
django-newsletter Documentation

Release 0.5.1

Mathijs de Bruin

May 06, 2018

Contents

1	Installation	3
2	Settings	5
2.1	Disabling email confirmation	5
2.2	Configure rich text widget	5
2.3	Delay and batch size	5
3	Usage	7
3.1	Embed A Sign-up Form Within Any Page	7
4	Templates	9
4.1	Web view templates	9
4.2	Email templates	10
4.3	Using a premailer	10
5	Reference	11
5.1	Models	12
5.2	Forms	13
5.3	Views	14
6	Upgrading	19
6.1	0.7: Management command instead of django-extensions cron job	19
6.2	0.6: Upgrading from South to Django Migrations	19
6.3	0.5: Message templates in files	19
6.4	0.4: South migrations	19
	Python Module Index	21

Django app for managing multiple mass-mailing lists with both plaintext as well as HTML templates with rich text widget integration, images and a smart queueing system all right from the admin interface.

1. Install the package from PyPI:

```
pip install django-newsletter
```

Or get the latest & greatest from Github and link it to your application tree:

```
pip install -e git://github.com/dokterbob/django-newsletter.git#egg=django-  
→newsletter
```

(In either case it is recommended that you use [VirtualEnv](#) in order to keep your Python environment somewhat clean.)

2. Add newsletter to `INSTALLED_APPS` in `settings.py` and make sure that your favourite rich text widget (optional), some Django contrib dependencies and [sorl-thumbnail](#) are in there as well:

```
INSTALLED_APPS = (  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.auth',  
    'django.contrib.sites',  
    ...  
    # Imperavi (or tinymce) rich text editor is optional  
    # 'imperavi',  
    'sorl.thumbnail',  
    ...  
    'newsletter',  
    ...  
)
```

3. Configure any of the optional *Settings*.
4. Import subscription, unsubscription and archive URL's somewhere in your `urls.py`:

```
urlpatterns = [  
    ...  
    url(r'^newsletter/', include('newsletter.urls')),  
    ...  
]
```

5. Enable Django's `staticfiles` app so the admin icons, CSS and JavaScript will be available where we expect it.
6. Create the required data structure:

```
./manage.py migrate
```

7. Change the default contact email listed in `templates/newsletter/subscription_subscribe.html` and `templates/newsletter/subscription_update.html`.
8. (Optionally) Create message template overrides for specific newsletters in `templates/newsletter/message/<newsletter_slug>/<message_type>[_subject].<html|txt>` where `<message_type>` can be one from `subscribe`, `unsubscribe`, `message` or `update`.
9. You may now navigate to the Django admin where the Newsletter module should be available for you to play with.

In order to test if submissions work, make sure you create a newsletter, a subscription, a message and finally a submission.

After creating the submission, you must schedule it by clicking the 'submit' button in the top right of the page where you edit it.

10. Now you may perform a test submission with the `submit_newsletter` management command (`-v 2` is for extra verbosity):

```
./manage.py submit_newsletter -v 2
```

11. Add the `submit_newsletter` management command to `crontab`.

For example (for sending every 15 minutes):

```
*/15 * * * * <path_to_virtualenv>/bin/python <project_root>/manage.  
→py submit_newsletter 1>/dev/null 2>&1
```

To send mail, `django-newsletter` uses Django-provided email utilities, so ensure that `email settings` are properly configured for your project.

The following optional features may be configured.

2.1 Disabling email confirmation

Disable email confirmation for subscribe, unsubscribe and update actions for subscriptions.

By default subscribe, unsubscribe and update requests made by a user who is not logged in need to be confirmed by clicking on an activation link in an email. If you want all requested actions to be performed without email confirmation, add following line to settings.py:

```
NEWSLETTER_CONFIRM_EMAIL = False
```

For more granular control the `NEWSLETTER_CONFIRM_EMAIL` setting can be overridden for each of subscribe, unsubscribe and update actions, by adding `NEWSLETTER_CONFIRM_EMAIL_SUBSCRIBE` and/or `NEWSLETTER_CONFIRM_EMAIL_UNSUBSCRIBE` and/or `NEWSLETTER_CONFIRM_EMAIL_UPDATE` set to True or False.

