules

At this moment, we can guarantee only the support of the checkboxes and dropdown lists, but normally you could use it for any type you want.

Also, you could specify types allowed for the conditions using the settings variable `FORMIDABLE_CONDITION_FIELDS_ALLOWED_TYPES` By default formidable will accept any type.

```
FORMIDABLE_CONDITION_FIELDS_ALLOWED_TYPES = []  # formidable will allow any type for␣
↪the conditional rules
FORMIDABLE_CONDITION_FIELDS_ALLOWED_TYPES = ['checkbox']  # formidable will allow␣
↪checkboxes only
```

In case you try to configure a conditional display based on a field that has been excluded from the allowed types, you'll receive a `ValidationError` when trying to save the form.

Here is a list of all the available types:

```
available_types = [
    'title', 'helpText', 'fieldset', 'fieldsetTable', 'separation',
    'checkbox', 'checkboxes', 'dropdown', 'radios', 'radiosButtons',
    'text', 'paragraph', 'file', 'date', 'email', 'number'
]
```

3.4 Python builder

In some cases, you may want to build a formidable object without using the RESTful API (in tests for example). `django-formidable` provides a Python API in order to that. Take a look at `formidable.forms.fields` to discover all the fields that are available through this API.

The main class to use is `formidable.forms.FormidableForm`. Feel free to subclass this form and define your own form(s), just like any other django form.

For example, let's say we need to build a form with a first name, last name and a description. We can use `formidable.fields` to accomplish this. Lets consider using the different roles defined in the installation part, jedi and padawan.

```
from formidable.forms import FormidableForm
from formidable.forms import fields


class MySubscriptionForm(FormidableForm):

    first_name = fields.CharField(label='Your First Name')
    last_name = fields.CharField(label='Your Last Name')
    description = fields.TextField(
        label='Description',
        help_text='Tell us about yourself.'
    )
```

Attributes like `required` should not be used as these will depend on the context when the form is built. If you want to define a field as required, it will need to be required for a specific role through the `accesses` argument. This argument is a dictionary containing the various access-rights for each role. By default, if you don't specify any access-rights for a previously defined role, the field will be created as `EDITABLE`:

```
class MySubscriptionForm(FormidableForm):

    first_name = fields.CharField(
        label='Your First Name',
        accesses={'padawan': constants.REQUIRED, 'jedi': constants.READONLY}
    )
    last_name = fields.CharField(label='Your Last Name')
    description = fields.TextField(
        label='Description',
        help_text='Tell us about yourself.'
    )
```

When the form definition is complete, you can create a new *formidable.models.Formidable* object:

```
formidable = MySubscriptionForm.to_formidable(
    label='My Subscription Form',
    description='This form is for subscribing to the jedi order.')
```

This method will create the object in the database and return the complete instance:

```
>>> formidable.pk
42
```

You can also get the django form class from the formidable object:

```
>>> form_class = formidable.get_django_form_class(role='padawan')
```

For our 'padawan' role, the first_name is required:

```
>>> form = form_class(data={'last_name': 'Kenobi'})
>>> form.is_valid()
False
>>> form.errors
{'first_name': ['This field is required.']}
```

3.4.1 Available fields

3.4.2 Available Widgets

CHAPTER 4

Formidable Form JSON Schema specification

definitions				
• Access	Different contexts that helps to render a form			
	type	*object*		
	properties			
	• **description**	Help text of the access		
		type	*string*	
	• **id**	ID of the access		
		type	*string*	
	• **label**	Label of the access		
		type	*string*	
	• preview_as	How to display the preview, default is *FORM*. Values are *FORM*, *TABLE*		
		type	*string*	
		enum	FORM, TABLE	
• BuilderError	type	*object*		
	properties			
	• fields	Errors on fields (key => field; value => list of messages)		
		type	*object*	
	• non_field_errors	Errors on anything except fields (validations. . .)		
		type	*array*	
		items	type	*string*
• BuilderForm	allOf	#/definitions/Form		
		properties		
		• fields	List of fields ordered in the form	
			type	*array*
			items	#/definitions/Field
• Condition	Describe conditional display of a field, depending on the value of another field. e.g.: “display the field ‘what is you favorite Star Wars character?’ if the boolean field ‘Do you like Star Wars?’ is checked”.			
	type	*object*		
	properties			

Continued on next page

Table 1 – continued from previous page

	• **action**	Name of the action to do when the condition is true. e.g. "display the field" == `display_iff`		
		type	*string*	
		enum	display_iff	
	• **field_ids**	List of field slugs to show/hide depending on the conditions.		
		type	*array*	
		items	type	*string*
		minItems	1	
	• name	A user-provided name for the Condition		
		type	*string*	
	• **tests**	List of conditions to test.		
		type	*array*	
		items	#/definitions/ConditionTest	
		minItems	1	
• ConditionTest	Condition definition.			
	type	*object*		
	properties			
	• **field_id**	'slug' of the reference field for the comparison.		
		type	*string*	
	• **operator**	Comparison operator for the condition.		
		type	*string*	
		enum	eq	
	• **values**	List of the possible values that would return a "true" condition.		
		type	*array*	
		items		
• Field	Field in a form			
	type	*object*		
	properties			
	• **accesses**	List of accesses of the field with a level		
		type	*array*	
		items	#/definitions/FieldAccess	
	• defaults	Default values selected/inputed when the form is newly displayed		
		type	*array*	
		items	type	*string*
	• **description**	Description of the field		
		type	*string*	
	• **id**	ID of the field		
		type	*integer*	
	• items	Values available		
		type	*array*	
		items	#/definitions/Item	
	• **label**	Label of the field		
		type	*string*	
	• multiple	Is the field can have multiple values?		
		type	*boolean*	
	• placeholder	Placeholder of the field		
		type	*string*	
	• **slug**	Slug of the field (us as uniq identifier of the field on the form)		
		type	*string*	
	• **type_id**	Type of field (see Field types table)		
		type	*string*	

