django-postman Documentation

Release 3.5.1.post1

Patrick Samson

Nov 18, 2017

Contents

This is an application for Django-powered websites.

Basically, the purpose is to allow authenticated users of a site to exchange private **messages** within the site. In this documentation, the word *user* is to be understood as an instance of a User, in the django.contrib.auth context.

So it is mainly for a User-to-User exchange. But it may be beneficial for a subscriber to receive inquiries from any visitor, ie even if non authenticated. For instance, a subscriber as a service provider wants an ask-me-details form on a presentation page to facilitate possible business contacts. In this case, the visitor is presented a compose message form with an additional field to give an email address for the reply. The email is obfuscated to the recipient.

What is a message ? Roughly a piece of text, about a subject, sent by a sender to a recipient. Each user has access to a collection of messages, stored in folders:

Inbox for incoming messages
Sent for sent messages
Archives for archived messages
Trash for messages marked as deleted

In folders, messages can be presented in two modes:

- by **conversation**, for a compact view: the original message and its replies are grouped in a set to constitute one sole entry. The lastest message (based on the time) is the representative of the set.
- by **message**, for an expanded view: each message is considered by itself.

Here is a summary of features:

- A non-User (email is undisclosed) can write to a User and get a reply (can be disabled by configuration)
- Exchanges can be moderated (with auto-accept and auto-reject plug-ins)
- Optional recipient filter plug-ins
- Optional exchange filtering plug-ins (blacklists)
- Multi-recipient writing is possible (can be disabled by configuration) with min/max constraints
- Messages are managed by conversations
- Messages in folders are sortable by sender|recipient|subject|date
- 'Archives' folder in addition to classic Inbox, Sent and Trash folders
- A Quick-Reply form to only ask for a response text
- A cleanup management command to clear the old deleted messages

It has support for optional additional applications:

- Autocomplete recipient field (default is 'django-ajax-selects'), with multiple recipient management
- New message notification (default is django-notification)
- Asynchronous mailer (default is django-mailer)

CHAPTER 1

Moderation

As an option, messages may need to be validated by a moderator before to be visible to the recipient. Possible usages are:

- to control there is no unwanted words in the text fields.
- to make sure that no direct contact informations are exchanged when the site is an intermediary and delivers services based on subscription fees.

Messages are first created in a *pending* state. A moderator is in charge to change them to a *rejected* or *accepted* state. This operation can be done in two ways:

- By a person, through the Admin site. A specially simplified change view is provided, with one-click buttons to accept or reject the message.
- Automatically, through one or more auto-moderator functions.

CHAPTER 2

Filters

As options, custom filters can disallow messages, in two ways:

- **user filter**: a user is not in a state to act as a recipient
- **exchange filter**: criteria for a message between a specific sender and a specific recipient are not fulfilled

Contents:

2.1 Quick start guide

2.1.1 Requisites and dependances

Python version >= 2.6 or >= 3.3

Some reasons:

- (2.6) use of `str.format()`

Django version >= 1.5 on py2, >= 1.5.5 on py3

Some reasons:

- (1.5/py2) `url` template tag syntax
- (1.5.5/py3) Six version >= 1.4.0
- (1.4.2) use of the Six library for supporting Python 2 and 3 in a single codebase

2.1.2 Installation

Get the code from the repository, which is hosted at Bitbucket.

You have two main ways to obtain the latest code and documentation:

With the version control software Mercurial installed, get a local copy by typing:

```
hg clone http://bitbucket.org/psam/django-postman/
```

Or download a copy of the package, which is available in several compressed formats, either from the `Download` tab or from the `get source` menu option.

In both case, make sure the directory is accessible from the Python import path.

2.1.3 Configuration

Required settings

Add `postman` to the `INSTALLED_APPS` setting of your project.

Run a **`manage.py migrate`** (or for Django <= 1.6 **`manage.py syncdb`**)

Include the URLconf `postman.urls` in your project's root URL configuration.

Optional settings

If you want to make use of a `postman_unread_count` context variable in your templates, add `postman.context_processors.inbox` to the `TEMPLATE_CONTEXT_PROCESSORS` setting of your project.

You may specify some additional configuration options in your `settings.py`:

`POSTMAN_I18N_URLS` *New in version 3.5.0.*

Set it to True if you want the internationalization of URL patterns. Translations are provided by the language files.