2.2 Configure rich text widget

Known to work are [django-imperavi](#) as well as for [django-tinymce](#). Be sure to follow installation instructions for respective widgets. After installation, the widgets can be selected as follows:

```
# Using django-imperavi
NEWSLETTER_RICHTEXT_WIDGET = "imperavi.widget.ImperaviWidget"

# Using django-tinymce
NEWSLETTER_RICHTEXT_WIDGET = "tinymce.widgets.TinyMCE"
```

If not set, django-newsletter will fall back to Django's default TextField widget.

2.3 Delay and batch size

The delay between each email, batches en batch size can be specified with e.g.:

```
# Amount of seconds to wait between each email. Here 100ms is used.
``NEWSLETTER_EMAIL_DELAY = 0.1``

# Amount of seconds to wait between each batch. Here one minute is used.
``NEWSLETTER_BATCH_DELAY = 60``

# Number of emails in one batch
``NEWSLETTER_BATCH_SIZE = 100``
```

For both delays, sub-second delays can also be used. If the delays are not set, it will default to not sleeping.

1. Start the development server:

```
./manage.py runserver
```

2. Navigate to `/admin/` and: behold!
3. Setup a newsletter and create an initial message.
4. Preview the message and create submission.
5. Queue the submission for submission.
6. Process the submission queue:

```
./manage.py submit_newsletter
```

7. For a proper understanding, please take a look at the *Reference*.

3.1 Embed A Sign-up Form Within Any Page

If you want to include a sign-up form on any page of your site, similar to the code that MailChimp or other email services may provide, you simply paste the following code snippet where you want the form to appear:

```
<form enctype="multipart/form-data" method="post" action="/newsletter/[NAME-OF-
↪NEWSLETTER]/subscribe/">
{% csrf_token %}
<label for="id_email_field">E-mail:</label> <input type="email" name="email_field"
↪required="" id="id_email_field">
<button id="id_submit" name="submit" value="Subscribe" type="submit">Subscribe</
↪button>
</form>
```

Replace `[NAME-OF-NEWSLETTER]` with the name of your newsletter. You do not need to add anything to views, urls, or any other file. This snippet alone should simply work. Take note that the name field is removed from this, since most people only want the user to have to enter an email address to sign up for a newsletter. If you want to include the name field, you'd add this line before the `<button>` line:

```
<label for="id_name_field">Name:</label> <input type="text" name="name_field"
↪maxlength="30" id="id_name_field"><span class="helptext">optional</span>
```

To get started, we recommend copying the existing ‘stub’-templates from the module directory to your project’s *templates* dir:

```
cp -rv `python -c 'import newsletter; from os import path; print(path.  
↪dirname(newsletter.__file__))` /templates/newsletter <project_dir>/templates/
```

4.1 Web view templates

newsletter_list.html Newsletter list view, showing all newsletters marked as public and allowing authenticated Django users to (un)subscribe directly.

newsletter_detail.html Newsletter detail view, linking to subscribe, update, unsubscribe and archive views for a particular newsletter.

submission_archive.html Archive; list of public submissions for a particular newsletter.

subscription_subscribe.html Subscribe form for unauthenticated users.

subscription_subscribe_email_sent.html Confirmation of subscription request.

subscription_activate.html Activation form for (un)subscriptions or updates of unauthenticated users.

subscription_subscribe_activated.html Confirmation of activation of subscription.

subscription_unsubscribe_activated.html Confirmation of activation of unsubscription.

subscription_update_activated.html Confirmation of activation of update.

subscription_subscribe_user.html Subscribe form for authenticated users.

subscription_unsubscribe.html Unsubscribe form for unauthenticated users.

subscription_unsubscribe_email_sent.html Confirmation of unsubscription request.

subscription_unsubscribe_user.html Unsubscribe form for authenticated users.

subscription_update.html Update form for unauthenticated users.

subscription_update_email_sent.html Confirmation of update request.

4.2 Email templates

Email templates can be specified per newsletter in *message/<newsletter_slug>*. If no newsletter-specific templates are found, the defaults in the *message* folder are used.

When a newsletter is configured to send HTML-messages, the HTML and txt are both used to create a multipart message. When the use of HTML is not configured only the text templates are used.

