django-slim Documentation

Release 0.7.3

Artur Barseghyan <artur.barseghyan@gmail.com>

October 22, 2014

Simple implementation of multi-lingual models for Django. Django-admin integration works out of the box. Supports *django-localeurl* integration.

CHAPTER 1

Prerequisites

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

CHAPTER 2

Installation

Note, that Django 1.5 is required. Earlier versions are not supported.

1. Installation

Latest stable version on PyPI:

    $ pip install django-slim

Latest stable version on bitbucket:

    $ pip install -e hg+https://bitbucket.org/barseghyanartur/django-slim@stable#egg=django-slim

Latest stable version on github:

    $ pip install -e git+https://github.com/barseghyanartur/django-slim/@stable#egg=django-slim

2. Add *slim* to `INSTALLED_APPS` of you settings module.

CHAPTER 3

Usage and examples

An extensive example project is available at https://github.com/barseghyanartur/django-slim/tree/stable/example directory.

Screenshots are present in documentation on PythonHosted (http://pythonhosted.org/django-slim/#screenshots).

3.1 Demo

In order to be able to quickly evaluate the django-slim, 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_slim_example_app_installer.sh*

> $ wget https://raw.github.com/barseghyanartur/django-slim/stable/example/django_slim_example_app_installer.sh

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

> $ chmod +x django_slim_example_app_installer.sh
>
> $ ./django_slim_example_app_installer.sh

Go to the front/back -end and test the app.

- Front-end URL: http://127.0.0.1:8001/en/foo/
- Admin URL: http://127.0.0.1:8001/admin/foo/fooitem/
- Admin username: admin
- Password: test

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

3.2 settings.py

Add *slim* to installed apps.

```
>>> INSTALLED_APPS = (
>>>     # ...
>>>     'slim',
>>>     # ...
>>> )
```

Add languages.

```
>>> LANGUAGES = (
>>>     ('en', gettext("English")), # Main language!
>>>     ('hy', gettext("Armenian")),
>>>     ('nl', gettext("Dutch")),
>>>     ('ru', gettext("Russian")),
>>> )
```

3.3 example/models.py

```
>>> from django.db import models
>>>
>>> from slim import LanguageField, Slim
>>>
>>> class FooItem(models.Model, Slim):
>>>     title = models.CharField(_("Title"), max_length=100)
>>>     slug = models.SlugField(unique=True, verbose_name=_("Slug"))
>>>     body = models.TextField(_("Body"))
>>>     language = LanguageField()
```

3.4 example/admin.py

```
>>> from django.contrib import admin
>>>
>>> from slim.admin import SlimAdmin
>>>
>>> class FooItemAdmin(SlimAdmin):
>>>     list_display = ('title',)
>>>     fieldsets = (
>>>         (None, {
>>>             'fields': ('title', 'slug', 'body')
>>>         }),
>>>     )
>>>
>>> admin.site.register(FooItem, FooItemAdmin)
```

3.5 example/views.py

We assume that language code is kept in the request object (django-localeurl behaviour, which you're advised to use).

```
>>> from slim import get_language_from_request
>>>
>>> from example.models import FooItem
>>>
>>> def browse(request, template_name='foo/browse.html'):
>>>     language = get_language_from_request(request)
>>>     queryset = FooItem._default_manager.filter(language=language)
>>>
>>>     # The rest of the code
```

3.6 More on ORM filtering

```
>>> from example.models import FooItem
>>> foo = FooItem._default_manager.all()[0]
<FooItem: Lorem ipsum>
```

Let's assume, we have such record and it has been translated to Armenian (*hy*) and Dutch (*nl*). Original translation is named *Lorem ipsum*. Other translations have the language code appended to the title.

```
>>> armenian_foo = foo.get_translation_for('hy')
<FooItem: Lorem ipsum HY>
>>> dutch_foo = foo.get_translation_for('nl')
<FooItem: Lorem ipsum NL>
```

If we have a translated object, we can always get the main translation.

```
>>> armenian_foo.original_translation == foo
True
```

All available translations for `foo`:

```
>>> foo.available_translations()
[<FooItem: Lorem ipsum HY>, <FooItem: Lorem ipsum NL>]
```

All available translations for Armenian `foo`.

```
>>> armenian_foo.available_translations()
[<FooItem: Lorem ipsum>, <FooItem: Lorem ipsum NL>]
```

See https://github.com/barseghyanartur/django-slim/tree/stable/example directory for a working example.

3.7 django-localeurl integration

3.7.1 Installation

django-localeurl integration is fully supported for Python 2.6.* and 2.7.* and installs automatically when installing django-slim. If you are using Python 3, install a forked version of django-localeurl (since official version does not yet have support for Python 3).

