Wagtail Streamforms Documentation

Release 1.6.3

STuart George

Nov 22, 2017

Contents:

1 What else is included? 3

1.1 Installation . . . 3
1.2 Basic Usage . . . 4
1.2.1 Using the template tag . . . 4
1.3 Form Templates . . . 4
1.3.1 Deleted forms . . . 5
1.3.2 Messaging . . . 5
1.4 Form Customisation . . . 6
1.4.1 Custom basic form . . . 6
1.4.2 Custom email form . . . 6
1.4.3 Custom email form with content . . . 6
1.4.4 Custom form submission model . . . 7
1.4.5 Reference . . . 8
1.5 Form Submission Methods . . . 9
1.5.1 Providing your own submission method . . . 10
1.6 Permissions . . . 13
1.6.1 Form submission permissions . . . 13
1.7 Enabling reCAPTCHA . . . 13
1.8 Settings . . . 14
1.9 Contributors . . . 14
1.9.1 In our team . . . 14
1.9.2 In the community . . . 14
1.10 Changelog . . . 14
1.10.1 1.6.2 . . . 14
1.10.2 1.6.1 . . . 14
1.10.3 1.6.0 . . . 14
1.10.4 1.5.2 . . . 15
1.10.5 1.5.1 . . . 15
1.10.6 1.5.0 . . . 15
1.10.7 1.4.4 . . . 15
1.10.8 1.4.3 . . . 15
1.10.9 1.4.2 . . . 15
1.10.10 1.4.1 . . . 15
1.10.11 1.4.0 . . . 15
1.10.12 1.3.0 . . . 15
1.10.13 1.2.3 . . . 15

1.10.14 1.2.2 16
1.10.15 1.2.1 16
1.10.16 1.2.0 16
1.10.17 1.1.1 16
1.11 Screenshots 16

2 Indices and tables **23**

Allows you to build forms in the CMS admin area and add them to any StreamField in your site. You can create your own types of forms meaning an endless array of possibilities. Templates can be created which will then appear as choices when you build your form, allowing you to display and submit a form however you want.

CHAPTER 1

What else is included?

- Customise things like success and error messages, post submit redirects and more.
- Forms are processed via a `before_page_serve` hook. Meaning there is no fuss like remembering to include a page mixin.
- The hook can easily be disabled to provide the ability to create your own.
- Forms are categorised by their class in the CMS admin for easier navigation.
- Form submissions are also listed by their form which you can filter by date and are ordered by newest first.
- You can add site wide regex validators for use in regex fields.
- A form and its fields can easily be copied to a new form.
- There is a template tag that can be used to render a form, in case you want it to appear outside a StreamField.
- Recaptcha can be added to a form.

1.1 Installation

Wagtail Streamform is available on PyPI - to install it, just run:

```
pip install wagtailstreamforms
```

Once thats done you need to add the following to your `INSTALLED_APPS` settings:

```
INSTALLED_APPS = [
    ...
    'wagtail.contrib.modeladmin',
    'wagtail.wagtailforms',
    'wagtailstreamforms'
    ...
]
```

Run migrations:

```
python manage.py migrate
```

Go to your cms admin area and you will see the `Streamforms` section.

1.2 Basic Usage

Just add the `wagtailstreamforms.blocks.WagtailFormBlock()` in any of your streamfields:

```
body = StreamField([
    ...
    ('form', WagtailFormBlock())
    ...
])
```

And you are ready to go.

1.2.1 Using the template tag

There is also a template tag you can use outside of a streamfield, within a page. All this is doing is rendering the form using the same block as in the streamfield.

The tag takes three parameters:

- **slug** (`string`) - The slug of the form instance.
- **reference** (`string`) - This should be a unique string and needs to be persistent on refresh/reload. See note below.
- **action** (`string`) optional - The form action url.

Note: The reference is used when the form is being validated.

Because you can have any number of the same form on a page there needs to be a way of uniquely identifying the form beyond its `PK`. This is so that when the form has validation errors and it is passed back through the pages context, We know what form it is.

This reference **MUST** be persistent on page refresh or you will never see the errors.

Usage:

```
{% load streamforms_tags %}
{% streamforms_form slug="form-slug" reference="some-very-unique-reference" action="
↪" %}
```

1.3 Form Templates

You can create your own form templates to use against any form in the system, providing a vast array of ways to create, style and submit your forms.

The default template located at `streamforms/form_block.html` can be seen below:

```
<h2>{{ value.form.name }}</h2>
<form action="{{ value.form_action }}" method="post" novalidate>
    {% csrf_token %}
    {% for hidden in form.hidden_fields %}{{ hidden }}{% endfor %}
    {% for field in form.visible_fields %}
        {% include 'streamforms/partials/form_field.html' %}
    {% endfor %}
    <input type="submit" value="{{ value.form.submit_button_text }}">
</form>
```

Note: It is important here to keep the hidden fields as the form will have some in order to process correctly.

Once you have created you own you will need to add it to the list of available templates.