Defaults to: False.

`POSTMAN_DISALLOW_ANONYMOUS` Set it to True if you do not allow visitors to write to users. That way, messaging is restricted to a User-to-User exchange.

Defaults to: False.

`POSTMAN_DISALLOW_MULTIRECIPIENTS` Set it to True if you do not allow more than one username in the recipient field.

Defaults to: False.

`POSTMAN_DISALLOW_COPIES_ON_REPLY` Set it to True if you do not allow additional recipients when replying.

Defaults to: False.

`POSTMAN_DISABLE_USER_EMAILING` Set it to True if you do not want basic email notification to users. This setting does not apply to visitors (refer to `POSTMAN_DISALLOW_ANONYMOUS`), nor to a notifier application (refer to `POSTMAN_NOTIFIER_APP`)

Defaults to: False.

`POSTMAN_FROM_EMAIL` *New in version 3.6.0.*

Set it if you want to override the default 'from' field value.

Defaults to: DEFAULT_FROM_EMAIL.

POSTMAN_PARAMS_EMAIL *New in version 3.6.0.*

You can customize the sending of emails by this means. The value is a function, receiving one parameter: a dictionary with the same context variables as for the subject and body template rendering: {'site': ..., 'object': ..., 'action': ...}. The return must be a dictionary, possibly empty, with django.core.mail.EmailMessage parameters as keys.

Defaults to: None.

Example:

```
def get_params_email(context):
    return {
        'reply_to': ['someone@domain.tld'],
        'headers': {'X-my-choice': 'my-value'}
    } if context['action'] == 'acceptance' else {}
POSTMAN_PARAMS_EMAIL = get_params_email  # default is None
```

Notes:

- 'reply_to' is available as of Django 1.8. For previous versions, you can embed it under 'headers' as: `{'Reply-To': 'someone@domain.tld'}`
- In case of use of django-mailer (v1.2.2), only 'headers' is supported and to the condition that a HTML-version email template is involved.

POSTMAN_AUTO_MODERATE_AS The default moderation status when no auto-moderation functions, if any, were decisive.

- `True` to accept messages.
- `False` to reject messages.
- `None` to leave messages to a moderator review.

Defaults to: None.

To disable the moderation feature (no control, no filter):

- Set this option to True
- Do not provide any auto-moderation functions

POSTMAN_SHOW_USER_AS How to represent a User for display, in message properties: `obfuscated_recipient` and `obfuscated_sender`, and in the `or_me` filter. The value can be specified as:

- The name of a property of User. For example: 'last_name'.
- The name of a method of User. For example: 'get_full_name'.
- A function, receiving the User instance as the only parameter. For example: `lambda u: u.get_profile().nickname`.
- *New in version 3.3.0.* The full path to a function, as a string, whose import will be deferred. For example: 'myapp.mymodule.myfunc'. The function is given the User object as the only parameter. This sort of reference can be useful when resolving circular import dependencies between applications or modules. Another approach, not promoted but compatible, is to specify a class instead of a function, like 'myapp.mymodule.MyClass'. In that case, an instance of the class is initialized with the User object and its representation is the final result.
- `None` : the default text representation of the User (username) is used.

Defaults to: None.

The default behaviour is used as a fallback when: the value names an attribute and the result is false (misspelled attribute name, empty result, ...), or the value names a function and an exception is raised (but any result, even empty, is valid).

POSTMAN_NAME_USER_AS *New in version 3.3.0.*

How to name a User as a recipient. The value can be specified as:

- The name of a property of User. For example: 'last_name' (in auth.User) or 'nick_name' (in a Custom User Model).
- `None` : the default User model attributes are used: USERNAME_FIELD and get_username().

Defaults to: None.

POSTMAN_QUICKREPLY_QUOTE_BODY *New in version 3.2.0.*

Set it to True if you want the original message to be quoted when replying directly from the display view. This setting does not apply to the reply view in which quote is the basic behaviour.

Defaults to: False.

POSTMAN_NOTIFIER_APP A notifier application name, used in preference to the basic emailing, to notify users of their rejected or received messages.

Defaults to: 'notification', as in django-notification.

Note: django-notification v0.2.0 works with Django version 1.3. As of Django 1.4, switch to at least django-notification v1.0.

If you already have a notifier application with the default name in the installed applications but you do not want it to be used by this application, set the option to None.