Continued on next page

Table 1 – continued from previous page

		enum	title, helpText, fieldset, fieldsetTable, separation, checkbox, checkboxes, dropdown, radios, radiosButtons, text, paragraph, file, date, email, number	
	• validations	List of validations of the field		
		type	*array*	
		items	#/definitions/FieldValidation	
• FieldAccess	The access is the way to use the field in the context			
	type	*object*		
	properties			
	• **access_id**	Access reference		
		type	*string*	
	• **level**	Level of this access for the field		
		type	*string*	
		enum	REQUIRED, EDITABLE, HIDDEN, READONLY	
• FieldValidation	This validation can only be performed on a single field			
	type	*object*		
	properties			
	• message	Error message if the validation is not verified		
		type	*string*	
	• **type**	Type of validation (see Validation types table)		
		type	*string*	
		enum	EQ, GT, GTE, IS_AGE_ABOVE, IS_AGE_UNDER, IS_DATE_IN_THE_FUTURE, IS_DATE_IN_THE_PAST, LT, LTE, MAXLENGTH, MINLENGTH, NEQ, REGEXP	
	• **value**	Value of the validation		
		type	*string*	
• Form	The central piece of this project			
	type	*object*		
	properties			
	• conditions	type	*array*	
		items	#/definitions/Condition	
	• **description**	Description of the form - can be empty		
		type	*string*	
	• **id**	ID of the form		
		type	*integer*	
	• **label**	Title of the form		
		type	*string*	
• InputError	Object that contains field errors as key with a list of string in value			
	type	*object*		
	properties			
	• __all__	Errors on anything except form's fields		
		type	*array*	
		items	type	*string*
• InputField	properties			
	• values	Values selected/inputed when the form is in edition mode		
		type	*array*	
		items	type	*string*

Continued on next page

Table 1 – continued from previous page

	allOf	#/definitions/Field		
• InputForm	allOf	#/definitions/Form		
		properties		
		• fields	List of fields ordered in the form	
			type	*array*
			items	#/definitions/InputField
• Item	Describe an item in a list			
	type	*object*		
	properties			
	• description	Description of the item		
		type	*string*	
	• **label**	Label of the item		
		type	*string*	
	• **value**	Value which defined the item		
		type	*string*	

Or, in raw JSON:

```
{
    "basePath": "/api",
    "definitions": {
        "Access": {
            "description": "Different contexts that helps to render a form",
            "properties": {
                "description": {
                    "description": "Help text of the access",
                    "type": "string"
                },
                "id": {
                    "description": "ID of the access",
                    "type": "string"
                },
                "label": {
                    "description": "Label of the access",
                    "type": "string"
                },
                "preview_as": {
                    "description": "How to display the preview, default is `FORM`.␣
↪Values are `FORM`, `TABLE`",
                    "enum": [
                        "FORM",
                        "TABLE"
                    ],
                    "type": "string"
                }
            },
            "required": [
                "id",
                "label",
                "description"
            ],
            "type": "object"
        },
        "BuilderError": {
            "properties": {
```

(continues on next page)

(continued from previous page)

```
                "fields": {
                    "description": "Errors on fields (key => field; value => list of␣
↪messages)",
                    "type": "object"
                },
                "non_field_errors": {
                    "description": "Errors on anything except fields (validations...)
↪",
                    "items": {
                        "type": "string"
                    },
                    "type": "array"
                }
            },
            "type": "object"
        },
        "BuilderForm": {
            "allOf": [
                {
                    "$ref": "#/definitions/Form"
                },
                {
                    "properties": {
                        "fields": {
                            "description": "List of fields ordered in the form",
                            "items": {
                                "$ref": "#/definitions/Field"
                            },
                            "type": "array"
                        }
                    }
                }
            ]
        },
        "Condition": {
            "description": "Describe conditional display of a field, depending on the␣
↪value of another field.\n\ne.g.: \"display the field 'what is you favorite Star␣
↪Wars character?' if the boolean field 'Do you like Star Wars?' is checked\".\n",
            "properties": {
                "action": {
                    "description": "Name of the action to do when the condition is␣
↪true. e.g. \"display the field\" == ``display_iff``",
                    "enum": [
                        "display_iff"
                    ],
                    "type": "string"
                },
                "field_ids": {
                    "description": "List of field slugs to show/hide depending on the␣
↪conditions.",
                    "items": {
                        "type": "string"
                    },
                    "minItems": 1,
                    "type": "array"
                },
                "name": {
```

(continues on next page)