This is as simple as adding it to the `WAGTAILSTREAMFORMS_FORM_TEMPLATES` in settings:

```
# this is the defaults

WAGTAILSTREAMFORMS_FORM_TEMPLATES = (
    ('streamforms/form_block.html', 'Default Form Template'),
)
```

1.3.1 Deleted forms

In the event of a form being deleted which is still in use in a streamfield the following template will be rendered in its place:

`streamforms/non_existent_form.html`

```
<p>Sorry, this form has been deleted.</p>
```

You can override this by putting a copy of the template in you own project using the same path under a templates directory ie `app/templates/streamforms/non_existent_form.html`. As long as the app is before `wagtailstreamforms` in `INSTALLED_APPS` it will use your template instead.

1.3.2 Messaging

When the `success` or `error` message options are completed in the form builder and upon submission of the form a message is sent to django's messaging framework.

You will need to add `django.contrib.messages` to your `INSTALLED_APPS` setting:

```
INSTALLED_APPS = [
    ...
    'django.contrib.messages'
    ...
]
```

To display these in your site you will need to include somewhere in your page's markup a snippet similar to the following:

```
{% if messages %}
<ul class="messages">
    {% for message in messages %}
```

```
    <li{% if message.tags %} class="{{ message.tags }}"{% endif %}>{{ message }}</li>
    {% endfor %}
</ul>
{% endif %}
```

Any message from the form will then be displayed.

1.4 Form Customisation

Currently we have defined two different types of forms, one which just enables saving the submission and one to additionally email the results of the submission.

1.4.1 Custom basic form

You can easily add your own all you have to do is create a model that inherits from `wagtailstreamforms.models.BaseForm` add any additional fields or properties and this will be added to the cms admin area.

Example:

```
from wagtailstreamforms.models import BaseForm

class CustomForm(BaseForm):

    def process_form_submission(self, form):
        super(CustomForm, self).process_form_submission(form) # handles the␣
↪submission saving
        # do your own stuff here
```

1.4.2 Custom email form

If you want to inherit the additional email sending functionality then inherit from `wagtailstreamforms.models.AbstractEmailForm`. The saving of the submission and sending of the email is handled in the `process_form_submission` so be sure to call `super` if overriding that method.

Example:

```
from wagtailstreamforms.models import AbstractEmailForm

class CustomEmailForm(AbstractEmailForm):
     """ As above with email sending. """

     def process_form_submission(self, form):
         super(CustomEmailForm, self).process_form_submission(form) # handles the␣
↪submission saving and emailing
         # do your own stuff here
```

1.4.3 Custom email form with content

Here is an example of an email form that has an additional `RichTextField` rendered with the form. This is especially useful if your form is being rendered from the template tag and you dont want to slot it in a streamfield.

Model:

```
from wagtail.wagtailadmin.edit_handlers import TabbedInterface, ObjectList, FieldPanel
from wagtail.wagtailcore.fields import RichTextField
from wagtailstreamforms.models import AbstractEmailForm, BaseForm


class EmailFormWithContent(AbstractEmailForm):
    """ A form with content that sends and email. """

    content = RichTextField(blank=True)

    content_panels = [
        FieldPanel('content', classname='full'),
    ]

    edit_handler = TabbedInterface([
        ObjectList(AbstractEmailForm.settings_panels, heading='General'),
        ObjectList(AbstractEmailForm.field_panels, heading='Fields'),
        ObjectList(AbstractEmailForm.email_panels, heading='Email Submission'),
        ObjectList(content_panels, heading='Content'),
    ])
```

Template:

```
{% load wagtailcore_tags %}
<h2>{{ value.form.name }}</h2>
{% if value.form.content %}
   <div class="form-content">{{ value.form.content|richtext }}</div>
{% endif %}
<form action="{{ value.form_action }}" method="post" novalidate>
    {% csrf_token %}
    {% for hidden in form.hidden_fields %}{{ hidden }}{% endfor %}
    {% for field in form.visible_fields %}
        {% include 'streamforms/partials/form_field.html' %}
    {% endfor %}
    <input type="submit" value="{{ value.form.submit_button_text }}">
</form>
```

1.4.4 Custom form submission model

If you need to save additional data, you can use a custom form submission model. To do this, you need to:

- Define a model that extends `wagtailstreamforms.models.AbstractFormSubmission`.
- Override the `get_submission_class` and `process_form_submission` methods in your form model.

Example:

```
import json

from django.core.serializers.json import DjangoJSONEncoder
from django.db import models
from django.utils.translation import ugettext_lazy as _

from wagtail.wagtailcore.models import Page
from wagtailstreamforms.models import AbstractFormSubmission, BaseForm
```

```
class CustomForm(BaseForm):
    """ A form that saves the current user and page. """

    def get_data_fields(self):
        data_fields = super(ExampleForm, self).get_data_fields()
        data_fields += [
            ('user', _('User')),
            ('page', _('Page'))
        ]
        return data_fields

    def get_submission_class(self):
        return CustomFormSubmission

    def process_form_submission(self, form):
        if self.store_submission:
            self.get_submission_class().objects.create(
                form_data=json.dumps(form.cleaned_data, cls=DjangoJSONEncoder),
                form=self,
                page=form.page,
                user=form.user if not form.user.is_anonymous() else None
            )


class CustomFormSubmission(AbstractFormSubmission):
    user = models.ForeignKey(settings.AUTH_USER_MODEL, null=True, blank=True)
    page = models.ForeignKey(Page)

    def get_data(self):
        form_data = super(CustomFormSubmission, self).get_data()
        form_data.update({
            'page': self.page,
            'user': self.user
        })
        return form_data
```

Note: Its important to note here that the `form.page` and `form.user` seen above are passed in via the `before_serve_page` hook `wagtailstreamforms.wagtail_hooks.process_form`.