POSTMAN_MAILER_APP An email application name, used in preference to the basic django.core.mail, to send emails.

Defaults to: 'mailer', as in django-mailer.

If you already have a mailer application with the default name in the installed applications but you do not want it to be used by this application, set the option to None.

POSTMAN_AUTOCOMPLETER_APP An auto-completer application specification, useful for recipient fields. To enable the feature, define a dictionary with these keys:

- **'name'** The name of the auto-completer application. Defaults to 'ajax_select'
- **'field'** The model class name. Defaults to 'AutoCompleteField'
- **'arg_name'** The name of the argument Defaults to 'channel'
- **'arg_default'** No default value. This is a mandatory default value, but you may supersede it in the field definition of a custom form or pass it in the url pattern definitions.

Defaults to: an empty dictionary.

Templates

A complete set of working templates is provided with the application. You may use it as it is with a CSS design of yours, re-use it or extend some parts of it, or only view it as an example.

Don't forget that you shouldn't modify the templates provided into the package (changes are lost with an application update) but use a copied set pointed to by the `DIRS` entry in TEMPLATES setting.

You may need to adjust some templates to match your version of Django. Permute the comment tags for the lines denoted by the marks: {# dj v1.x #} in:

- (currently no case)

Relations between templates:

```
base.html
|_ base_folder.html
|  |_ inbox.html
|  |_ sent.html
|  |_ archives.html
|  |_ trash.html
|_ base_write.html
|  |_ write.html
|  |_ reply.html
|_ view.html
```

The `postman/base.html` template extends a `base.html` site template, in which some blocks are expected:

- title: in <html><head><title>, at least for a part of the entire title string
- extrahead: in <html><head>, to put some <script> and <link> elements
- content: in <html><body>, to put the page contents
- postman_menu: in <html><body>, to put a navigation menu

Static Files

A CSS file is provided with the application, for the Admin site: `postman/css/admin.css`. It is not obligatory but makes the display more confortable.

A basic CSS file is provided to style the views: `postman/css/postman.css`. You may use it as a starting point to make your own design.

These files are provided under `postman/static/`.

See also *styles* for the stylesheets of views.

For Django 1.3+, just follow the instructions related to the staticfiles app.

2.1.4 Examples

`settings.py`:

```
INSTALLED_APPS = (
    # 'pagination'  # has to be before postman
    # ...
    'postman',
    # ...
    # 'ajax_select'
    # 'notification'
    # 'mailer'
)
# POSTMAN_I18N_URLS = True  # default is False
# POSTMAN_DISALLOW_ANONYMOUS = True  # default is False
# POSTMAN_DISALLOW_MULTIRECIPIENTS = True  # default is False
# POSTMAN_DISALLOW_COPIES_ON_REPLY = True  # default is False
```

```
# POSTMAN_DISABLE_USER_EMAILING = True  # default is False
# POSTMAN_FROM_EMAIL = 'from@host.tld'  # default is DEFAULT_FROM_EMAIL
# POSTMAN_PARAMS_EMAIL = get_params_email  # default is None
# POSTMAN_AUTO_MODERATE_AS = True  # default is None
# POSTMAN_SHOW_USER_AS = 'get_full_name'  # default is None
# POSTMAN_NAME_USER_AS = 'last_name'  # default is None
# POSTMAN_QUICKREPLY_QUOTE_BODY = True  # default is False
# POSTMAN_NOTIFIER_APP = None  # default is 'notification'
# POSTMAN_MAILER_APP = None  # default is 'mailer'
# POSTMAN_AUTOCOMPLETER_APP = {
    # 'name': '',  # default is 'ajax_select'
    # 'field': '',  # default is 'AutoCompleteField'
    # 'arg_name': '',  # default is 'channel'
    # 'arg_default': 'postman_friends',  # no default, mandatory to enable the feature
# }  # default is {}
```

urls.py:

```
url(r'^messages/', include('postman.urls', namespace='postman', app_name='postman')),
```

2.2 Moderation

When created, a message is in a *pending* state. It is not delivered to the recipient immediately. By default, some person must review its contents and must either accept or reject the message.

Moderation is done through the Admin site. To ease the action, a special message type is available: PendingMessage. It's nothing else but the classic Message type, but:

- It is intended to collect only messages in the *pending* state
- A dedicated simplified change view is available, with two main buttons: Accept and Reject

The moderator can give a reason in case of rejection of the message. If provided, this piece of information will be reported in the notification to the sender.