Forked version from bitbucket:

> $ pip install -e hg+https://bitbucket.org/barseghyanartur/django-localeurl@stable#egg=localeurl

3.7.2 Integration

Use *slim.models.decorators.auto_prepend_language* decorator in order to have it working.

Example (have in mind our *FooItem* model.

```
>>> from django.core.urlresolvers import reverse
>>>
>>> from slim.models.decorators import auto_prepend_language
>>>
>>> class FooItem(models.Model):
>>>     # Some other code; have in mind previous pieces.
>>>     @auto_prepend_language
```

```
>>>     def get_absolute_url(self):
>>>         kwargs = {'slug': self.slug}
>>>         return reverse('foo.detail', kwargs=kwargs)
```

Do not forget to add the `LocaleURLMiddleware` to the `MIDDLEWARE_CLASSES` (as first).

```
>>> MIDDLEWARE_CLASSES = (
>>>     'localeurl.middleware.LocaleURLMiddleware',
>>>     # The rest...
>>> )
```

Also, add *localeurl* to `INSTALLED_APPS`.

```
>>> INSTALLED_APPS = (
>>>     # Some apps...
>>>     'localeurl',
>>>     # Some more apps...
>>> )
```

CHAPTER 4

License

GPL 2.0/LGPL 2.1

CHAPTER 5

Support

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

Author

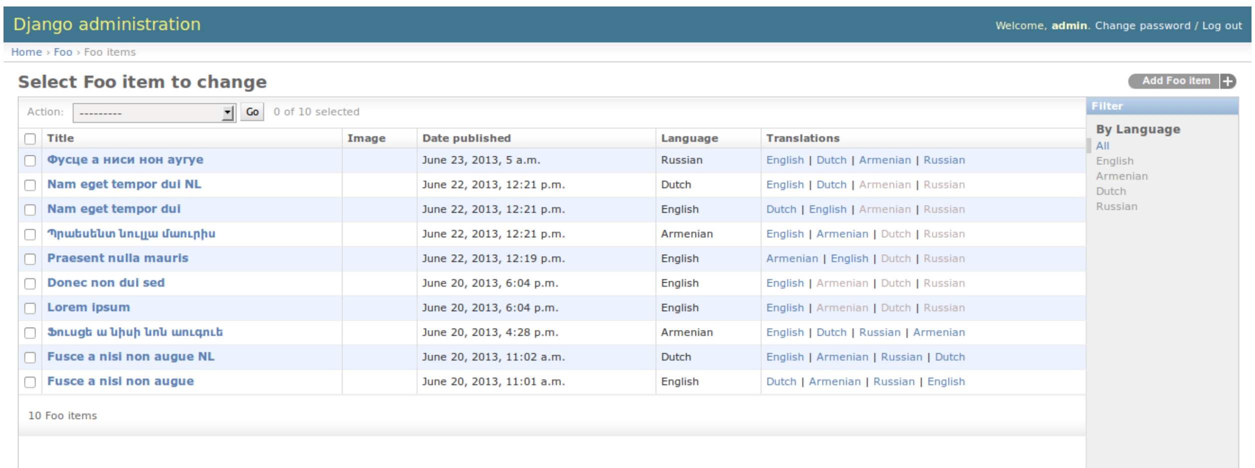
Artur Barseghyan <artur.barseghyan@gmail.com>

