latest

Release 0.13.0

Jul 10, 2017

Contents

1 HTML5 Features 3

2 Installation 5

3 Usage 7

4 File Uploads 9

5 Configuration 11

6 Custom Fields and Widgets 13

7 Email Templates 15

8 Signals 17

9 Dynamic Field Defaults 19

10 XLS Export 21

Created by Stephen McDonald

A Django reusable app providing the ability for admin users to create their own forms within the admin interface, drawing from a range of field widgets such as regular text fields, drop-down lists and file uploads. Options are also provided for controlling who gets sent email notifications when a form is submitted. All form entries are made available in the admin via filterable reporting with CSV/XLS export.

Form builder:

Data reporting:

CHAPTER 1

HTML5 Features

The following HTML5 form features are supported.

- `placeholder` attributes
- `required` attributes
- `email` fields
- `date` fields
- `datetime` fields
- `number` fields
- `url` fields

CHAPTER 2

Installation

The easiest way to install django-forms-builder is directly from PyPi using pip by running the command below:

```
$ pip install -U django-forms-builder
```

Otherwise you can download django-forms-builder and install it directly from source:

```
$ python setup.py install
```

Once installed you can configure your project to use django-forms-builder with the following steps.

Add `forms_builder.forms` to `INSTALLED_APPS` in your project's `settings` module:

```
INSTALLED_APPS = (
    # other apps
    'forms_builder.forms',
)
```

If you haven't already, ensure `django.core.context_processors.request` is in the `TEMPLATE_CONTEXT_PROCESSORS` setting in your project's `settings` module:

```
TEMPLATE_CONTEXT_PROCESSORS = (
    # other context processors
    "django.core.context_processors.request",
    # Django 1.6 also needs:
    'django.contrib.auth.context_processors.auth',
)
```

Then add `forms_builder.forms.urls` to your project's `urls` module:

```
from django.conf.urls.defaults import patterns, include, url
import forms_builder.forms.urls # add this import

from django.contrib import admin
admin.autodiscover()
```

```
urlpatterns = patterns('',
    # other urlpatterns
    url(r'^admin/', include(admin.site.urls)),
    url(r'^forms/', include(forms_builder.forms.urls)),
)
```

Finally, sync your database:

```
$ python manage.py syncdb
```

As of version 0.5, django-forms-builder provides South migrations. If you use south in your project, you'll also need to run migrations:

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

CHAPTER 3

Usage

Once installed and configured for your project just go to the admin page for your project and you will see a new Forms section. In this you can create and edit forms. Forms are then each viewable with their own URLs. A template tag `render_built_form` is also available for displaying forms outside of the main form view provided. It will display a form when given an argument in one of the following formats, where `form_instance` is an instance of the `Form` model:

```
{% render_built_form form_instance %}
{% render_built_form form=form_instance %}
{% render_built_form id=form_instance.id %}
{% render_built_form slug=form_instance.slug %}
```

This allows forms to be displayed without having a form instance, using a form's slug or ID, which could be hard-coded in a template, or stored in another model instance.

CHAPTER 4

File Uploads

It's possible for admin users to create forms that allow file uploads which can be accessed via a download URL for each file that is provided in the CSV export. By default these uploaded files are stored in an obscured location under your project's `MEDIA_ROOT` directory but ideally the should be stored somewhere inaccessible to the public. To set the location where files are stored to be somewhere outside of your project's `MEDIA_ROOT` directory you just need to define the `FORMS_BUILDER_UPLOAD_ROOT` setting in your project's `settings` module. Its value should be an absolute path on the web server that isn't accessible to the public.

CHAPTER 5

Configuration

The following settings can be defined in your project's `settings` module.

- `FORMS_BUILDER_FIELD_MAX_LENGTH` - Maximum allowed length for field values. Defaults to `2000`
- `FORMS_BUILDER_LABEL_MAX_LENGTH` - Maximum allowed length for field labels. Defaults to `20`
- `FORMS_BUILDER_EXTRA_FIELDS` - Sequence of custom fields that will be added to the form field types. Defaults to `()`
- `FORMS_BUILDER_UPLOAD_ROOT` - The absolute path where files will be uploaded to. Defaults to `None`
- `FORMS_BUILDER_USE_HTML5` - Boolean controlling whether HTML5 form fields are used. Defaults to `True`
- `FORMS_BUILDER_USE_SITES` - Boolean controlling whether forms are associated to Django's Sites framework. Defaults to `"django.contrib.sites" in settings.INSTALLED_APPS`
- `FORMS_BUILDER_EDITABLE_SLUGS` - Boolean controlling whether form slugs are editable in the admin. Defaults to `False`
- `FORMS_BUILDER_CHOICES_QUOTE` - Char to start a quoted choice with. Defaults to the backtick char: ‘
- `FORMS_BUILDER_CHOICES_UNQUOTE` - Char to end a quoted choice with. Defaults to the backtick char: ‘
- `FORMS_BUILDER_CSV_DELIMITER` - Char to use as a field delimiter when exporting form responses as CSV. Defaults to a comma: ,
- `FORMS_BUILDER_EMAIL_FAIL_SILENTLY` - Bool used for Django's `fail_silently` argument when sending email. Defaults to `settings.DEBUG`.