(continued from previous page)

```
                    "description": "A user-provided name for the Condition",
                    "type": "string"
                },
                "tests": {
                    "description": "List of conditions to test.",
                    "items": {
                        "$ref": "#/definitions/ConditionTest"
                    },
                    "minItems": 1,
                    "type": "array"
                }
            },
            "required": [
                "field_ids",
                "action",
                "tests"
            ],
            "type": "object"
        },
        "ConditionTest": {
            "description": "Condition definition.",
            "properties": {
                "field_id": {
                    "description": "\\`slug\\` of the reference field for the␣
↪comparison.",
                    "type": "string"
                },
                "operator": {
                    "description": "Comparison operator for the condition.",
                    "enum": [
                        "eq"
                    ],
                    "type": "string"
                },
                "values": {
                    "description": "List of the possible values that would return a \
↪"true\" condition.",
                    "items": {},
                    "type": "array"
                }
            },
            "required": [
                "field_id",
                "operator",
                "values"
            ],
            "type": "object"
        },
        "Field": {
            "description": "Field in a form",
            "properties": {
                "accesses": {
                    "description": "List of accesses of the field with a level",
                    "items": {
                        "$ref": "#/definitions/FieldAccess"
                    },
                    "type": "array"
```

(continues on next page)

(continued from previous page)

```
                },
                "defaults": {
                    "description": "Default values selected/inputed when the form is␣
↪newly displayed",
                    "items": {
                        "type": "string"
                    },
                    "type": "array"
                },
                "description": {
                    "description": "Description of the field",
                    "type": "string"
                },
                "id": {
                    "description": "ID of the field",
                    "readOnly": true,
                    "type": "integer"
                },
                "items": {
                    "description": "Values available",
                    "items": {
                        "$ref": "#/definitions/Item"
                    },
                    "type": "array"
                },
                "label": {
                    "description": "Label of the field",
                    "type": "string"
                },
                "multiple": {
                    "description": "Is the field can have multiple values?",
                    "type": "boolean"
                },
                "placeholder": {
                    "description": "Placeholder of the field",
                    "type": "string"
                },
                "slug": {
                    "description": "Slug of the field (us as uniq identifier of the␣
↪field on the form)",
                    "type": "string"
                },
                "type_id": {
                    "description": "Type of field (see Field types table)",
                    "enum": [
                        "title",
                        "helpText",
                        "fieldset",
                        "fieldsetTable",
                        "separation",
                        "checkbox",
                        "checkboxes",
                        "dropdown",
                        "radios",
                        "radiosButtons",
                        "text",
                        "paragraph",
```

(continues on next page)

(continued from previous page)

```
                        "file",
                        "date",
                        "email",
                        "number"
                    ],
                    "type": "string"
                },
                "validations": {
                    "description": "List of validations of the field",
                    "items": {
                        "$ref": "#/definitions/FieldValidation"
                    },
                    "type": "array"
                }
            },
            "required": [
                "id",
                "slug",
                "label",
                "type_id",
                "description",
                "accesses"
            ],
            "type": "object"
        },
        "FieldAccess": {
            "description": "The access is the way to use the field in the context",
            "properties": {
                "access_id": {
                    "description": "Access reference",
                    "type": "string"
                },
                "level": {
                    "description": "Level of this access for the field",
                    "enum": [
                        "REQUIRED",
                        "EDITABLE",
                        "HIDDEN",
                        "READONLY"
                    ],
                    "type": "string"
                }
            },
            "required": [
                "access_id",
                "level"
            ],
            "type": "object"
        },
        "FieldValidation": {
            "description": "This validation can only be performed on a single field",
            "properties": {
                "message": {
                    "description": "Error message if the validation is not verified",
                    "type": "string"
                },
                "type": {
```

(continues on next page)

(continued from previous page)

```
                    "description": "Type of validation (see Validation types table)",
                    "enum": [
                        "EQ",
                        "GT",
                        "GTE",
                        "IS_AGE_ABOVE",
                        "IS_AGE_UNDER",
                        "IS_DATE_IN_THE_FUTURE",
                        "IS_DATE_IN_THE_PAST",
                        "LT",
                        "LTE",
                        "MAXLENGTH",
                        "MINLENGTH",
                        "NEQ",
                        "REGEXP"
                    ],
                    "type": "string"
                },
                "value": {
                    "description": "Value of the validation",
                    "type": "string"
                }
            },
            "required": [
                "type",
                "value"
            ],
            "type": "object"
        },
        "Form": {
            "description": "The central piece of this project",
            "properties": {
                "conditions": {
                    "items": {
                        "$ref": "#/definitions/Condition"
                    },
                    "type": "array"
                },
                "description": {
                    "description": "Description of the form - can be empty",
                    "type": "string"
                },
                "id": {
                    "description": "ID of the form",
                    "readOnly": true,
                    "type": "integer"
                },
                "label": {
                    "description": "Title of the form",
                    "type": "string"
                }
            },
            "required": [
                "id",
                "label",
                "description"
            ],
```

(continues on next page)