CHAPTER 7

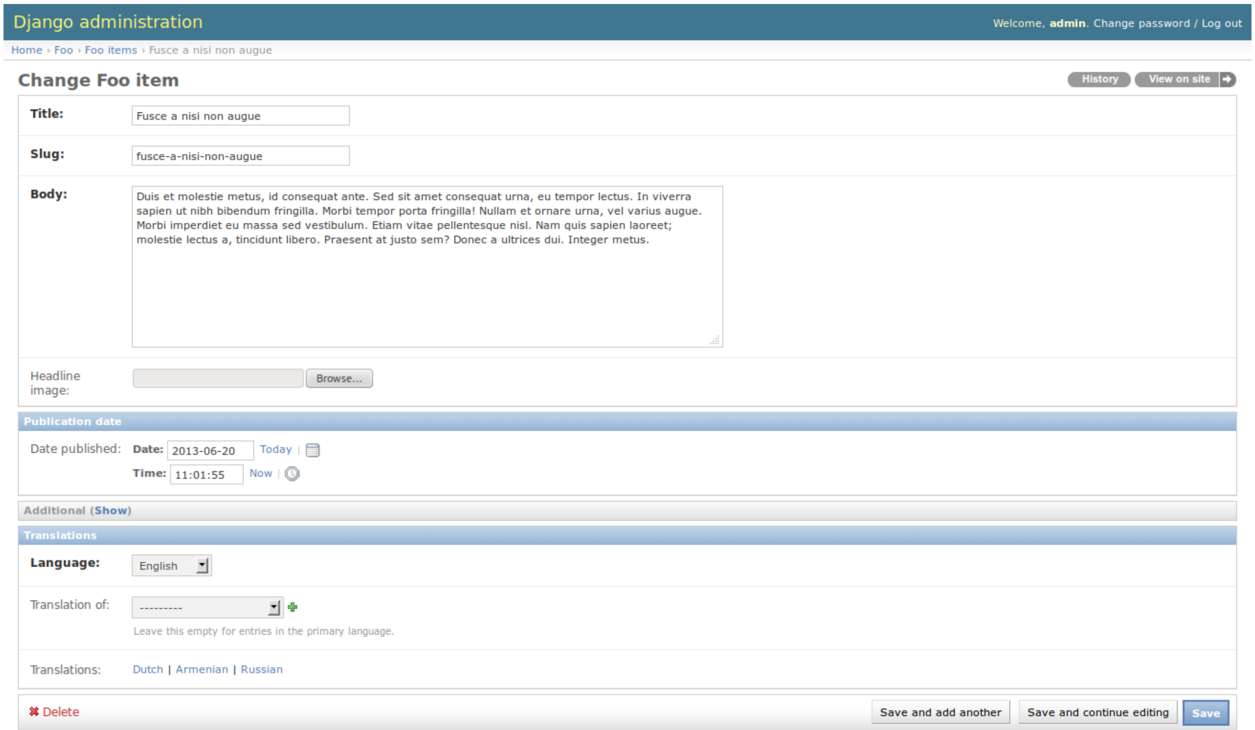
Documentation

7.1 Screenshots

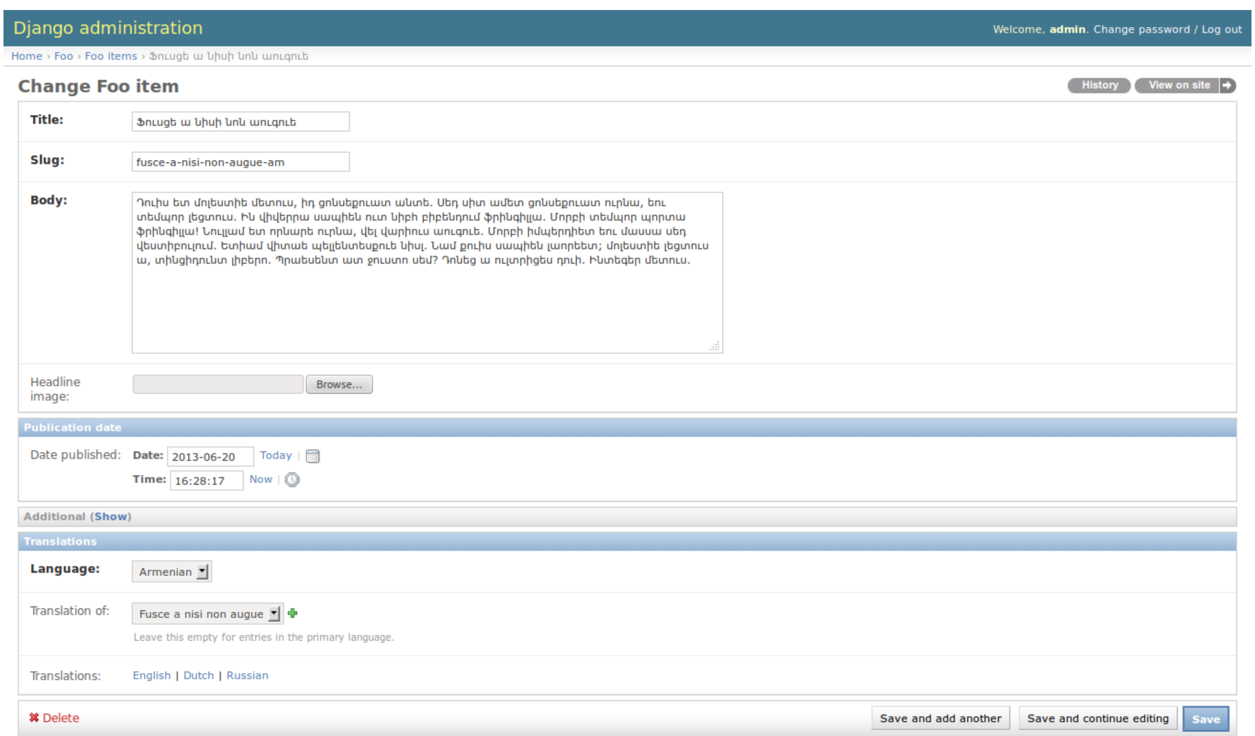
List view:



Edit view for main language:



Edit view for translated item:



7.2 `models` Package

class `slim.models.__init__.`**`Slim`**

Bases: `object`

Add this class to all your multi-lingual Django models, where you use `slim.models.fields.LanguageField`. Alternatively, you may use the `slim.models.SlimBaseModel`.