The following templates can be defined:

message.(html|txt)

Template for rendering a messages with the following context available:

- *subscription*: Subscription containing name and email of recipient.
- *site*: Current *site* object.
- *submission*: Current submission.
- *message*: Current message.
- *newsletter*: Current newsletter.
- *date*: Publication date of submission.
- *STATIC_URL*: Django's *STATIC_URL* setting.
- *MEDIA_URL*: Django's *MEDIA_URL* setting.

message_subject.txt Template for the subject of an email newsletter. Context is the same as with messages.

subscribe.(html|txt) Template with confirmation link for subscription.

subscribe_subject.txt Subject template with confirmation link for subscription.

unsubscribe.(html|txt) Template with confirmation link for unsubscription.

unsubscribe_subject.txt Subject template with confirmation link for unsubscription.

update.(html|txt) Template with confirmation link for updating subscriptions.unsubscription.

update_subject.txt Subject template with confirmation link for updating subscriptions.

4.3 Using a premailer

A premailer is a program that translates embedded CSS into inline CSS. Inline CSS is much more widely supported in emails, but can make templates very messy if you have more than a couple lines of styling.

[django-premailer](#) is an open-source package on PyPI that adds a template tag that applies a premailer. Unfortunately, the package was [broken for Django 1.6 and upwards](#) at the time of writing. An example of a working version is available at this [gist](#) (requires [premailer](#) to be installed).

You can then use the template tag in your templates as follows:

```
{% load premailer %}{% premailer %}
<html>
<style type="text/css">
h1 { border:1px solid black }
p { color:red;}
</style>

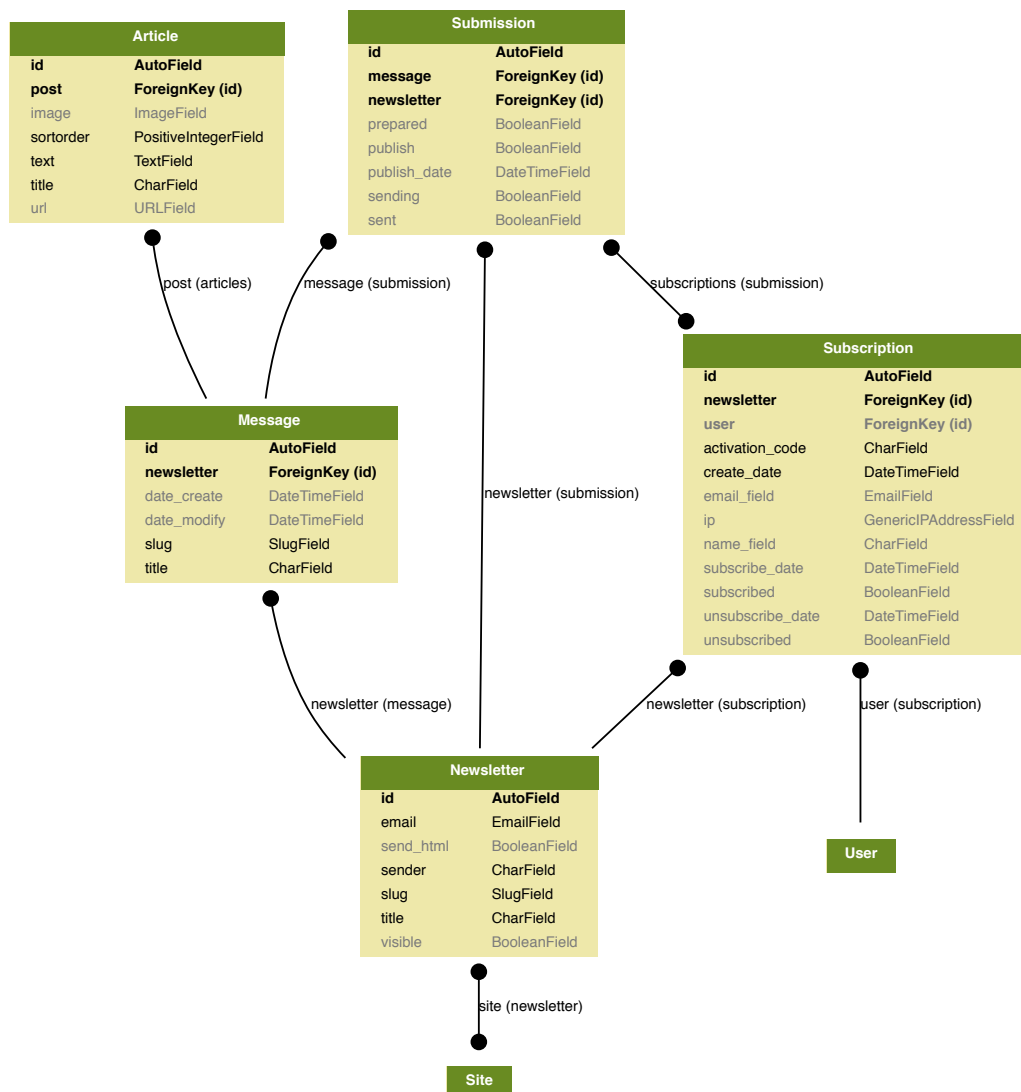
<h1 style="font-weight:bolder">Hey</h1>
<p>Hej</p>
</html>
{% endpremailer %}
```