If you want to use a different method of saving the form and you require these you will need to pass them in yourself when adding `request.POST` to the form.

Example usage can be seen in *Providing your own submission method*

1.4.5 Reference

class `wagtailstreamforms.models.`**`BaseForm`**(**args*, ***kwargs*)

A form base class, any form should inherit from this.

`copy`()

Copy this form and its fields.

`get_data_fields`()

Returns a list of tuples with (field_name, field_label).

get_form_fields()
Form expects *form_fields* to be declared. If you want to change backwards relation name, you need to override this method.

get_submission_class()
Returns submission class.

You can override this method to provide custom submission class. Your class must be inherited from AbstractFormSubmission.

process_form_submission(*form*)
Accepts form instance with submitted data. Creates submission instance if self.store_submission = True.

You can override this method if you want to have custom creation logic. For example, you want to additionally send an email.

specific
Return this form in its most specific subclassed form.

specific_class
Return the class that this page would be if instantiated in its most specific form

class wagtailstreamforms.models.**AbstractEmailForm**(**args*, ***kwargs*)
A form that sends and email.

You can create custom form model based on this abstract model. For example, if you need a form that will send an email.

process_form_submission(*form*)
Process the form submission and send an email.

send_form_mail(*form*)
Send an email.

class wagtailstreamforms.models.**AbstractFormSubmission**(**args*, ***kwargs*)
Data for a form submission.

You can create custom submission model based on this abstract model. For example, if you need to save additional data or a reference to a user.

get_data()
Returns dict with form data.

You can override this method to add additional data.

1.5 Form Submission Methods

Form submissions are handled by the means of a wagtail `before_serve_page` hook. The built in hook at `wagtailstreamforms.wagtail_hooks.process_form` looks for a form in the post request, and either:

- processes it redirecting back to the current page or defined page in the form setup.
- or renders the current page with any validation error.

If no form was posted then the page serves in the usual manner.

Note: Currently the hook expects the form to be posting to the same page it exists on.

1.5.1 Providing your own submission method

If you do not want the current hook to be used you need to disable it by setting the `WAGTAILSTREAMFORMS_ENABLE_FORM_PROCESSING` to `False` in your settings:

```
WAGTAILSTREAMFORMS_ENABLE_FORM_PROCESSING = False
```

With this set no forms will be processed of any kind and you are free to process them how you feel fit.

A basic hook example

```
from django.contrib import messages
from django.shortcuts import redirect
from django.template.response import TemplateResponse

from wagtail.wagtailcore import hooks
from wagtailstreamforms.utils import get_form_instance_from_request


@hooks.register('before_serve_page')
def process_form(page, request, *args, **kwargs):
    """ Process the form if there is one, if not just continue. """

    if request.method == 'POST':
        form_def = get_form_instance_from_request(request)

        if form_def:
            form = form_def.get_form(request.POST, request.FILES, page=page,␣
↪user=request.user)
            context = page.get_context(request, *args, **kwargs)

            if form.is_valid():
                # process the form submission
                form_def.process_form_submission(form)

                # create success message
                if form_def.success_message:
                    messages.success(request, form_def.success_message, fail_
↪silently=True)

                # redirect to the page defined in the form
                # or the current page as a fallback - this will avoid refreshing and␣
↪submitting again
                redirect_page = form_def.post_redirect_page or page

                return redirect(redirect_page.get_url(request), context=context)

            else:
                # update the context with the invalid form and serve the page
                # IMPORTANT you must set these so that the when the form in the␣
↪streamfield is
                # rendered it knows that it is the form that is invalid
                context.update({
                    'invalid_stream_form_reference': form.data.get('form_reference'),
                    'invalid_stream_form': form
                })
```

```
                # create error message
                if form_def.error_message:
                    messages.error(request, form_def.error_message, fail_
↪silently=True)

                return TemplateResponse(
                    request,
                    page.get_template(request, *args, **kwargs),
                    context
                )
```

Supporting ajax requests

The only addition here from the basic example is just the `if request.is_ajax:` and the `JsonResponse` parts.

We are just making it respond with this if the request was ajax.

```
from django.contrib import messages
from django.http import JsonResponse
from django.shortcuts import redirect
from django.template.response import TemplateResponse

from wagtail.wagtailcore import hooks
from wagtailstreamforms.utils import get_form_instance_from_request


@hooks.register('before_serve_page')
def process_form(page, request, *args, **kwargs):
    """ Process the form if there is one, if not just continue. """

    if request.method == 'POST':
        form_def = get_form_instance_from_request(request)

        if form_def:
            form = form_def.get_form(request.POST, request.FILES, page=page,␣
↪user=request.user)
            context = page.get_context(request, *args, **kwargs)

            if form.is_valid():
                # process the form submission
                form_def.process_form_submission(form)

                # if the request is_ajax then just return a success message
                if request.is_ajax():
                    return JsonResponse({'message': form_def.success_message or
↪'success'})

                # create success message
                if form_def.success_message:
                    messages.success(request, form_def.success_message, fail_
↪silently=True)

                # redirect to the page defined in the form
                # or the current page as a fallback - this will avoid refreshing and␣
↪submitting again
                redirect_page = form_def.post_redirect_page or page
```

```
                return redirect(redirect_page.get_url(request), context=context)

            else:
                # if the request is_ajax then return an error message and the form␣
↪errors
                if request.is_ajax():
                    return JsonResponse({
                        'message': form_def.error_message or 'error',
                        'errors': form.errors
                    })

                # update the context with the invalid form and serve the page
                # IMPORTANT you must set these so that the when the form in the␣
↪streamfield is
                # rendered it knows that it is the form that is invalid
                context.update({
                    'invalid_stream_form_reference': form.data.get('form_reference'),
                    'invalid_stream_form': form
                })

                # create error message
                if form_def.error_message:
                    messages.error(request, form_def.error_message, fail_
↪silently=True)

                return TemplateResponse(
                    request,
                    page.get_template(request, *args, **kwargs),
                    context
                )
```