(continued from previous page)

```
            "type": "object"
        },
        "InputError": {
            "description": "Object that contains field errors as key with a list of
↪string in value",
            "properties": {
                "__all__": {
                    "description": "Errors on anything except form's fields",
                    "items": {
                        "type": "string"
                    },
                    "type": "array"
                }
            },
            "type": "object"
        },
        "InputField": {
            "allOf": [
                {
                    "$ref": "#/definitions/Field"
                }
            ],
            "properties": {
                "values": {
                    "description": "Values selected/inputed when the form is in
↪edition mode",
                    "items": {
                        "type": "string"
                    },
                    "type": "array"
                }
            }
        },
        "InputForm": {
            "allOf": [
                {
                    "$ref": "#/definitions/Form"
                },
                {
                    "properties": {
                        "fields": {
                            "description": "List of fields ordered in the form",
                            "items": {
                                "$ref": "#/definitions/InputField"
                            },
                            "type": "array"
                        }
                    }
                }
            ]
        },
        "Item": {
            "description": "Describe an item in a list",
            "properties": {
                "description": {
                    "description": "Description of the item",
                    "type": "string"
```

(continues on next page)

(continued from previous page)


```
                },
                "label": {
                    "description": "Label of the item",
                    "type": "string"
                },
                "value": {
                    "description": "Value which defined the item",
                    "type": "string"
                }
            },
            "required": [
                "label",
                "value"
            ],
            "type": "object"
        }
    },
    "host": "localhost:8000",
    "info": {
        "description": "django-formidable is a full django application which allows␣
↪you to create,\nedit, delete and use forms.\n\n##### Field types\n\nList of known␣
↪types available:\n\n| Type | Description | HTML Component |\n| ---- | ----------- |␣
↪-------------- |\n| title | Title | h2 |\n| helpText | Helptext | p |\n| fieldset |␣
↪Group of fields (iterable) | fieldset |\n| fieldsetTable | Group of fields display␣
↪as table (iterable) | table |\n| separation | Separator line (design only) | hr␣
↪|\n| checkbox | Checkbox alone | input type=checkbox |\n| checkboxes | Some␣
↪checkboxes, all checkable | input type=checkbox, all have the same name |\n|␣
↪dropdown | Dropdown with values | select |\n| radios | Some radios, only one is␣
↪selected at once | input type=radio |\n| radiosButtons | Some radios display as␣
↪toggle button | input type=radio |\n| text | Input for one line text | input␣
↪type=text |\n| paragraph | Input for multiline text | textarea |\n| file | Field to␣
↪select a local file to be uploaded | input type=file |\n| date | Input for a date␣
↪(datepicker with it, lang know by application parameter, validation by momentjs) |␣
↪input type=date |\n| email | Input for an email (validation by regexp) | input |\n|␣
↪number | Input for a number | input |\n\n##### Validations types\n\nList of␣
↪validations available by types:\n\n| Field | Validation type |\n| ----- | ----------
↪- |\n| text | MINLENGTH, MAXLENGTH, REGEXP |\n| paragraph | MINLENGTH, MAXLENGTH,␣
↪REGEXP |\n| date | GT, GTE, LT, LTE, EQ, NEQ, IS_AGE_ABOVE (>=), IS_AGE_UNDER (<),␣
↪IS_DATE_IN_THE_PAST (< today), IS_DATE_IN_THE_FUTURE (< today) |\n| number | GT,␣
↪GTE, LT, LTE, EQ, NEQ |\n",
        "title": "Formidable API",
        "version": "1.0.0"
    },
    "paths": {
        "/builder/accesses/": {
            "get": {
                "description": "List of accesses available.",
                "responses": {
                    "200": {
                        "description": "A list of accesses",
                        "schema": {
                            "items": {
                                "$ref": "#/definitions/Access"
                            },
                            "type": "array"
                        }
                    }
```


(continues on next page)

(continued from previous page)

```
                },
                "summary": "Get accesses"
            }
        },
        "/builder/forms/": {
            "post": {
                "description": "Create Form Description",
                "parameters": [
                    {
                        "in": "body",
                        "name": "form",
                        "required": true,
                        "schema": {
                            "$ref": "#/definitions/BuilderForm"
                        }
                    }
                ],
                "responses": {
                    "201": {
                        "description": "Newly created form",
                        "schema": {
                            "$ref": "#/definitions/BuilderForm"
                        }
                    },
                    "400": {
                        "description": "Unexpected error",
                        "schema": {
                            "$ref": "#/definitions/BuilderError"
                        }
                    }
                },
                "summary": "Create a new form"
            }
        },
        "/builder/forms/{id}/": {
            "get": {
                "parameters": [
                    {
                        "in": "path",
                        "name": "id",
                        "required": true,
                        "type": "integer"
                    }
                ],
                "responses": {
                    "200": {
                        "description": "Form",
                        "schema": {
                            "$ref": "#/definitions/BuilderForm"
                        }
                    }
                },
                "summary": "Retrieve a Form"
            },
            "put": {
                "parameters": [
                    {
```

(continues on next page)