CHAPTER 5

Reference

For now, this documentation is automatically generated from the source code.

5.1 Models



```
class Newsletter (id, title, slug, email, sender, visible, send_html)
```

```
Bases: django.db.models.base.Model
```

```
get_templates (action)
```

Return a subject, text, HTML tuple with e-mail templates for a particular action. Returns a tuple with subject, text and e-mail template.

```
class Subscription (id, user, name_field, email_field, ip, newsletter, create_date, activation_code, subscribed, subscribe_date, unsubscribed, unsubscribe_date)
```

```
Bases: django.db.models.base.Model
```

```
update (action)
```

Update subscription according to requested action: subscribe/unsubscribe/update/, then save the changes.

```
save (*args, **kwargs)
```


Perform some basic validation and state maintenance of Subscription. TODO: Move this code to a more suitable place (i.e. *clean()*) and cleanup the code. Refer to comment below and <https://docs.djangoproject.com/en/dev/ref/models/instances/#django.db.models.Model.clean>

class Article (*args, **kwargs)

Bases: `django.db.models.base.Model`

An Article within a Message which will be send through a Submission.

class Message (*args, **kwargs)

Bases: `django.db.models.base.Model`

Message as sent through a Submission.

get_next_article_sortorder ()

Get next available sortorder for Article.

class Submission (*args, **kwargs)

Bases: `django.db.models.base.Model`

Submission represents a particular Message as it is being submitted to a list of Subscribers. This is where actual queueing and submission happen.

save (**kwargs)

Set the newsletter from associated message upon saving.

5.2 Forms

class NewsletterForm (*args, **kwargs)

Bases: `django.forms.models.ModelForm`

This is the base class for all forms managing subscriptions.

class SubscribeRequestForm (*args, **kwargs)

Bases: `newsletter.forms.NewsletterForm`

Request subscription to the newsletter. Will result in an activation email being sent with a link where one can edit, confirm and activate one's subscription.

class UpdateRequestForm (*args, **kwargs)

Bases: `newsletter.forms.NewsletterForm`

Request updating or activating subscription. Will result in an activation email being sent.

class UnsubscribeRequestForm (*args, **kwargs)

Bases: `newsletter.forms.UpdateRequestForm`

Similar to previous form but checks if we have not already been unsubscribed.

class UpdateForm (*args, **kwargs)

Bases: `newsletter.forms.NewsletterForm`

This form allows one to actually update to or unsubscribe from the newsletter. To do this, a correct activation code is required.

class UserUpdateForm (data=None, files=None, auto_id='id_%s', prefix=None, initial=None, error_class=<class 'django.forms.utils.ErrorList'>, label_suffix=None, empty_permitted=False, instance=None, use_required_attribute=None)

Bases: `django.forms.models.ModelForm`

Form for updating subscription information/unsubscribing as a logged-in user.

5.3 Views

`is_authenticated` (*user*)

class NewsletterViewBase

Bases: object

Base class for newsletter views.