CHAPTER 6

Custom Fields and Widgets

You can also add your own custom fields or widgets to the choices of fields available for a form. Simply define a sequence for the FORMS_BUILDER_EXTRA_FIELDS setting in your project's settings module, where each item in the sequence is a custom field that will be available.

Each field in the sequence should be a three-item sequence containing an ID, a dotted import path for the field class, and a field name, for each custom field type. The ID is simply a numeric constant for the field, but cannot be a value already used, so choose a high number such as 100 or greater to avoid conflicts:

```
FORMS_BUILDER_EXTRA_FIELDS = (
    (100, "django.forms.BooleanField", "My cool checkbox"),
    (101, "my_module.MyCustomField", "Another field"),
)
```

You can also define custom widget classes for any of the existing or custom form fields via the FORMS_BUILDER_EXTRA_WIDGETS setting. Each field in the sequence should be a two-item sequence containing the same ID referred to above for the form field class, and a dotted import path for the widget class:

```
FORMS_BUILDER_EXTRA_WIDGETS = (
    (100, "my_module.MyCoolWidget"),
    (101, "my_other_module.AnotherWidget"),
)
```

Note that using the FORMS_BUILDER_EXTRA_WIDGETS setting to define custom widgets for field classes of your own is somewhat redundant, since you could simply define the widgets on the field classes directly in their code.

CHAPTER 7

Email Templates

The django-email-extras package is used to send multipart email notifications using Django's templating system for constructing the emails, to users submitting forms, and any recipients specified when creating a form via Django's admin.

Templates for HTML and text versions of the email can be found in the `templates/email_extras` directory. This allows you to customize the look and feel of emails that are sent to form submitters. Along with each of the `form_response` email templates which are used to email the form submitter, you'll also find corresponding `form_response_copies` templates, that extend the former set - these are used as the templates for emailing any extra recipients specified for the form in the admin interface. By default they simply extend the `form_response` templates, but you can modify them should you need to customize the emails sent to any extra recipients.

Note: With `django-email-extras` installed, it's also possible to configure PGP encrypted emails to be send to staff members, allowing forms to be built for capturing sensitive information. Consult the django-email-extras documentation for more info.

CHAPTER 8

Signals

Two signals are provided for hooking into different states of the form submission process.

- `form_invalid(sender=request, form=form)` - Sent when the form is submitted with invalid data.
- `form_valid(sender=request, form=form, entry=entry)` - Sent when the form is submitted with valid data.

For each signal the sender argument is the current request. Both signals receive a `form` argument is given which is the `FormForForm` instance, a `ModelForm` for the `FormEntry` model. The `form_valid` signal also receives a `entry` argument, which is the `FormEntry` model instance created.

Some examples of using the signals would be to monitor how users are causing validation errors with the form, or a pipeline of events to occur on successful form submissions. Suppose we wanted to store a logged in user's username against each form when submitted, given a form containing a field with the label `Username` with its field_type set to `Hidden`:

```
from django.dispatch import receiver
from forms_builder.forms.signals import form_valid

@receiver(form_valid)
def set_username(sender=None, form=None, entry=None, **kwargs):
    request = sender
    if request.user.is_authenticated():
        field = entry.form.fields.get(label="Username")
        field_entry, _ = entry.fields.get_or_create(field_id=field.id)
        field_entry.value = request.user.username
        field_entry.save()
```

CHAPTER 9

Dynamic Field Defaults

As of version 0.6, you can use Django template code for default field values. For example you could enter `{{ request.user.username }}` and the field will be pre-populated with a user's username if they're authenticated.

CHAPTER 10

XLS Export

By default, django-forms-builder provides export of form entries via CSV file. You can also enable export via XLS file (Microsoft Excel) by installing the xlwt package:

```
$ pip install xlwt
```