django-werewolf Documentation

Release 0.4.2

Artur Barseghyan <artur.barseghyan@gmail.com>

November 29, 2013

django-werewolf

CHAPTER 1

Description

Item publishing workflow for Django (fully integrated into Django admin).

CHAPTER 2

Prerequisites

- Django 1.5.+
- Python 2.7.+, 3.3.+

CHAPTER 3

Installation

1. Install django-werewolf into your virtual environment:

   $ pip install django-werewolf

2. Add *werewolf* to your `INSTALLED_APPS`.

That's all. See the *Usage and examples* section for more.

CHAPTER 4

Usage and examples

It's all about item publishing in a workflow. We have various *intermediate* statuses (work in-progress) and a final *status* which means that the item is actually published. Some users should be able to set the item status to *published*, some others not. This app allows you (and gives you a good working example with pre-configured Django environment) to write a custom workflow for publishing your items with minimal efforts.

For a complete example of a working django-werewolf app see the (https://github.com/barseghyanartur/django-werewolf/tree/stable/example) and read the *readme.rst* of the *news* app.

4.1 Imaginary app concept

There are three user groups. All of them should be able to log into Django admin.

- Writers
- Editors
- Chief Editors

In short, our imaginary app would work as follows.

- Chief Editor creates a News item and chooses a Writer and an Editor. The status of a new News item is then set to *new*.
- Once a News item with status *new* has been created, both Writer and the Editor assigned do get an e-mail notification about the fact that a News item has been assigned to them.
- Writer is supposed to fill the assigned News item with content and once the News item is ready, change its' status to *ready*.
- The assigned Editor would get an e-mail notification about the fact that the News item has been changed to *ready*.
- The assigned Editor is supposed to check the News item with status *ready* and if it's acceptable, change the News item status to *reviewed*.
- Once a News item status has been set to *reviewed*, the assigned Writer can no longer access it in the Django admin.
- The assigned Chief Editor would get an e-mail notification about the fact that the News item has been changed to *reviewed.*
- The assigned Chief Editor is supposed to check the News item with status *reviewed* and if it acceptable, change the News item status to *published*.

- Once a News item status has been set to *published*, the assigned Editor can no longer access it in the Django admin.
- Once a News item status has been changed to *published*, all Chief Editors in the system, as well as the assigned Writer and Editor get an e-mail notification about the fact that the News item has been published.

4.2 Demo

In order to be able to quickly evaluate the django-werewolf, a demo app (with a quick installer) has been created (Debian only). Follow the instructions below for having the demo running within a minute.

Grab the latest *django-werewolf-example-app-install.sh*

> $ wget https://raw.github.com/barseghyanartur/django-werewolf/stable/django-werewolf-example-app-install.sh

Create a new- or switch to existing- virtual environement, assign execute rights to the installer and run the *django-werewolf-example-app-install.sh*.

> $ chmod +x django-werewolf-example-app-install.sh

> $ ./django-werewolf-example-app-install.sh

Go to the backend and test the app.

- URL: http://127.0.0.1:8000/admin/news/newsitem/
- Admin username: admin
- Admin password: test
- Chief Editor username: chief_editor
- Chief Editor password: test
- Editor username: editor
- Editor password: test
- Writer username: writer
- Writer password: test

Let's now step-by-step review our imaginary example app.

4.3 settings.py

```
>>> # Workflow statuses; order is preserved.
>>> WEREWOLF_STATUS_CHOICES = (
>>>     ('new', gettext('New')), # New - this is how it's assigned to a writer.
>>>     ('draft', gettext('Draft')), # Draft - this is how the writer works on it.
>>>     ('ready', gettext('Ready')), # Ready to be reviewed by editor.
>>>     ('reviewed', gettext('Reviewed')), # Reviewed by editor (means positive
>>>                                        # and ready to be published).
>>>     ('published', gettext('Published')), # Published.
>>> )
>>>
>>> # Published status.
>>> WEREWOLF_STATUS_PUBLISHED = 'published'
>>>
```

```
>>> # When set to True, django-reversion is used.
>>> WEREWOLF_USE_DJANGO_REVERSION = True
```

4.4 news/models.py

In the example below we have a basic news item model. We have Chief Editors with full access to news items, we have editors with less privelleges and Writers with very little privelleges. Chief Editors create articles, select an Editor and a Writer (both get notified) and let them work on the article. Writers can only set an article status to *new*, *draft* and *ready* (ready to be checked). Editors review the articles with status *ready* and set the status to *reviewed*. Chief Editors publish articles that are *reviewed*. Your implementation can be as custom as you want it. Think in Django user groups (`django.contrib.auth.models.Group`) and Django permissions system.