`queryset = <QuerySet []>`

`allow_empty = False`

`slug_url_kwarg = 'newsletter_slug'`

class NewsletterDetailView (***kwargs*)

Bases: `newsletter.views.NewsletterViewBase`, `django.views.generic.detail.DetailView`

class NewsletterListView (***kwargs*)

Bases: `newsletter.views.NewsletterViewBase`, `django.views.generic.list.ListView`

List available newsletters and generate a formset for (un)subscription for authenticated users.

`post` (*request*, ***kwargs*)

Allow post requests.

`get_context_data` (***kwargs*)

`get_formset` ()

Return a formset with newsletters for logged in users, or None.

class ProcessUrlDataMixin

Bases: object

Mixin providing the ability to process args and kwargs from url before dispatching request.

`process_url_data` (**args*, ***kwargs*)

Subclasses should put url data processing in this method.

`dispatch` (**args*, ***kwargs*)

class NewsletterMixin

Bases: `newsletter.views.ProcessUrlDataMixin`

Mixin retrieving newsletter based on `newsletter_slug` from url and adding it to context and form kwargs.

`process_url_data` (**args*, ***kwargs*)

Get newsletter based on `newsletter_slug` from url and add it to instance attributes.

`get_form_kwargs` ()

Add newsletter to form kwargs.

`get_context_data` (***kwargs*)

Add newsletter to context.

class ActionMixin

Bases: `newsletter.views.ProcessUrlDataMixin`

Mixin retrieving action from url and adding it to context.

`action = None`

`process_url_data` (**args*, ***kwargs*)

Add action from url to instance attributes if not already set.

`get_context_data` (***kwargs*)

Add action to context.

get_template_names ()
Return list of template names for proper action.

```
class ActionTemplateView (**kwargs)
    Bases: newsletter.views.NewsletterMixin, newsletter.views.ActionMixin,
           django.views.generic.base.TemplateView
```

View that renders a template for proper action, with newsletter and action in context.

```
class ActionFormView (**kwargs)
    Bases: newsletter.views.NewsletterMixin, newsletter.views.ActionMixin,
           django.views.generic.edit.FormView
```

FormView with newsletter and action support.

get_url_from_viewname (*viewname*)
Return url for given *viewname* and associated with this view newsletter and action.

```
class ActionUserView (**kwargs)
    Bases: newsletter.views.ActionTemplateView
```

Base class for subscribe and unsubscribe user views.

```
template_name = 'newsletter/subscription_%(action)s_user.html'
```

process_url_data (**args, **kwargs*)
Add confirm to instance attributes.

post (*request, *args, **kwargs*)

dispatch (**args, **kwargs*)

```
class SubscribeUserView (**kwargs)
    Bases: newsletter.views.ActionUserView
```

```
action = 'subscribe'
```

get (*request, *args, **kwargs*)

```
class UnsubscribeUserView (**kwargs)
    Bases: newsletter.views.ActionUserView
```

```
action = 'unsubscribe'
```

get (*request, *args, **kwargs*)

```
class ActionRequestView (**kwargs)
    Bases: newsletter.views.ActionFormView
```

Base class for subscribe, unsubscribe and update request views.

```
template_name = 'newsletter/subscription_%(action)s.html'
```

process_url_data (**args, **kwargs*)
Add error to instance attributes.

get_context_data (**kwargs)
Add error to context.

get_subscription (*form*)
Return subscription for the current request.

no_email_confirm (*form*)
Subscribe/unsubscribe user and redirect to action activated page.

get_success_url ()