(continued from previous page)


```
                        "in": "path",
                        "name": "id",
                        "required": true,
                        "type": "integer"
                    },
                    {
                        "in": "body",
                        "name": "form",
                        "required": true,
                        "schema": {
                            "$ref": "#/definitions/BuilderForm"
                        }
                    }
                ],
                "responses": {
                    "200": {
                        "description": "Form",
                        "schema": {
                            "$ref": "#/definitions/BuilderForm"
                        }
                    },
                    "400": {
                        "description": "Unexpected error",
                        "schema": {
                            "$ref": "#/definitions/BuilderError"
                        }
                    }
                },
                "summary": "Update a Form"
            }
        },
        "/forms/{id}/": {
            "get": {
                "parameters": [
                    {
                        "in": "path",
                        "name": "id",
                        "required": true,
                        "type": "integer"
                    }
                ],
                "responses": {
                    "200": {
                        "description": "A form",
                        "schema": {
                            "$ref": "#/definitions/BuilderForm"
                        }
                    }
                },
                "summary": "Get a contextualized form"
            }
        },
        "/forms/{id}/validate/": {
            "post": {
                "parameters": [
                    {
                        "in": "path",
```


(continues on next page)

(continued from previous page)

```
                        "name": "id",
                        "required": true,
                        "type": "integer"
                    }
                ],
                "responses": {
                    "204": {
                        "description": "Validation OK"
                    },
                    "400": {
                        "description": "Validation KO",
                        "schema": {
                            "$ref": "#/definitions/InputError"
                        }
                    }
                },
                "summary": "Validate user-data against a form schema."
            }
        }
    },
    "produces": [
        "application/json"
    ],
    "schemes": [
        "http"
    ],
    "swagger": "2.0"
}
```

CHAPTER 5

API Specifications

CHAPTER 6

Security setup

As any other web application, Django Formidable might be targeted by pirates who would try to inject SQL or malicious code through Javascript or any other XSS method.

6.1 How to secure your django-formidable installation

Add the following settings: `DJANGO_FORMIDABLE_SANITIZE_FUNCTION`. It should be a string that points at a function.

Important: We highly recommend to use bleach, with dedicated adjustments in order to make sure you're sanitizing your content in a proper way.

See bleach documentation for creating your own parameters when calling the `clean()` function.

6.2 Example

In your `settings.py`, add the following:

```
DJANGO_FORMIDABLE_SANITIZE_FUNCTION = "path.to.module.clean_func"
```

And then in the `path/to/module.py` module, add a function that would look like this:

```
import bleach

def clean_func(obj):
    """
    Sanitize API text content
    """
    return bleach.clean(obj, strip=True)
```

Warning: If you don't add this settings or if its value is not importable (typo, missing PYTHONPATH, etc.):

- an error log will be raised,
- django-formidable won't sanitize your contents for you.

6.3 Secured fields

- Form label & description,
- Field label, description (help text), defaults, placeholder.

CHAPTER 7

Callbacks

New in version 0.5.

Each time a formidable form is created or updated, the API views are able to call a function that can help you trigger actions. For example, you can use the `django.contrib.messages` to inform your current user that their form has been successfully saved or that a problem has occurred ; or send an email or ping an API, or... whatever you want.

By default, the view won't load and launch anything. In order to set a callback up, you'll need to give a value to any of the following variables:

- `FORMIDABLE_POST_CREATE_CALLBACK_SUCCESS`: callback to call when form creation is successful.
- `FORMIDABLE_POST_CREATE_CALLBACK_FAIL`: callback to call when form creation has failed.
- `FORMIDABLE_POST_UPDATE_CALLBACK_SUCCESS`: callback to call when form update is successful.
- `FORMIDABLE_POST_UPDATE_CALLBACK_FAIL`: callback to call when form update has failed.

7.1 The callback functions

A callback function is a function that accepts only one argument, the *request* object coming from the API View. It doesn't have to return anything, it can make multiple calls... it's up to you.

```
def callback_on_success(request):
    mail.send(request.user.email, 'All is fine')
```

Warning: the DRF request is not inherited from django core, `HTTPRequest`, and you should not assume they'll behave the same way. It shares some properties, so it may quack like a duck, but it's not a duck.

If you need the "true" HTTPRequest object, use `self.request._request`. That might be the case if you want to use the `django.contrib.messages`.

```
def callback_on_success(request):
    messages.info(request._request, "Your form is recorded")
```

7.2 Fails silently

At the moment, if your callback fails for some reason and throws an exception, the exception is logged and the error is skipped. We've decided not to re-raise the exception to avoid your database transaction to be rolled back and the form you've tried to save being lost. After all, it's not the users fault if the callback has failed, but the integrator's.

At some point, we may add a "fail" mode and re-raise the exception and allow the integrator to make sure that the DB transaction is either committed if everything is fine, or aborted if something bad happened in the callback function.

CHAPTER 8

Developer's documentation

8.1 Testing

8.1.1 Prerequisites

If you want to run the whole test suite, you'll need to have a working Postgresql server instance (preferrably the latest), with the `pg_virtualenv` tool available. On Debian, this executable is provided by the package `postgresql-common`.

If you don't have this tools in your toolbox, then... if you're doing a change that **impacts the performance records**, you won't be able to see or generate the diff, **so your branch tests would fail.**

Note: Postgresql driver is only available for Linux or MacOS.

8.1.2 Using tox

Tests are launched using tox. You may want to become proficient with this tool but the core command you need to know is:

```
$ tox
```

This will run all the test suite, combining

- all versions of Django supported,
- all the Python interpreters supported,
- all versions of Django REST Framework supported,
- on SQLite Databases + Postgresql Databases

Targeting a specific environment is done using:

```
$ tox -e django22-py38-drf310-sqlite
```

If you want to target a specific test, simply add its namespace after a double-dash `--`.