2.2.1 Auto moderators

You may automate the moderation by giving zero, one, or many auto-moderator functions to the views. The value of the parameter can be one single function or a sequence of functions as a tuple or a list.

Views supporting an `auto_moderators` parameter are: `WriteView`, `ReplyView`.

Example:

```
def mod1(message):
    # ...
    return None

def mod2(message):
    # ...
    return None
mod2.default_reason = 'mod2 default reason'

urlpatterns = patterns('postman.views',
    # ...
    url(r'^write/(?:(?P<recipients>[^/#]+)/)?$',
```

```
        WriteView.as_view(auto_moderators=(mod1, mod2)),
        name='write'),
    url(r'^reply/(?P<message_id>[\d]+)/$',
        ReplyView.as_view(auto_moderators=mod1),
        name='reply'),
    # ...
)
```

Each auto-moderator function will be called for the message to moderate, in the same order as the one set in the parameter.

Input:

- `message`: a Message instance

Output:

The structure of the output is either a `rating` or a tuple `(rating, reason)`.

`rating` may take the following values:

- `None`
- 0 or `False`
- 100 or `True`
- an integer between 1 and 99

`reason` is a string, giving a specific reason for a rejection. If not provided, a default reason will be taken from the `default_reason` attribute of the function, if any. Otherwise, there will be no reason.

The processing of the chain of auto-moderators is managed by these rules:

1. If return is `None` or outside the range 0..100, the auto-moderator is neutral
2. If return is 0, no other function is processed, the message is rejected
3. If return is 100, no other function is processed, the message is accepted
4. Otherwise, the rating will count for an average among the full set of returned ratings

At the end of the loop, if the decision is not final, the sequence is:

1. If there was no valid rating at all, then the `POSTMAN_AUTO_MODERATE_AS` setting applies.
2. An average rating is computed: if greater or equal to 50, the message is accepted.
3. The message is rejected. The final reason is a comma separated collection of reasons coming from moderators having returned a rating lesser than 50.

2.3 Notification

Parties should be notified of these events:

- when a message is rejected (sender)
- when a message or a reply is received (recipient)

New in version 3.4.0 With the default templates provided within the application, a text-only email is sent. If you want the email to be in HTML format:

1. Create the .html version.

2. Override the .txt version (otherwise the template provided by default applies) with either:
    - a non empty content of your design.
    - or an empty content, meaning by convention a fallback to the .html content with all tags stripped.

2.3.1 For visitors

An email is sent, using these templates:

- `postman/email_visitor_subject.txt` for the subject
- `postman/email_visitor.txt` and/or `.html` for the body

The available context variables are:

- `site`: the *Site* instance if the "sites" framework is installed, otherwise a *RequestSite* instance with a fallback to *None* in the case of the API
- `object`: the Message instance
- `action`: 'rejection' or 'acceptance'

Default templates are provided with the application. Same as for the views, you can override them, and design yours.

2.3.2 For users

Special case: In case of a rejection by the auto moderation feature, the user is immediately aware of it, so there is no need for a notification in addition.

If a notifier application is configured (see *Optional settings*), the following labels are used:

- `postman_rejection` to notify the sender of the rejection
- `postman_message` to notify the recipient of the reception of a message
- `postman_reply` to notify the recipient of the reception of a reply

Some extra context variables are passed in the call to the notifier application and so are available in the templates:

- `pm_message`: the Message instance
- `pm_action`: 'rejection' or 'acceptance'
- `pm_site`: the Site instance

If no notifier application is used, an email is sent, using these templates:

- `postman/email_user_subject.txt` for the subject
- `postman/email_user.txt` and/or `.html` for the body

In that case, the information about context variables and templates is the same as in the *For visitors* section above.

2.4 Custom views

2.4.1 styles

Here is a sample of some CSS rules, usable for `postman/views.html`:

```
.pm_message.pm_deleted              { text-decoration: line-through; }
.pm_message.pm_deleted .pm_body     { display: none; }
.pm_message.pm_archived             { font-style: italic; color: grey; }
.pm_message.pm_unread .pm_subject   { font-weight: bolder; }
.pm_message.pm_pending .pm_header   { background-color: #FFC; }
.pm_message.pm_rejected .pm_header  { background-color: #FDD; }
```

These rules are provided with the application, as an example, in a static file (See *Static Files*).