form_valid (*form*)

```
class SubscribeRequestView (**kwargs)
    Bases: newsletter.views.ActionRequestView
```

```

    action = 'subscribe'

    form_class
        alias of SubscribeRequestForm

    confirm = False

    get_form_kwargs ()
        Add ip to form kwargs for submitted forms.

    get_subscription (form)

    dispatch (request, *args, **kwargs)

class UnsubscribeRequestView (**kwargs)
    Bases: newsletter.views.ActionRequestView

    action = 'unsubscribe'

    form_class
        alias of UnsubscribeRequestForm

    confirm = False

    dispatch (request, *args, **kwargs)

class UpdateRequestView (**kwargs)
    Bases: newsletter.views.ActionRequestView

    action = 'update'

    form_class
        alias of UpdateRequestForm

    no_email_confirm (form)
        Redirect to update subscription view.

class UpdateSubscriptionView (**kwargs)
    Bases: newsletter.views.ActionFormView

    form_class
        alias of UpdateForm

    template_name = 'newsletter/subscription_activate.html'

    process_url_data (*args, **kwargs)
        Add email, subscription and activation_code to instance attributes.

    get_initial ()
        Returns the initial data to use for forms on this view.

    get_form_kwargs ()
        Add instance to form kwargs.

    get_success_url ()

    form_valid (form)
        Get our instance, but do not save yet.

class SubmissionViewBase
    Bases: newsletter.views.NewsletterMixin

    Base class for submission archive views.

    date_field = 'publish_date'

    allow_empty = True

    queryset = <QuerySet []>

    slug_field = 'message__slug'

    year_format = '%Y'

```

```
month_format = '%m'
```

```
day_format = '%d'
```

```
process_url_data (*args, **kwargs)
```

Use only visible newsletters.

```
get_queryset ()
```

Filter out submissions for current newsletter.

```
class SubmissionArchiveIndexView (**kwargs)
```

Bases: `newsletter.views.SubmissionViewBase`, `django.views.generic.dates.ArchiveIndexView`

```
class SubmissionArchiveDetailView (**kwargs)
```

Bases: `newsletter.views.SubmissionViewBase`, `django.views.generic.dates.DateDetailView`

```
get_context_data (**kwargs)
```

Make sure the actual message is available.

```
get_template ()
```

Get the message template for the current newsletter.

```
render_to_response (context, **response_kwargs)
```

Return a simplified response; the template should be rendered without any context. Use a `SimpleTemplateResponse` as a `RequestContext` should not be used.

6.1 0.7: Management command instead of django-extensions cron job

In this version, we have deprecated support for the django-extensions cron job. Hence, it will become necessary to update the crontab; whereas before messages were submitted with the *runjobs hourly* cron job, this has now become *submit_newsletter*.

6.2 0.6: Upgrading from South to Django Migrations

Based on <https://docs.djangoproject.com/en/1.9/topics/migrations/#upgrading-from-south>, the procedure should be:

1. Remove 'south' from `INSTALLED_APPS`.
2. Run `python manage.py migrate --fake-initial`.

If you are upgrading from below 0.5, you need to upgrade to 0.5 first to perform required South migrations before moving to 0.6.

6.3 0.5: Message templates in files

As of 0.5 message templates are living in the filesystem like normal files instead of resorting in the EmailTemplate in the database. In most cases, South should take care of writing your existing templates to disk and deleting the database models.

6.4 0.4: South migrations

Since 5f79f40, the app makes use of `South` for schema migrations. As of this version, using South with django-newsletter is the official recommendation and [installing it](#) is easy.

When upgrading from a pre-South version of newsletter to a current release (in a project for which South has been enabled), you might have to fake the initial migration as the DB tables already exist. This can be done by running the following command:

```
./manage.py migrate newsletter 0001 --fake
```

- [Changes \(GitHub\)](#)
- [Contributors \(GitHub\)](#)

Python Module Index

- newsletter.forms, 13
newsletter.models, 12
newsletter.views, 14

A

action (ActionMixin attribute), 14
action (SubscribeRequestView attribute), 15
action (SubscribeUserView attribute), 15
action (UnsubscribeRequestView attribute), 16
action (UnsubscribeUserView attribute), 15
action (UpdateRequestView attribute), 16
ActionFormView (class in newsletter.views), 15
ActionMixin (class in newsletter.views), 14
ActionRequestView (class in newsletter.views), 15
ActionTemplateView (class in newsletter.views), 15
ActionUserView (class in newsletter.views), 15
allow_empty (NewsletterViewBase attribute), 14
allow_empty (SubmissionViewBase attribute), 16
Article (class in newsletter.models), 13

C

confirm (SubscribeRequestView attribute), 16
confirm (UnsubscribeRequestView attribute), 16

D

date_field (SubmissionViewBase attribute), 16
day_format (SubmissionViewBase attribute), 17
dispatch() (ActionUserView method), 15
dispatch() (ProcessUrlDataMixin method), 14
dispatch() (SubscribeRequestView method), 16
dispatch() (UnsubscribeRequestView method), 16