For example, the following will run `test_fields` test module using Django 2.2, Python 3.8 using a SQLite database:

```
$ tox -e django22-py38-drf310-sqlite -- tests.test_fields
```

And the following will run the same test class for all the supported environments:

```
$ tox -- tests.test_fields.RenderingFormatField
```

If somehow you've messed-up with your environment(s), you can still recreate it/them using:

```
$ tox -r  # RECREATE ALL THE THINGS
# recreate and run tests using django 2.2 + python 3.8 + DRF 3.10 + SQLite DB.
$ tox -re django22-py38-drf310-sqlite
```

8.1.3 Using py.test

You can also run tests with `py.test`.

You can install it with the following command:

```
$ pip install pytest{,-django}
# Optionally
$ pip install pytest-sugar
```

We've added a section in our `setup.cfg`, so you should be able to run tests simply with:

```
$ cd demo/
$ py.test
```

8.2 Swagger documentation update

If at any point you've changed something in the `docs/swagger/formidable.yml` file, you'll **have** to run the following to refresh at least the `docs/source/_static/specs/formidable.js` file that will be used in the *API Specifications* document.

Run the following to regenerate all the necessary statics:

```
$ tox -e swagger-statics
```

and commit the diffs in your PR.

CHAPTER 9

Translations

9.1 Crowdin support

deprecated:: 2.0.0

As of the version 2.0.0, the Django Formidable project doesn't handle any translatable string anymore.

CHAPTER 10

Deprecation timeline

10.1 From 7.0.0 to x.y.z

10.1.1 Django REST Framework versions

New in version x.y.z: Add/Confirm support of Django REST Framework 3.11

10.2 From 6.1.0 to 7.0.0

New in version <7.0.0>: The *description* field in the `Formidable` model class would now allow empty values.

10.3 From 5.0.0 to 6.0.0

10.3.1 Python versions

Deprecated since version 6.0.0: Drop support for Python 3.5

10.4 From 4.0.1 to 5.0.0

10.4.1 Django versions

Deprecated since version 5.0.0: Drop support for Django 1.11

10.4.2 Django REST Framework versions

Deprecated since version 5.0.0: Drop support for Django Rest Framework 3.8

10.5 From 3.3.0 to 4.0.0

Jan 8th, 2020.

10.5.1 Python versions

Deprecated since version 4.0.0: Drop support for Python 2.7 (EOL is January 1st, 2020)

10.5.2 Configuration option

New in version 4.0.0: Added support for XSS prevention using the `DJANGO_FORMIDABLE_SANITIZE_FUNCTION` settings. See the security Documentation for more information.

10.6 From 3.2.0 to 3.3.0

10.6.1 Django versions

New in version 3.3.0: Added support for Django 2.2. Django Formidable should probably work on Django 2.0 and 2.1, but it's not in our test suite. We've decided to skip those versions because of their short-term support.

10.6.2 Python versions

New in version 3.3.0: Added support for Python 3.7 and 3.8

10.7 From 3.1.0 to 3.2.0

November 7th, 2019

10.7.1 Django versions

Deprecated since version 3.2.0: Drop support for Django 1.10 (EOL was in December 2nd, 2017)

10.8 From 3.0.1 to 3.1.0

June 3rd, 2019

10.8.1 Django REST Framework versions

New in version 3.1.0: Support for Django REST Framework on all versions up to the 3.9 series.

10.9 From 2.1.2 to 3.0.0

October 31st, 2018

10.9.1 Django REST Framework versions

Deprecated since version 3.0.0: Support for Django REST Framework stricly greater than 3.8. The 3.9 series has introduced an incompatibility with `django-formidable`.

10.10 From 1.7.0 to 2.0.0

(end of May 2018)

10.10.1 Django versions

Deprecated since version 2.0.0: Support for Django 1.8 & 1.9.

10.10.2 Crowdin

Deprecated since version 2.0.0: The Django Formidable project doesn't handle any translatable string anymore.

10.11 From 1.3.0 to 1.4.0

10.11.1 Validation endpoint

Deprecated since version 1.4.0: Validation endpoint for **user data** doesn't allow GET method anymore.

10.12 From 0.15 to 1.0.0

(September 2017)

10.12.1 Form Presets

Deprecated since version 1.0.0: Form presets will be deprecated in favor of Field validation rules. If needed, you'll have to convert your existing Presets to Field validations, because Presets data will be destroyed using a table deletion.

10.12.2 Django Rest Framework version

Deprecated since version 1.0.0: DRF 3.3 support will be deprecated. We recommend to use the latest to date (3.6.4).

10.13 From 0.11.1 to 0.12.0

Deprecated since version 0.12.0: Python 3.4 support has been dropped.

10.14 From 0.8.2 to 0.9

Deprecated since version 0.9: Python 3.3 support has been dropped.

CHAPTER 11

External Field Plugin Mechanism

New in version 3.0.0.

We've included a mechanism to add your own fields to the collection of available fields in `django-formidable`.

It'll be possible to:

- define a new form using this new type of field,
- store their definition and parameters in a Formidable object instance (and thus, in the database),
- using this form definition, validate the end-user data when filling this form against your field business logic mechanism.