The template for the form might look like:

```
<h2>{{ value.form.name }}</h2>
<form action="{{ value.form_action }}" method="post" id="id_streamforms_{{ form.
↪initial.form_id }}" novalidate>
    {% csrf_token %}
    {% for hidden in form.hidden_fields %}{{ hidden }}{% endfor %}
    {% for field in form.visible_fields %}
        {% include 'streamforms/partials/form_field.html' %}
    {% endfor %}
    <input type="submit" value="{{ value.form.submit_button_text }}">
</form>
<script>
    $("#id_streamforms_{{ form.initial.form_id }}").submit(function(e) {
        $.ajax({
            type: "POST",
            url: ".",
            data: $(this).serialize(),
            success: function(data) {
                // do something with data
                console.log(data);
            },
            error: function(data) {
                // do something with data
                console.log(data);
            }
        });
```

```
        e.preventDefault();
    });
</script>
```

1.6 Permissions

Setting the level of access to administer your different types of forms is the same as it is for any page. Your types of forms will appear in the groups section of the wagtail admin > settings area.

Here you can assign the usual add, change and delete permissions.

Note: Its worth noting here that if you do delete a form it will also delete all submissions for that form.

1.6.1 Form submission permissions

Because the form submission models are not listed in the admin area the following statement applies.

Important: If you can either add, change or delete a form type then you can view all of its submissions. However to be able to delete the submissions, it requires that you can delete the form type.

1.7 Enabling reCAPTCHA

Has been enabled via the django-recaptcha package.

Once you have signed up for reCAPTCHA.

Follow the below and an option will be in the form setup `fields` tab to add a reCAPTCHA.

Just add `captcha` to your `INSTALLED_APPS` settings:

```
INSTALLED_APPS = [
    ...
    'captcha'
    ...
]
```

Add the required keys in your settings:

```
RECAPTCHA_PUBLIC_KEY = 'xxx'
RECAPTCHA_PRIVATE_KEY = 'xxx'
```

If you would like to use the new No Captcha reCaptcha add the setting `NOCAPTCHA = True`. For example:

```
NOCAPTCHA = True
```

1.8 Settings

Any settings with their defaults are listed below for quick reference.

```
# the label of the forms area in the admin sidebar
WAGTAILSTREAMFORMS_ADMIN_MENU_LABEL = 'Streamforms'

# the order of the forms area in the admin sidebar
WAGTAILSTREAMFORMS_ADMIN_MENU_ORDER = None

# enable the built in hook to process form submissions
WAGTAILSTREAMFORMS_ENABLE_FORM_PROCESSING = True

# the default form template choices
WAGTAILSTREAMFORMS_FORM_TEMPLATES = (
    ('streamforms/form_block.html', 'Default Form Template'),
)
```

1.9 Contributors

People that have helped in any way shape or form to get to where we are, many thanks.

1.9.1 In our team

- Dave Fuller
- Stuart George
- Tim Donhou

1.9.2 In the community

- Nathan Victor
- Tom Dyson

1.10 Changelog

1.10.1 1.6.2

- Added javascript to auto populate the form slug from the name

1.10.2 1.6.1

- Small tidy up in form code

1.10.3 1.6.0

- Stable Release

1.10.4 1.5.2

- Added `AbstractEmailForm` to more easily allow creating additional form types.

1.10.5 1.5.1

- Fix migrations being regenerated when template choices change

1.10.6 1.5.0

- Removed all project dependencies except wagtail and recapcha
- The urls no longer need to be specified in your `urls.py` and can be removed.

1.10.7 1.4.4

- The template tag now has the full page context incase u need a reference to the user or page

1.10.8 1.4.3

- Fixed bug where messages are not available in the template tags context

1.10.9 1.4.2

- Removed label value from recapcha field
- Added setting to set order of menu item in cms admin

1.10.10 1.4.1

- Added an optional error message to display if the forms have errors

1.10.11 1.4.0

- Added a template tag that can be used to render a form. Incase you want it to appear outside a streamfield

1.10.12 1.3.0

- A form and it's fields can easily be copied to a new form from within the admin area

1.10.13 1.2.3

- Fix paginator on submission list not remembering date filters

1.10.14 1.2.2

- Form submission viewing and deleting permissions have been implemented

1.10.15 1.2.1

- On the event that a form is deleted that is still referenced in a streamfield, we are rendering a generic template that can be overridden to warn the end user