F

form_class (SubscribeRequestView attribute), 16
form_class (UnsubscribeRequestView attribute), 16
form_class (UpdateRequestView attribute), 16
form_class (UpdateSubscriptionView attribute), 16
form_valid() (ActionRequestView method), 15
form_valid() (UpdateSubscriptionView method), 16

G

get() (SubscribeUserView method), 15
get() (UnsubscribeUserView method), 15
get_context_data() (ActionMixin method), 14
get_context_data() (ActionRequestView method), 15
get_context_data() (NewsletterListView method), 14
get_context_data() (NewsletterMixin method), 14

get_context_data() (SubmissionArchiveDetailView method), 17

get_form_kwargs() (NewsletterMixin method), 14

get_form_kwargs() (SubscribeRequestView method), 16

get_form_kwargs() (UpdateSubscriptionView method), 16

get_formset() (NewsletterListView method), 14

get_initial() (UpdateSubscriptionView method), 16

get_next_article_sortorder() (Message method), 13

get_queryset() (SubmissionViewBase method), 17

get_subscription() (ActionRequestView method), 15

get_subscription() (SubscribeRequestView method), 16

get_success_url() (ActionRequestView method), 15

get_success_url() (UpdateSubscriptionView method), 16

get_template() (SubmissionArchiveDetailView method), 17

get_template_names() (ActionMixin method), 14

get_templates() (Newsletter method), 12

get_url_from_viewname() (ActionFormView method), 15

I

is_authenticated() (in module newsletter.views), 14

M

Message (class in newsletter.models), 13

month_format (SubmissionViewBase attribute), 17

N

Newsletter (class in newsletter.models), 12

newsletter.forms (module), 13

newsletter.models (module), 12

newsletter.views (module), 14

NewsletterDetailView (class in newsletter.views), 14

NewsletterForm (class in newsletter.forms), 13

NewsletterListView (class in newsletter.views), 14

NewsletterMixin (class in newsletter.views), 14

NewsletterViewBase (class in newsletter.views), 14

no_email_confirm() (ActionRequestView method), 15

no_email_confirm() (UpdateRequestView method), 16

P

post() (ActionUserView method), 15
post() (NewsletterListView method), 14
process_url_data() (ActionMixin method), 14
process_url_data() (ActionRequestView method), 15
process_url_data() (ActionUserView method), 15
process_url_data() (NewsletterMixin method), 14
process_url_data() (ProcessUrlDataMixin method), 14
process_url_data() (SubmissionViewBase method), 17
process_url_data() (UpdateSubscriptionView method),
16
ProcessUrlDataMixin (class in newsletter.views), 14

Q

queryset (NewsletterViewBase attribute), 14
queryset (SubmissionViewBase attribute), 16

R

render_to_response() (SubmissionArchiveDetailView
method), 17

S

save() (Submission method), 13
save() (Subscription method), 12
slug_field (SubmissionViewBase attribute), 16
slug_url_kwarg (NewsletterViewBase attribute), 14
Submission (class in newsletter.models), 13
SubmissionArchiveDetailView (class in newslet-
ter.views), 17
SubmissionArchiveIndexView (class in newslet-
ter.views), 17
SubmissionViewBase (class in newsletter.views), 16
SubscribeRequestForm (class in newsletter.forms), 13
SubscribeRequestView (class in newsletter.views), 15
SubscribeUserView (class in newsletter.views), 15
Subscription (class in newsletter.models), 12

T

template_name (ActionRequestView attribute), 15
template_name (ActionUserView attribute), 15
template_name (UpdateSubscriptionView attribute), 16

U

UnsubscribeRequestForm (class in newsletter.forms),
13
UnsubscribeRequestView (class in newsletter.views),
16
UnsubscribeUserView (class in newsletter.views), 15
update() (Subscription method), 12
UpdateForm (class in newsletter.forms), 13
UpdateRequestForm (class in newsletter.forms), 13
UpdateRequestView (class in newsletter.views), 16
UpdateSubscriptionView (class in newsletter.views), 16
UserUpdateForm (class in newsletter.forms), 13

Y

year_format (SubmissionViewBase attribute), 16