For the sake of the example, let's say you want to add a "Color Picker" field in django-formidable. You'll have to create a django library project that we'll call `django-formidable-color-picker`. Let's say that this module has its own `setup.py` with the appropriate scripts to be installed in dev mode using `pip install -e ./`.

Let's also say that you have added it in your `INSTALLED_APPS`.

11.1 Tree structure

```
.
├── formidable_color_picker
│   ├── apps.py
│   ├── __init__.py
|   |── field_builder.py
│   ├── serializers.py
├── setup.cfg
└── setup.py
```

11.2 Loading the field for building time

The first file we're going to browse is `serializers.py`. Here's a minimal version of it:

```
from formidable.register import load_serializer, FieldSerializerRegister
from formidable.serializers.fields import FieldSerializer, BASE_FIELDS

field_register = FieldSerializerRegister.get_instance()


@load_serializer(field_register)
class ColorPickerFieldSerializer(FieldSerializer):

    type_id = 'color_picker'

    class Meta(FieldSerializer.Meta):
        fields = BASE_FIELDS
```

Then you're going to need to make sure that Django would catch this file at startup, and thus load the Serializer. It's done via the `apps.py` file.

```
from django.apps import AppConfig


class FormidableColorPickerConfig(AppConfig):
    """
    Formidable Color Picker configuration class.
    """
    name = 'formidable_color_picker'

    def ready(self):
        """
        Load external serializer when ready
        """
        from . import serializers  # noqa
```

As you'd do for any other Django application, you can now add this line to your `__init__.py` file at the root of the python module:

```
default_app_config = 'formidable_color_picker.apps.FormidableColorPickerConfig'
```

11.2.1 Check that it's working

Loading the Django shell:

```
>>> from formidable.serializers import FormidableSerializer
>>> data = {
    "label": "Color picker test",
    "description": "May I help you pick your favorite color?",
    "fields": [{
        "slug": "color",
        "label": "What is your favorite color?",
        "type_id": "color_picker",
        "accesses": [],
    }]
```

(continues on next page)

(continued from previous page)

```
}
>>> instance = FormidableSerializer(data=data)
>>> instance.is_valid()
True
>>> formidable_instance = instance.save()
```

This means that you can create a form with a field whose type is not in `django-formidable` code, but in your module's.

Then you can also retrieve this instance JSON defintion

```
>>> import json
>>> print(json.dumps(formidable_instance.to_json(), indent=2))
{
  "label": "Color picker test",
  "description": "May I help you pick your favorite color?",
  "fields": [
    {
      "slug": "color",
      "label": "What is your favorite color?",
      "type_id": "color_picker",
      "placeholder": null,
      "description": null,
      "accesses": [],
      "validations": [],
      "defaults": [],
    }
  ],
  "id": 42,
  "conditions": [],
  "version": 5
}
```

11.2.2 Making your field a bit more clever

Let's say that colors can be expressed in two ways: RGB tuple (`rgb`) or Hexadecimal expression (`hex`). This means your field has to be parametrized in order to store this information at the builder step. Let's imagine your JSON payload would look like:

```
{
    "label": "Color picker test",
    "description": "May I help you pick your favorite color?",
    "fields": [{
        "slug": "color",
        "label": "What is your favorite color?",
        "type_id": "color_picker",
        "accesses": [],
        "color_format": "hex"
    }]
}
```

You want then to make sure that your user would not send a wrong parameter, as in these BAD examples:

```
"color_format": ""
"color_format": "foo"
"color_format": "wrong"
```

For this specific field, you only want one parameter and its key is `format` and its values are only `hex` or `rgb`

Let's add some validation in your Serializer, then.

```
from rest_framework import serializers
from formidable.register import load_serializer, FieldSerializerRegister
from formidable.serializers.fields import FieldSerializer, BASE_FIELDS

field_register = FieldSerializerRegister.get_instance()


@load_serializer(field_register)
class ColorPickerFieldSerializer(FieldSerializer):

    type_id = 'color_picker'

    allowed_formats = ('rgb', 'hex')
    default_error_messages = {
        "missing_parameter": "You need a `format` parameter for this field",
        "invalid_format": "Invalid format: `{format}` is not one of {formats}."
    }

    class Meta(FieldSerializer.Meta):
        config_fields = ('color_format', )
        fields = BASE_FIELDS + ('parameters',)

    def to_internal_value(self, data):
        data = super().to_internal_value(data)
        # Check if the parameters are compliant
        format = data.get('color_format')
        if format is None:
            self.fail('missing_parameter')

        if format not in self.allowed_formats:
            self.fail("invalid_format",
                      format=format, formats=self.allowed_formats)

        return data
```

11.3 Load your field for the form filler

In your Django settings, add or update the `settings.FORMIDABLE_EXTERNAL_FIELD_BUILDERS` variable, like this:

```
FORMIDABLE_EXTERNAL_FIELD_BUILDERS = {
    "color_picker": 'formidable_color_picker.field_builder.ColorPickerFieldBuilder',
}
```

Then this namespace should point at your `ColorPickerFieldBuilder` class, which can be written as follows:

Important: The classes you're pointing at in this settings must be subclasses of `formidable.forms.`