1.10.16 1.2.0

- In the form builder you can now specify a page to redirect to upon successful submission of the form
- The page mixin StreamFormPageMixin that needed to be included in every page has now been replaced by a wagtail before_serve_page hook so you will need to remove this mixin

1.10.17 1.1.1

- Fixed bug where multiple forms of same type in a streamfield were both showing validation errors when one submitted

1.11 Screenshots

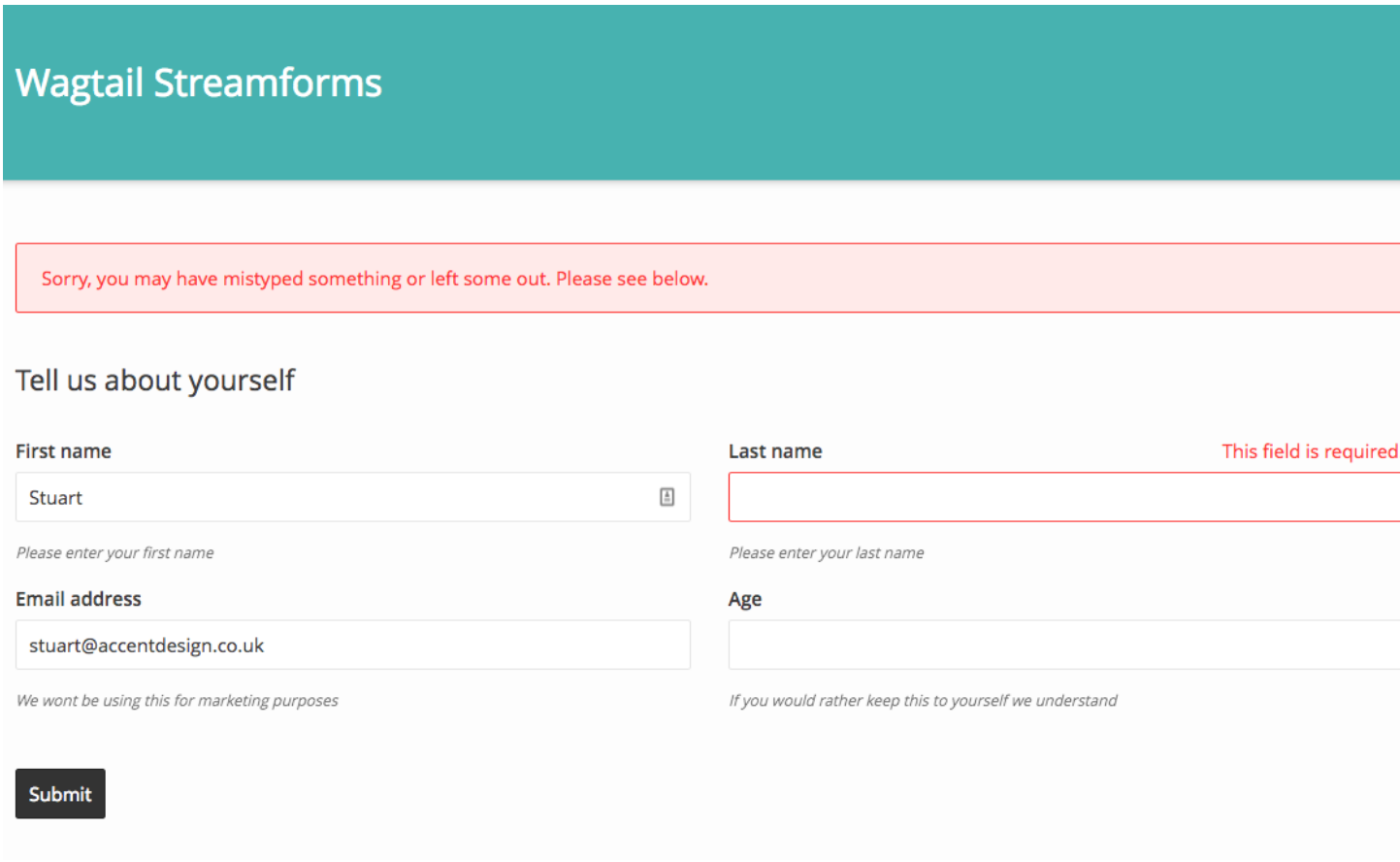