2.4.2 forms

You can replace the default forms in views.

Examples:

```
urlpatterns = patterns('postman.views',
    # ...
    url(r'^write/(?:(?P<recipients>[^/#]+)/)?$',
        WriteView.as_view(form_classes=(MyCustomWriteForm,␣
↪MyCustomAnonymousWriteForm)),
        name='write'),
    url(r'^reply/(?P<message_id>[\d]+)/$',
        ReplyView.as_view(form_class=MyCustomFullReplyForm),
        name='reply'),
    url(r'^view/(?P<message_id>[\d]+)/$',
        MessageView.as_view(form_class=MyCustomQuickReplyForm),
        name='view'),
    # ...
)
```

2.4.3 templates

You can replace the default template name in all views.

Example:

```
urlpatterns = patterns('postman.views',
    # ...
    url(r'^view/(?P<message_id>[\d]+)/$',
        MessageView.as_view(template_name='my_custom_view.html'),
        name='view'),
    # ...
)
```

2.4.4 after submission

You can supersede the default view where to return to, after a successful submission.

The default algorithm is:

1. Return where you came from
2. If it cannot be known, fall back to the inbox view
3. But if the submission view has a `success_url` parameter, use it preferably

4. In all cases, a `next` parameter in the query string has higher precedence

The parameter `success_url` is available to these views:

- `WriteView`
- `ReplyView`
- `ArchiveView`
- `DeleteView`
- `UndeleteView`

Example:

```
urlpatterns = patterns('postman.views',
    # ...
    url(r'^reply/(?P<message_id>[\d]+)/$',
        ReplyView.as_view(success_url='postman:inbox'),
        name='reply'),
    # ...
)
```

Example:

```
<a href="{% url 'postman:reply' reply_to_pk %}?next={{ next_url|urlencode }}">Reply</
↪a>
```

2.4.5 reply formatters

You can replace the default formatters used for replying.

Examples:

```
def format_subject(subject):
    return "Re_ " + subject

def format_body(sender, body):
    return "{0} _ {1}".format(sender, body)

urlpatterns = patterns('postman.views',
    # ...
    url(r'^reply/(?P<message_id>[\d]+)/$',
        ReplyView.as_view(formatters=(format_subject, format_body)),
        name='reply'),
    url(r'^view/(?P<message_id>[\d]+)/$',
        MessageView.as_view(formatters=(format_subject, format_body)),
        name='view'),
    # ...
)
```

See also:

- the `POSTMAN_QUICKREPLY_QUOTE_BODY` setting in *Optional settings*

2.5 Features

2.5.1 Direct write to

In the pages of your site, you can put links containing the recipient name(s).

Example:

```
<a href="{% url 'postman:write' username %}">write to {{ username }}</a>
```

Separate multiple usernames with a `:` character.

Example:

```
<a href="{% url 'postman:write' 'adm1:adm2:adm3' %}">write to admins</a>
```

2.5.2 Prefilled fields

You may prefill the contents of some fields by providing a query string in the link.

Example:

```
<a href="{% url 'postman:write' %}?subject=details request&body=give me details about␣
↪...">
ask for details
</a>
```

2.5.3 Recipients Min/Max

If you need to constraint the maximum number of recipients in the forms, you can pass the optional `max` parameter to the view. There is no parameter for a minimum number, but you can code a custom form and pass a `min` parameter to the recipient field (see Advanced Usage below for details).

Views supporting the parameter are: `WriteView`, `ReplyView`.

But this parameter does not apply to the default `AnonymousWriteForm` for visitors: The maximum is enforced to 1 (see Advanced Usage below for knowing how), in order to keep the features available to anonymous users to a strict minimum.

Example:

```
urlpatterns = patterns('postman.views',
    # ...
    url(r'^write/(?:(?P<recipients>[^/#]+)/)?$',
        WriteView.as_view(max=3),
        name='write'),
    # ...
)
```

Advanced usage

If you define your own custom form, you may specify a `min` parameter and a `max` parameter to the recipients field.

For example:

```
from postman.forms import WriteForm
class MyWriteForm(WriteForm):
    recipients = CommaSeparatedUserField(label="Recipients", min=2, max=5)
```

If you do not want the fixed `max` parameter of the recipients field in your custom form, to be superseded by the parameter passed to the view, set the `can_overwrite_limits` form attribute to `False`.