NOTE: See the *Permission tuning* section.

```
>>> from django.contrib.auth.models import User
>>>
>>> from werewolf.models import WerewolfBaseModel, WerewolfBaseMeta
>>>
>>> _chief_editors = {'groups__name__iexact': 'Chief editors'}
>>> _editors = {'groups__name__iexact': 'Editors'}
>>> _writers = {'groups__name__iexact': 'Writers'}
>>>
>>> class NewsItem(WerewolfBaseModel): # Important!
>>>     title = models.CharField(_("Title"), max_length=100)
>>>     body = models.TextField(_("Body"))
>>>     date_published = models.DateTimeField(_("Date published"), \
>>>                                           default=datetime.datetime.now())
>>>     author = models.ForeignKey(User, verbose_name=_("Author"), \
>>>                                related_name='authors', \
>>>                                limit_choices_to=_writers)
>>>     editor = models.ForeignKey(User, verbose_name=_("Editor"), \
>>>                                related_name='editors', \
>>>                                limit_choices_to=_editors)
>>>     chief_editor = models.ForeignKey(User, verbose_name=_("Chief editor"), \
>>>                                      related_name='chief_editors', \
>>>                                      limit_choices_to=_chief_editors)
>>>
>>>     class Meta(WerewolfBaseMeta): # Important!
>>>         verbose_name = "News item"
>>>         verbose_name_plural = "News items"
```

Or if you want to define custom permissions for your model as well, do extend the django-werewolf permissions as follows:

```
>>> from werewolf.models import WerewolfBaseModel
>>> from werewolf.utils import extend_werewolf_permissions
>>>
>>> class NewsItem(WerewolfBaseModel):
>>>     # Your fields here
>>>     class Meta:
>>>         verbose_name = "News item"
>>>         verbose_name_plural = "News items"
>>>
>>>         # Important!
>>>         permissions = extend_werewolf_permissions(
>>>             ('can_change_author', _("Can change author")),
```

```
>>>             ('can_change_editor', _("Can change editor")),
>>>             ('can_change_chief_editor', _("Can change chief editor"))
>>>         )
```

4.5 news/admin.py

Basic admin for the news item model.

NOTE: See the *Permission tuning* section.

```
>>> from werewolf.admin import WerewolfBaseAdmin
>>>
>>> from news.models import NewsItem
>>>
>>> class NewsItemAdmin(WerewolfBaseAdmin):
>>>     werewolf_protected_fields = (
>>>         ('author', 'can_change_author'),
>>>         ('editor', 'can_change_editor'),
>>>         ('chief_editor', 'can_change_chief_editor')
>>>     )
>>>
>>> admin.site.register(NewsItem, NewsItemAdmin)
```

The `werewolf_protected_fields` property is a list of fields that are supposed to be protected. Each item in the list is a tuple of (`field_name_to_protect`, `required_permission`). If given, django-werewolf hides fields listed as protected from users that do not have the permission required. In order to do so, django-werewolf overrides the Django's ModelAdmin `get_field` and `get_fieldsets` methods. If you happen to override that method for your own needs, make sure the it also reflects the django-werewolf concepts.

NOTE: If you override the `queryset` method of your model's admin class, make sure to see the source code of *werewolf.admin.WerewolfBaseAdmin.queryset* and copy the approach from there. Otherwise, your users with no permission to change the *published* status will be able to chgange the status of already published items to non-published statuses.