Fig. 1.1: Example Front End

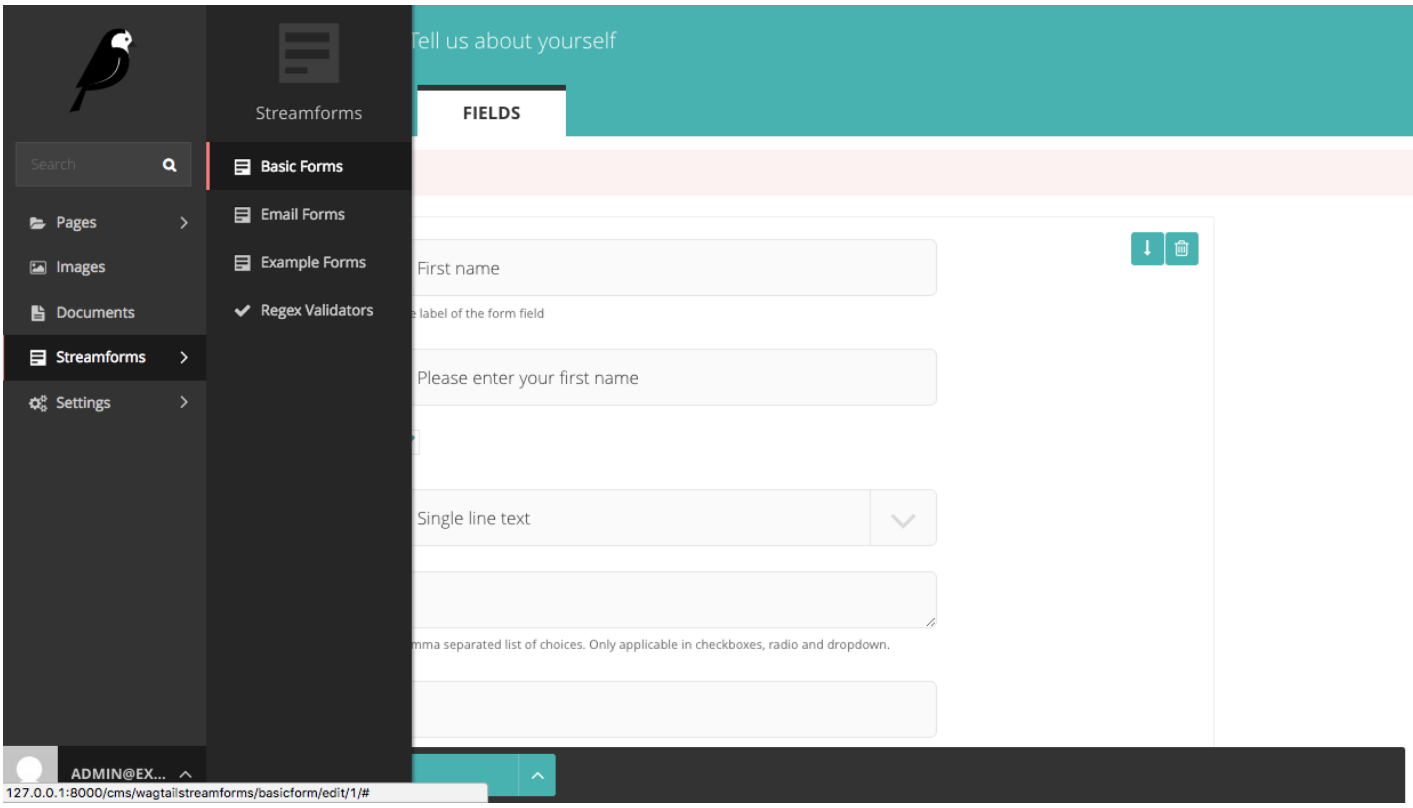

Streamforms
Search
Basic Forms
Email Forms
Example Forms
Regex Validators
Pages
Images
Documents
Streamforms
Settings
ADMIN@EX...
127.0.0.1:8000/cms/wagtailstreamforms/basicform/edit/1/#
Tell us about yourself
FIELDS
First name
label of the form field
Please enter your first name
Single line text
mma separated list of choices. Only applicable in checkboxes, radio and dropdown.