For example:

```
class MyThreeAnonymousWriteForm(MyBaseAnonymousWriteForm):
    can_overwrite_limits = False
    recipients = CommaSeparatedUserField(label="Recipients", max=3)
```

See also:

- the `POSTMAN_DISALLOW_MULTIRECIPIENTS` setting in *Optional settings*

2.5.4 User filter

If there are some situations where a user should not be a recipient, you can write a filter and pass it to the view.

Views supporting a user filter are: `WriteView`, `ReplyView`.

Example:

```
def my_user_filter(user):
    if user.get_profile().is_absent:
        return "is away"
    return None

urlpatterns = patterns('postman.views',
    # ...
    url(r'^write/(?:(?P<recipients>[^/#]+)/)?$',
        WriteView.as_view(user_filter=my_user_filter),
        name='write'),
    # ...
)
```

The filter will be called for each recipient, for validation.

Input:

- `user`: a User instance, as the recipient of the message

Output:

If the recipient is allowed, just return `None`.

To forbid the message, use one of these means:

- return `False` or `''`, if you do not want to give a reason for the refusal. The error message will be: "Some usernames are rejected: foo, bar."
- return a string, as a reason for the refusal. The error message will be: "Some usernames are rejected: foo (reason), bar (reason)."
- raise a `ValidationError` with an error message to your liking.

Advanced usage

If you define your own custom form, you may specify a user filter inside.

For example:

```
def my_user_filter(user):
    # ...
    return None

from postman.forms import WriteForm
class MyWriteForm(WriteForm):
    recipients = CommaSeparatedUserField(label="Recipients", user_filter=my_user_
↪filter)
```

2.5.5 Exchange filter

If there are some situations where an exchange should not take place, you can write a filter and pass it to the view. Typical usages would be: blacklists, users that do not want solicitation from visitors.

Views supporting an exchange filter are: `WriteView`, `ReplyView`.

An example, with the django-relationships application:

```
def my_exchange_filter(sender, recipient, recipients_list):
    if recipient.relationships.exists(sender, RelationshipStatus.objects.blocking()):
        return "has blacklisted you"
    return None

urlpatterns = patterns('postman.views',
    # ...
    url(r'^write/(?:(?P<recipients>[^/#]+)/)?$',
        WriteView.as_view(exchange_filter=my_exchange_filter),
        name='write'),
    # ...
)
```

The filter will be called for each couple, to validate that the exchange is possible.

New in version 3.3.0 In the case of a reply, there is an additional call for the implicit recipient when it is a User. The value of the `recipients_list` parameter allows to differentiate the context.

Inputs:

- `sender`: a User instance, as the sender of the message, or None if the writer is not authenticated
- `recipient`: a User instance, as the recipient of the message
- `recipients_list`: the full list of recipients or (*New in version 3.3.0*) None in the case of the implicit recipient for a reply. Provided as a convenient additional element of decision.

Output:

If the exchange is allowed, just return `None`.

To forbid the exchange, use one of these means:

- return `False` or `''`, if you do not want to give a reason for the refusal. The error message will be: "Writing to some users is not possible: foo, bar."

- return a string, as a reason for the refusal. The error message will be: "Writing to some users is not possible: foo (reason), bar (reason)."
- raise a `ValidationError` with an error message to your liking.

Advanced usage

If you define your own custom form, you may specify an exchange filter inside.

For example:

```
def my_exchange_filter(sender, recipient, recipients_list):
    # ...
    return None

from postman.forms import WriteForm
class MyWriteForm(WriteForm):
    exchange_filter = staticmethod(my_exchange_filter)
```

2.5.6 Auto-complete field

An auto-complete functionality may be useful on the recipients field.

To activate the option, set at least the `arg_default` key in the `POSTMAN_AUTOCOMPLETER_APP` dictionary. If the default `ajax_select` application is used, define a matching entry in the `AJAX_LOOKUP_CHANNELS` dictionary.

Example:

```
AJAX_LOOKUP_CHANNELS = {
    'postman_users': dict(model='auth.user', search_field='username'),
}
POSTMAN_AUTOCOMPLETER_APP = {
    'arg_default': 'postman_users',
}
```

Don't forget that not-custom channels are restricted to users having the `is_staff` property.