4.6 news/views.py

```
>>> from news.models import NewsItem
>>>
>>> def browse(request):
>>>     news_items = NewsItem._default_manager.published()
>>>     # Other code
```

4.7 news/werewolf_triggers.py

In order to perform extra tasks on status change, triggers are used. You simply make a new file in your app called *werewolf_triggers.py* and define custom classes that should be called when a `status` field of your model changes to a certain value. Each trigger should subclass the `werewolf.triggers.WerewolfBaseTrigger` class.

```
>>> from werewolf.triggers import WerewolfBaseTrigger, registry
>>>
>>> class StatusNewTrigger(WerewolfBaseTrigger):
```

```
>>>     """
>>>     News item status changed to 'new'.
>>>     """
>>>     def process(self):
>>>         # Your code
>>>
>>> class StatusReadyTrigger(WerewolfBaseTrigger):
>>>     """
>>>     News item status changed to 'ready' (ready for review).
>>>     """
>>>     def process(self):
>>>         # Your code
>>>
>>> # Triggers status change to 'new' for news.newsitem model.
>>> registry.register('news', 'newsitem', 'new', StatusNewTrigger)
>>>
>>> # Triggers status change to 'ready' for news.newsitem model.
>>> registry.register('news', 'newsitem', 'ready', StatusReadyTrigger)
```

4.8 urls.py

In order to have triggers autodiscovered, place the following code into your main *urls* module.

```
>>> from werewolf import autodiscover as werewolf_autodiscover
>>> werewolf_autodiscover()
```

4.9 Permission tuning

Have in mind our `news.models.NewsItem` model.

1. Create three user groups:
   (a) Chief editors (permissions listed):
   - news | News item | Can add News item
   - news | News item | Can change author
   - news | News item | Can change chief editor
   - news | News item | Can change editor
   - news | News item | Can change News item
   - news | News item | Can change status to draft
   - news | News item | Can change status to new
   - news | News item | Can change status to published
   - news | News item | Can change status to ready
   - news | News item | Can change status to reviewed
   - news | News item | Can delete News item

   (a) Editors (permissions listed):
   - news | News item | Can change News item

- news | News item | Can change author
- news | News item | Can change status to draft
- news | News item | Can change status to new
- news | News item | Can change status to ready
- news | News item | Can change status to reviewed

(a) Writers (permissions listed):

- news | News item | Can change News item
- news | News item | Can change status to draft
- news | News item | Can change status to new
- news | News item | Can change status to ready

3. Create three users:
   - chief editor: Belongs to group *Chief editors*.
   - editor: Belongs to group *Editors*.
   - writer: Belongs to group *Writers*.
4. Now log into the admin with different user and see your admin for the *News item* (created items with *chiefeditor* account, then view them with *editor* and *writer*.

That's it. If somehow you don't see the new permissions (*Can change status to draft*, *Can change status to new*, etc) run a management command *syncww*:

```
$ ./manage.py syncww
```

CHAPTER 5

Running the example project

A working example of a django-werewolf app is available here: https://github.com/barseghyanartur/django-werewolf/tree/stable/example

1. Go to example/example directory

   $ cd example/example

2. Install requirements (in your virtual environment)

   $ pip install -r ../requirements.txt

3. Copy local_settings.example to local_settings.py

   $ cp local_settings.example local_settings.py

4. Create the database

   $ ./manage.py syncdb

5. Insert example test groups and users

   $ ./manage.py news_create_groups_and_test_users

6. Run the project

   $ ./manage.py runserver

CHAPTER 6

Documentation

Contents:

6.1 werewolf Package

6.1.1 werewolf Package

werewolf.__init__.**autodiscover**()

Autodiscovers the werewolf triggers in project apps. Each trigger file which should be found by werewolf, should be named "werewolf_triggers.py".

6.1.2 admin Module

class werewolf.admin.**WerewolfBaseAdmin**(**args*, ***kwargs*)

Bases: reversion.admin.VersionAdmin

Base werewolf admin model.

Property list werewolf_protected_fields List of fields to protect in form of the following tuple (field_name, required_permission).

formfield_for_dbfield(*db_field*, ***kwargs*)

Here we replace the choices based on the user permissions.

get_changelist_formset(*request*, ***kwargs*)

Removes protected fields from the list_editiable field list.

get_fieldsets(*request*, *obj=None*)

Hiding fields that non-authorised users should not have access to. It's done based on the werewolf_protected_fields of your ModelAdmin. But if happen to override that method for your own needs, make sure the it also reflects the django-werewolf concepts.

get_form(*request*, *obj=None*, ***kwargs*)

Hiding fields that non-authorised users should not have access to. It's done based on the werewolf_protected_fields of your ModelAdmin. But if happen to override that method for your own needs, make sure the it also reflects the django-werewolf concepts.

media

queryset(*request*)

Make sure users with no rights to edit an object with status, don't even see it.

save_model (*request*, *obj*, *form*, *change*)

status_change_trigger (*request*, *obj*, *form*, *change*)
Status change trigger. Executes appropriate registered trigger if applicable.