`field_builder.FieldBuilder`.

```
import re
from formidable.forms.fields import ParametrizedFieldMixin, CharField
from formidable.forms.field_builder import FieldBuilder

COLOR_RE = re.compile('^#(?:[0-9a-fA-F]{3}){1,2}$')

class ColorPickerWidget(TextInput):
    """
    This widget class enables to use the :meth:`to_formidable()` helper.
    """
    type_id = 'color_picker'

class ColorPickerField(ParametrizedFieldMixin, CharField):
    """
    The ColorPickerField should inherit from a ``formidable.forms.fields``
    subclass.
    """
    widget = ColorPickerWidget

    def to_python(self, value):
        return value

    def validate(self, value):
        # Depending on the parent class, it might be a good idea to call
        # super() in order to use the parents validation.
        super().validate(value)
        params = getattr(self, '__formidable_field_parameters', {})
        color_format = params.get('color_format')
        if color_format == 'rgb':
            if value not in ('red', 'green', 'blue'):
                raise forms.ValidationError("Invalid color: {}".format(value))
        elif color_format == 'hex':
            if not COLOR_RE.match(value):
                raise forms.ValidationError("Invalid color: {}".format(value))
        else:
            raise forms.ValidationError("Invalid color format.")

class ColorPickerFieldBuilder(FieldBuilder):
    field_class = ColorPickerField
```

Important:

- The field should inherit from a formidable Field class, to enable `to_formidable()` and `to_json()` to be used
- The `widget` associated with the Field should have the `type_id` property set to the same than the Serializer.

Note: Full example

You may browse this as a complete directly usable example in the following repository: "django-formidable-color-picker"

CHAPTER 12

Maintainers' documentation

12.1 How to release

The contents of this section is a detailed version of the "release" part of the `.github/PULL_REQUEST_TEMPLATE.md` file.

12.1.1 Requirements

You can use a dedicated virtualenv, or install the following in your userspace, but these should be available in your `$PATH`:

- Python3 (any version)
- twine

12.1.2 Pre-release

- Create a branch with an adequate name, such as `release/x.y.z`.
- Edit the `formidable/__init__.py` source file and change the value of `formidable.version` to the appropriate version number.
- Amend the `CHANGELOG.rst` file to reflect your change. Put there the version number, the date, and do not hesitate to re-arrange its content if needed (e.g.: put sub-sections in the release notes).
- *If the version deprecates one or more feature(s)* check the docs `deprecations.rst` file and change it if necessary.
- Check if you have to edit other files and change them accordingly (e.g.: README).

Commit

Once your content is ready, **commit it**:

```
git commit -am "Release x.y.z"
```

If you want, you can also make a more detailed commit message, by copying/pasting the contents of the Changelog.

Push

Push your branch on Github and wait for the CI to return green.

You can also start to **create your Pull-Request** at this point, and check if you are at the correct step in the "Release" checklist.

> **Attention:** When to tag?
>
> If you are very confident, you can tag here. But we'd recommend to wait to be sure that you have everything sorted out.

Back to development

- Edit the `CHANGELOG.rst` file to add a "master (unreleased)" section, with a dummy log item, such as "Nothing to see here yet".
- Edit the `formidable/__init__.py` source file and put a non-release version number, such as `x.y+1.0.dev0`.
- Commit this change with, for example, the following command: `git commit -a -m "Back to dev => x.y+1.0.dev0"`

Again, push the branch and wait for the tests to be green.

At this point, the pull-request should be ready for review.

12.1.3 Release

If the CI has returned a successful result, and your peers have reviewed your PR, you're ready to proceed with the release.

Tag the right commit

You should have two commits in your log corresponding to your latest changes:

```
$ git log --pretty=format:'%h %ad | %s' --date=short -n 2
8fd30ec 2021-04-29 | Back to dev => x.y+1.0.dev0
5b65073 2021-04-29 | Release x.y.0
```

Checkout to the "Release" commit and tag it.

```
$ git checkout 5b65073
$ git tag x.y.0
```

This tag can be pushed to Github with:

```
$ git push --tags
```

Generate files

Now you can generate the files using the following command at the root of the project:

```
$ python3 setup.py sdist bdist_wheel
```

This should produce two files:

- `dist/django-formidable-x.y.0.tar.gz`
- `dist/django_formidable-x.y.0-py3-none-any.whl`

Merge the Pull Request

Merge from Github, or, if you dislike merge commits, type the following commands from your local copy:

```
$ git checkout master
$ git merge --ff release/x.y.z
$ git push
```

Upload to PyPI

In order to upload to PyPI, you should have an account and have at least the *maintainer* or *owner* role for this project **and** have your `.pypirc` correctly configured to upload files (i.e. have the pypi repository as default and correct credentials, using your password or a project token).

Using `twine` you may now upload the two files previously generated:

```
twine upload dist/django-formidable-x.y.0.tar.gz dist/django_formidable-x.y.0-py3-
↪none-any.whl
```

You can then go to https://pypi.org/project/django-formidable/ to check the latest version.

Hint: Due to asynchronous tasks and cache invalidation, the latest version may not appear immediately. Be patient.

12.1.4 Post-release

There are a few cleanup tasks, such as:

- Delete the release branch,
- Edit the Release page on Github to reflect the changelog,
- Eventually make some feedback on the issues impacted by the new release,
- Enjoy & celebrate!

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Index

F

G