In case of version 1.1.4/5 of django-ajax-selects:

> Support for multiple recipients is not turned on by default by django-ajax-selects. To allow this capability, you have to pass the option `multiple: true` to jquery-plugin-autocomplete.
>
> Make your own templates, based on these two files, given as implementation examples:
>
> - `postman/templates/autocomplete_postman_multiple_as1-1.html`
> - `postman/templates/autocomplete_postman_single_as1-1.html`
>
> These examples include a correction necessary for the support of the 'multiple' option.

In case of version 1.2.x of django-ajax-selects:

> Refer to the installation guide of this application, in particular the use of AJAX_SELECT_BOOTSTRAP and AJAX_SELECT_INLINES. Support for multiple recipients is not as simple as an option: see the examples in the jQuery UI demos.
>
> You can use the following working implementation example:
>
> - `postman/templates/autocomplete_postman_multiple_as1-2.html`

New in version 3.3.0 In case of version 1.3.x of django-ajax-selects:

To make your own `templates/autocomplete.html` or `templates/autocomplete_<channel>.html`, you can use the following working implementation example:

- `postman/templates/autocomplete_postman_multiple_as1-3.html`

Customization

You may attach a specific channel, different from the default one, to a particular view.

Views supporting an auto-complete parameter are: `WriteView`, `ReplyView`.

For the `WriteView` view, the parameter is named `autocomplete_channels` (note the plural). It supports two variations:

- a 2-tuple of channels names: the first one for authenticated users, the second for visitors. Specify `None` if you let the default channel name for one of the tuple parts.
- a single channel name: the same for users and visitors

For the `ReplyView` view, the parameter is named `autocomplete_channel` (note the singular). The value is the channel name.

Example:

```
urlpatterns = patterns('postman.views',
    # ...
    url(r'^write/(?:(?P<recipients>[^/#]+)/)?$',
        WriteView.as_view(autocomplete_channels=(None,'anonymous_ac')),
        name='write'),
    url(r'^reply/(?P<message_id>[\d]+)/$',
        ReplyView.as_view(autocomplete_channel='reply_ac'),
        name='reply'),
    # ...
)
```

Example:

```
urlpatterns = patterns('postman.views',
    # ...
    url(r'^write/(?:(?P<recipients>[^/#]+)/)?$',
        WriteView.as_view(autocomplete_channels='write_ac'),
        name='write'),
    # ...
)
```

Advanced usage

If you define your own custom form, you may specify an autocomplete channel inside.

For example:

```
from postman.forms import WriteForm
class MyWriteForm(WriteForm):
    recipients = CommaSeparatedUserField(label="Recipients", channel='my_channel')
```

2.6 Tags and Filters

The following tags and filters are available to your templates by loading the library:

```
{% load postman_tags %}
```

Here are the other special libraries in the `postman/templatetags/` directory, that are not intended for your site design:

- `postman_admin_modify.py`: a library exclusively designed for a customized change_form template used in the Admin site for the moderation of pending messages.
- `pagination_tags.py`: a mock of the django-pagination application template tags. For convenience, the design of the default template set is done with the use of that application. The mock will avoid failures in template rendering if the real application is not installed, as it may be the case for the test suite run in a minimal configuration. To deactivate the mock and use the real implementation, just make sure that `pagination` is declared before `postman` in the `INSTALLED_APPS` setting.

2.6.1 Tags

postman_unread

Gives the number of unread messages for a user. Returns nothing (an empty string) for anonymous users.

Storing the count in a variable for further processing is advised, such as:

```
{% postman_unread as unread_count %}
...
{% if unread_count %}
    You have <strong>{{ unread_count }}</strong> unread messages.
{% endif %}
```

postman_order_by

Returns a formatted GET query string, usable to have the messages list presented in a specific order. This string must be put in the href attribute of a <a> HTML tag.

One argument is required: a keyword to specify the field used for the sort. Supported values are:

- sender
- recipient
- subject
- date

If the list is already sorted by the keyword, the returned value will specify the reversed order. If there are other existing parameters, such as a page number, they are preserved in the resulting output.

Example:

```
<a href="{% postman_order_by subject %}">...</a>
```

2.6.2 Filters

or_me

If the value is equal to the argument, replace it with the constant string '<me>'.