Parameters

- **request** (*django.http.HttpRequest*) –
- **obj** (*django.db.models.Model*) – Subclass of `django.db.models.Model`.
- **form** –
- **change** (*bool*) –

werewolf_protected_fields = []

6.1.3 helpers Module

werewolf.helpers.**admin_edit_url** (*app_label*, *module_name*, *object_id*, *url_title=None*)
Gets an admin edit URL for the object given.

Parameters

- **app_label** (*str*) –
- **module_name** (*str*) –
- **object_id** (*int*) –
- **url_title** (*str*) – If given, an HTML a tag is returned with *url_title* as the tag title. If left to None just the URL string is returned.

Return str

werewolf.helpers.**admin_edit_url_for_object** (*obj*, *url_title=None*)
Gets an admin edit URL for the object given.

Parameters

- **obj** (*django.db.models.Model*) – Django model subclass.
- **url_title** (*str*) – If given, an HTML a tag is returned with *url_title* as the tag title. If left to None just the URL string is returned.

Return str

6.1.4 triggers Module

class werewolf.triggers.**WerewolfBaseTrigger** (*obj*, *request*)
Bases: `object`

Werewolf base trigger.

6.1.5 utils Module

werewolf.utils.**permission_key** (*status*, *choice_key*)
Gets the permission key from `choice_key` given.

Parameters

- **status** (*str*) –

- **choice_key** (*str*) –

Return str

werewolf.utils.**permissions_for_base_model**(*permissions*=[])

Gets/extends permissions for the base model based on the `STATUS_CHOICES` defined.

Parameters **permissions** (*list|tuple*) – Permissions you want to have in your model. Those permissions would be extended by werewolf permissions.

Return list

werewolf.utils.**status_choices_for_user**(*user*, *app_label*)

Gets available status choices for the user given.

Parameters

- **user** (*django.contrib.auth.models.User*) – User for who the permissions are checked.
- **module_name** (*str*) – *app_label* of the model to check permissions to.

Return list List of choices in a same form as `werewolf.defaults.STATUS_CHOICES` but then limited to actual choices that user has permissions to.

werewolf.utils.**extend_werewolf_permissions**(**args*)

Extends model permissions with werewolf permissions.

Example

```
>>> from werewolf.models import WerewolfBaseModel
>>> from werewolf.utils import extend_werewolf_permissions
>>> class NewsItem(WerewolfBaseModel):
>>>     # Some fields here
>>>
>>>     class Meta:
>>>         verbose_name = _("News item")
>>>         verbose_name_plural = _("News items")
>>>
>>>         permissions = extend_werewolf_permissions(
>>>             ('can_change_author', _("Can change author")),
>>>             ('can_change_editor', _("Can change editor")),
>>>             ('can_change_chief_editor', _("Can change chief editor"))
>>>         )
```

6.1.6 Subpackages

management Package

Subpackages

commands Package

syncww Module

class werewolf.management.commands.syncww.**Command**

Bases: `django.core.management.base.BaseCommand`

args = '<app app ...>'

handle(**args*, ***options*)

help = 'reloads permissions for specified apps, or all apps if no args are specified'

models Package

models Package

class werewolf.models.**WerewolfBaseMeta**

Bases: object

Base Meta class of the WerewolfBaseModel. Every subclass of the WerewolfBaseModel shall extend it.

class werewolf.models.**WerewolfBaseModel**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

Base Werewolf model. If you want to have a workflow in your model (for statuses like new, draft, published, etc) you should extend this model.

managers Module

class werewolf.models.managers.**WerewolfBaseManager**

Bases: django.db.models.manager.Manager

Werewolf base manager.

CHAPTER 7

Indices and tables

- *genindex*
- *modindex*
- *search*

CHAPTER 8

License

GPL 2.0/LGPL 2.1

CHAPTER 9

Support

For any issues contact me at the e-mail given in the *Author* section.

CHAPTER 10

Author

Artur Barseghyan <artur.barseghyan@gmail.com>

W

werewolf.__init__, **??**
werewolf.admin, **??**
werewolf.helpers, **??**
werewolf.management.commands.syncww, **??**
werewolf.models, **??**
werewolf.models.managers, **??**
werewolf.triggers, **??**
werewolf.utils, **??**