Fig. 1.2: Menu

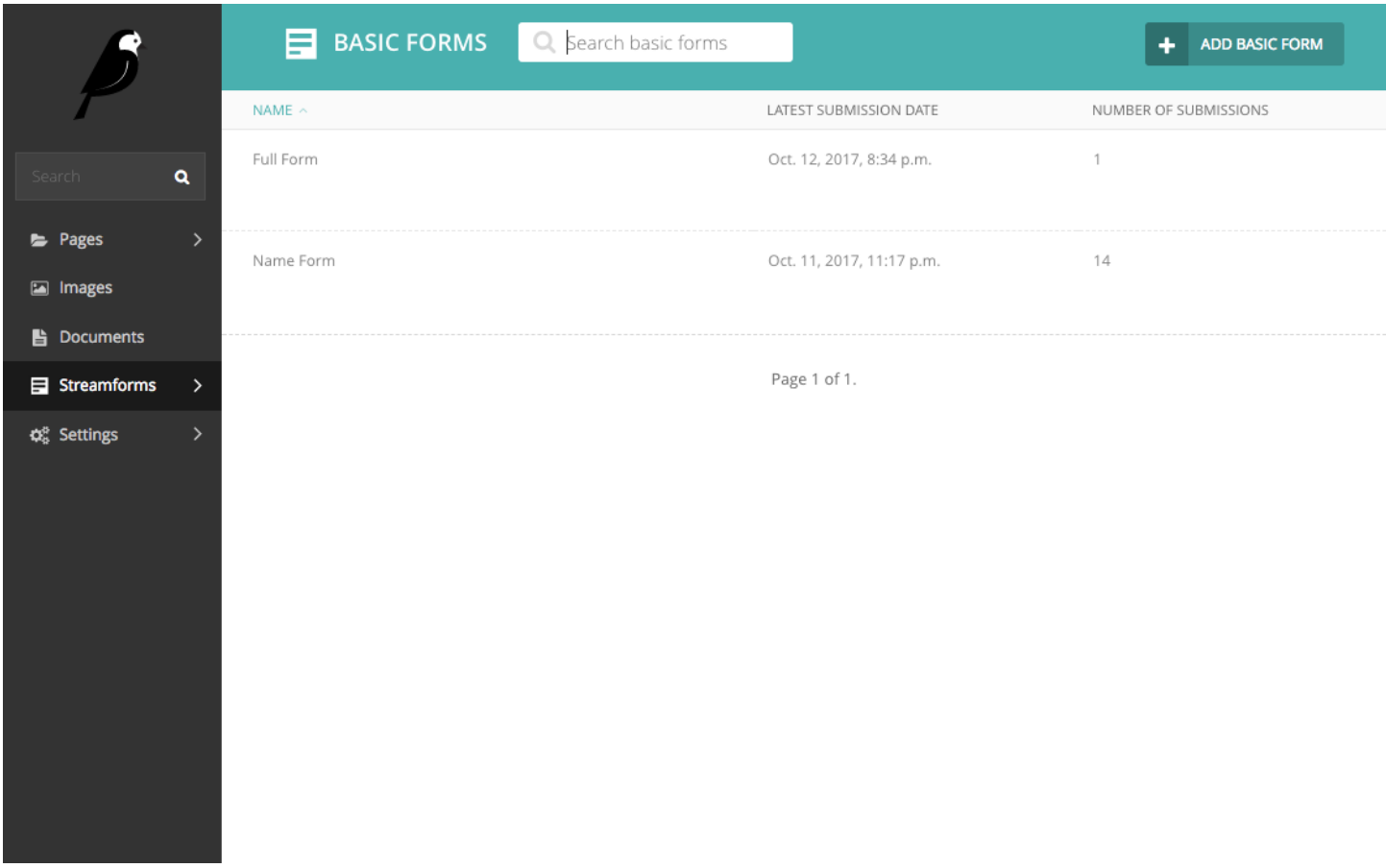


Fig. 1.3: Form Listing

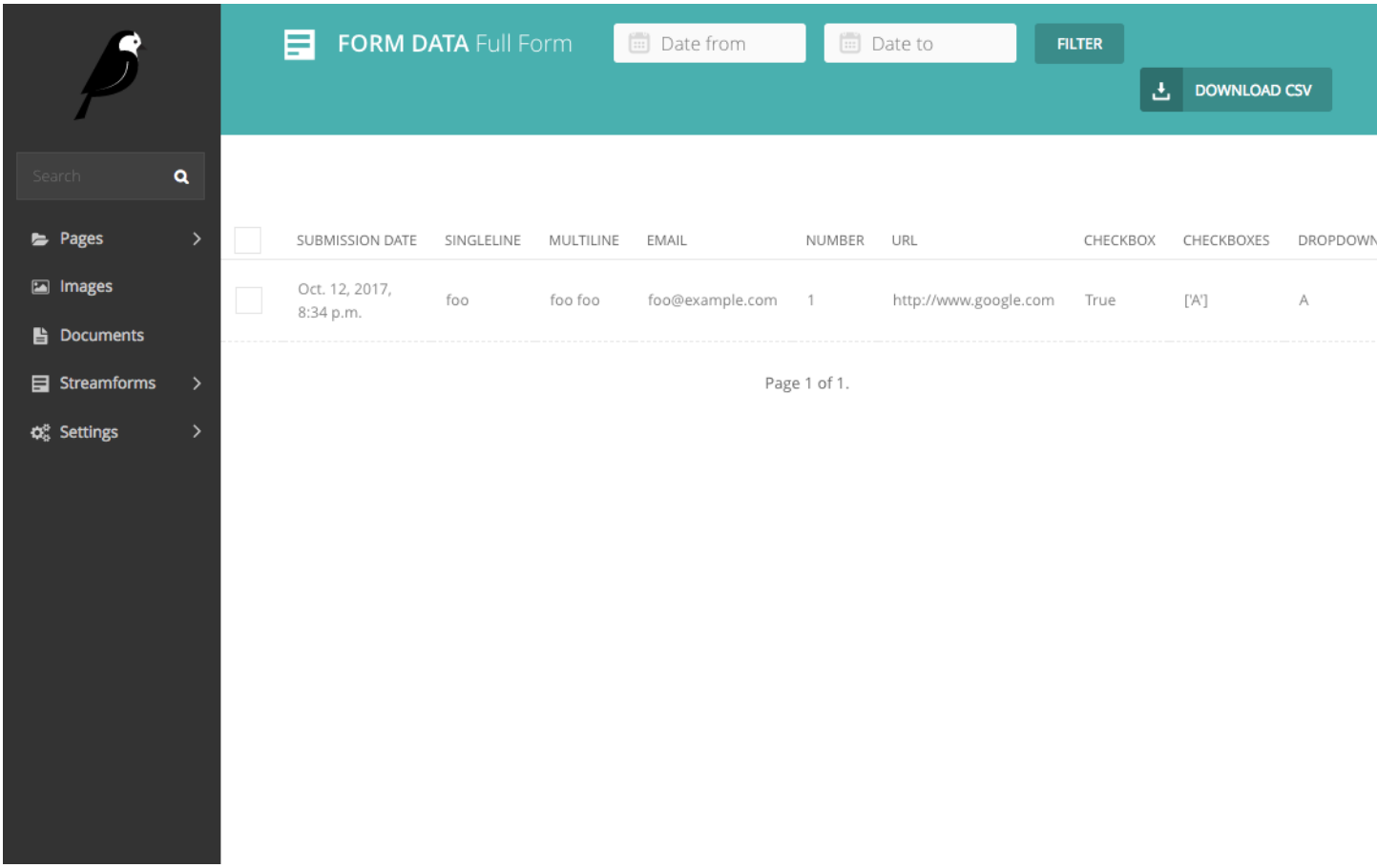


Fig. 1.4: Submission Listing

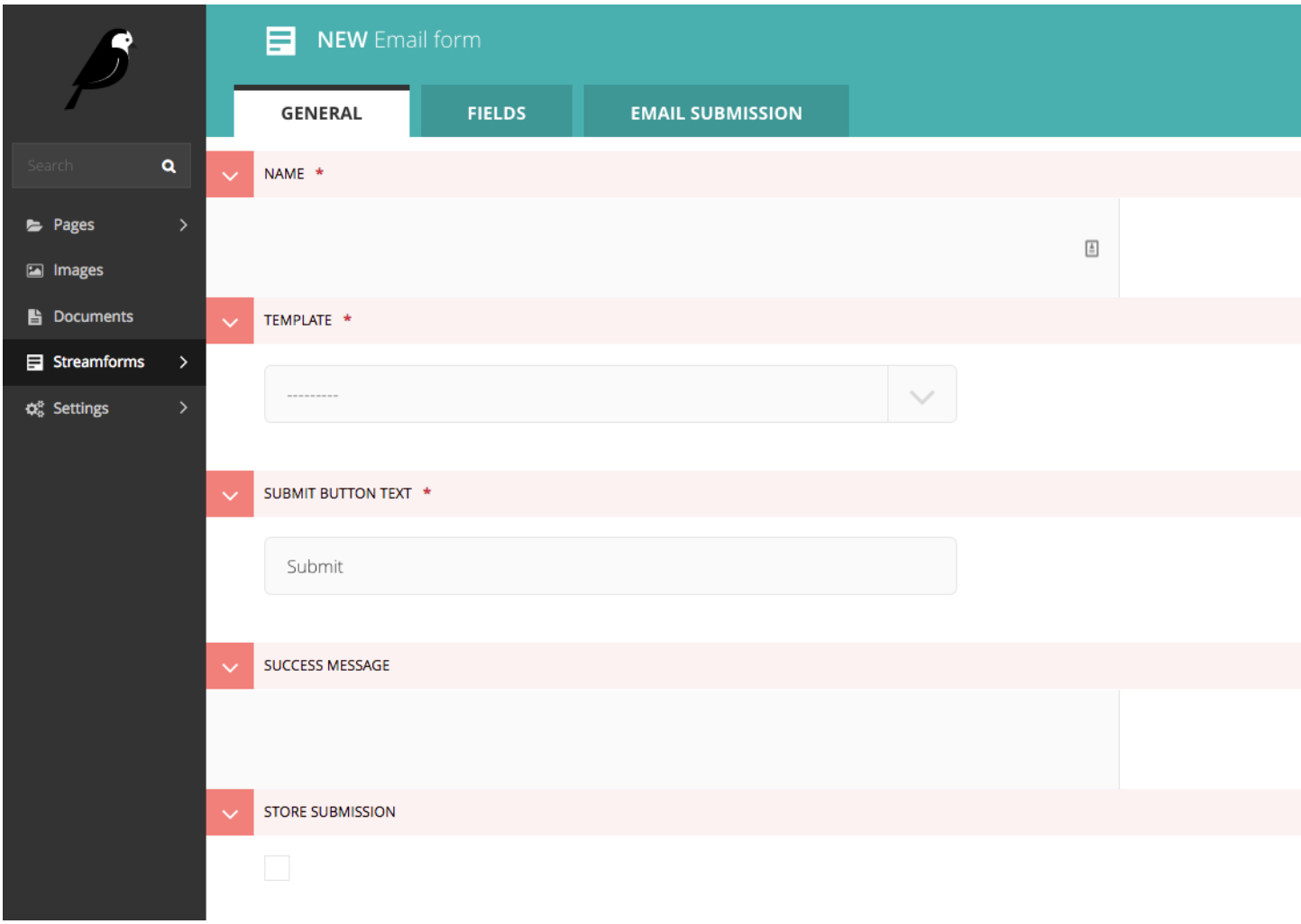


Fig. 1.5: Form Editing

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index

A

AbstractEmailForm (class in wagtailstreamforms.models), 9
AbstractFormSubmission (class in wagtailstreamforms.models), 9

B

BaseForm (class in wagtailstreamforms.models), 8

C

copy() (wagtailstreamforms.models.BaseForm method), 8

G

get_data() (wagtailstreamforms.models.AbstractFormSubmission method), 9
get_data_fields() (wagtailstreamforms.models.BaseForm method), 8
get_form_fields() (wagtailstreamforms.models.BaseForm method), 8
get_submission_class() (wagtailstreamforms.models.BaseForm method), 9

P

process_form_submission() (wagtailstreamforms.models.AbstractEmailForm method), 9
process_form_submission() (wagtailstreamforms.models.BaseForm method), 9

S

send_form_mail() (wagtailstreamforms.models.AbstractEmailForm method), 9
specific (wagtailstreamforms.models.BaseForm attribute), 9
specific_class (wagtailstreamforms.models.BaseForm attribute), 9