For example, if we have:

```
{{ message.obfuscated_sender|or_me:user }}
```

and the sender is the currently logged-in user, the output is compacted to show only the simple pattern '<me>'.

Note that this pattern cannot be confused with the username of a real user, because the brackets are not in the valid character set for a username.

compact_date

Output a date as short as possible. The argument must provide three date format patterns. The pattern used depends on how the date compares to the current instant:

- pattern 1 if in the same day
- pattern 2 if in the same year
- pattern 3 otherwise

For example:

```
{{ message.sent_at|compact_date:_("g:i A,M j,n/j/y") }}
```

With a message sent on "5 dec 2010, 09:21:58":

for the day:	the output is:
5 dec 2010	9:21 AM
6 dec 2010	Dec 5
1 jan 2011	12/5/10

2.7 Management Commands

2.7.1 postman_cleanup

When a user deletes a message, the object is not deleted from the database right away, it is moved to a `trash` folder. One reason is to allow a message to be undeleted if the user wants to retrieve it. Another reason is that there is only one copy of a message for both the sender and the recipient, so the message must be marked for deletion by the two parties before to be considered for a withdraw. An additional constraint is that a message may be a member of a conversation and the reply chain must be kept consistent.

So there are some criteria to fulfill by a record to be really deleted from the database:

- both the sender and the recipient must have marked the message as deleted
- if the message is in a conversation, all the messages of the conversation must be marked for deletion
- the action of deletion must have been done enough time ago

A management command is provided for this purpose:

`django-admin.py postman_cleanup`

It can be run as a cron job or directly.

The `--days` option can be used to specify the minimal number of days a message/conversation must have been marked for deletion. Default value is 30 days.

2.7.2 postman_checkup

A management command to run a test suite on the messages presently in the database. It checks messages and conversations for possible inconsistencies, in a read-only mode. No change is made on the data.

`django-admin.py postman_checkup`

It can be run directly or better as a nightly cron job.

2.8 API

For an easier usage of the application from other applications in the project, an API is provided.

Note: The "sites" framework is optional but you need it if you want the `site` context variable to provide a *Site* instance (not *None*) in the notification templates.

2.8.1 pm_broadcast()

Broadcast a message to multiple Users.

For an easier cleanup, all these messages are directly marked as archived and deleted on the sender side. The message is expected to be issued from a trusted application, so moderation is not necessary and the status is automatically set to 'accepted'.

Arguments: (sender, recipients, subject, body='', skip_notification=False)

2.8.2 pm_write()

Write a message to a User.

Contrary to pm_broadcast(), the message is archived and/or deleted on the sender side only if requested. The message may come from an untrusted application, a gateway for example, so it may be useful to involve some auto moderators in the processing.

Arguments: (sender, recipient, subject, body='', skip_notification=False, auto_archive=False, auto_delete=False, auto_moderators=[])

Return: The created Message instance.

2.8.3 Arguments

- `auto_archive`: to mark the message as archived on the sender side
- `auto_delete`: to mark the message as deleted on the sender side
- `auto_moderators`: a list of auto-moderation functions

- `body`: the contents of the message
- `recipient`: a User instance
- `recipients`: a list or tuple of User instances, or a single User instance
- `sender`: a User instance
- `skip_notification`: if the normal notification event is not wished
- `subject`: the subject of the message

2.8.4 Example

Suppose an application managing Event objects. Whenever a new Event is generated, you want to broadcast an announcement to Users who have subscribed to be informed of the availability of such a kind of Event.

Code sample:

```
from postman.api import pm_broadcast
events = Event.objects.filter(...)
for e in events:
    pm_broadcast(
        sender=e.author,
        recipients=e.subscribers,
        subject='New {0} at Our School: {1}'.format(e.type, e.title),
        body=e.description)
```

2.9 Frequently-asked questions

2.9.1 General

I don't want to bother with the moderation feature, how to bypass it? Set the configuration option:

```
POSTMAN_AUTO_MODERATE_AS = True
```

I installed django-pagination, and still I don't see any pagination widgets

- Is there really more messages than one page capacity (default is 20)?
- Check that `pagination` is declared before `postman` in the `INSTALLED_APPS` setting.
- See if it's better by disabling `postman/templatetags/pagination_tags.py` and `.pyc` (rename or move the files).

CHAPTER 3

Indices and tables

- genindex
- modindex
- search