`available_translations`()

Returns available translations.

Return interable At this moment a list of objects.

`available_translations_admin`(**args*, ***kwargs*)

Gets a HTML with all available translation URLs for current object if available. For admin use.

Return str

`available_translations_exclude_current_admin`(**args*, ***kwargs*)

Same as *available_translations_admin* but does not include itself to the list.

Return str

`get_original_translation`(**args*, ***kwargs*)

Gets original translation of current object.

Return obj Object of the same class as the one queried.

`get_redirect_to_target`(*request*)

Find an acceptable redirect target. If this is a local link, then try to find the page this redirect references and translate it according to the user's language. This way, one can easily implement a localized "/"-url to welcome page redirection.

`get_translation_for`(*language*)

Get translation article in given language.

Parameters language (*str*) – Which shall be one of the languages specified in `LANGUAGES` in *settings.py*.

Return obj Either object of the same class as or None if no translations are available for the given `language`.

`is_multilingual`

Simple flat to use on objects to find our wheither they are multilinugal or not

Return bool Always returns boolean True

`original_translation`

Property for `get_original_translation` method.

Return obj Object of the same class as the one queried.

`translation_admin`(**args*, ***kwargs*)

Gets a HTML with URL to the original translation of available. For admin use.

Return str

class `slim.models.__init__.`**`SlimBaseModel`**(**args*, ***kwargs*)

Bases: `django.db.models.base.Model`, `slim.models.__init__.Slim`

An abstract Django model.

class Meta

abstract = False

7.3 fields Module

class slim.models.fields.**LanguageField**(**args*, ***kwargs*)

Bases: django.db.models.fields.CharField

LanguageField model. Stores language string in a CharField field.

Using *contrib_to_class* melthod adds *translation_of* field, which is simply a ForeignKey to the same class.

contribute_to_class(*cls*, *name*)

formfield(***kwargs*)

Returns best form field to represent this model field

validate(*value*, *model_instance*)

Validating the field.

We shall make sure that there are double translations for the same language for the same object. That's why, in case if model is not yet saved (translated_object does not yet have a primary key), we check if there are already translations of the same object in the language we specify now.

Otherwise, if model_instance already has a primary key, we anyway try to get a translated_object and compare it with our model_instance. In case if translated_object exists and not equal to our model_instance we raise an error.

NOTE: This has nothing to do with unique fields in the original model_instance. Make sure you have properly specified all unique attributes with respect to LanguageField\` of your original \`\`model_instance if you need those records to be unique.

class slim.models.fields.**SimpleLanguageField**(**args*, ***kwargs*)

Bases: django.db.models.fields.CharField

SimpleLanguageField model. Stores language string in a CharField field.

formfield(***kwargs*)

Returns best form field to represent this model field

7.4 decorators Module

slim.models.decorators.**prepend_language**(*func*, *language_field='language'*)

Prepends the language from the model to the path resolved.

slim.models.decorators.**localeurl_prepend_language**(*func*, *language_field='language'*)

Prepends the language from the model to the path resolved when *django-localeurl* package is used.

slim.models.decorators.**auto_prepend_language**(*func*, *language_field='language'*)

Prepends the language from the model to the path resolved.

7.5 `admin` Module

class `slim.admin.`**`SlimAdmin`**(*model*, *admin_site*)

Bases: `django.contrib.admin.options.ModelAdmin`

SlimAdmin.

`list_view_primary_only` - if set to True, onlt primary language items would be shown in the list view. Default value is False.

`language_field` - name of the language field defined in your model. Default value *language*.

`auto_add_edit_view` - if set to True, extra fields for language editing are added to the list view. Do NOT set this value to False!

`collapse_slim_fieldset` if set to True, the language fieldset is shown collapsed.

`auto_add_edit_view` = **True**

`auto_add_list_view` = **True**

`collapse_slim_fieldset` = **True**

`declared_fieldsets`

`get_list_display`(**args*, ***kwargs*)

`get_list_filter`(**args*, ***kwargs*)

`get_readonly_fields`(**args*, ***kwargs*)

`language_field` = **'language'**

`list_view_primary_only` = **False**

`media`

`queryset`(**args*, ***kwargs*)

7.6 `conf` Module

`slim.conf.`**`get_setting`**(*setting*, *override=None*)

Get a setting from `slim` conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

7.7 `helpers` Module

`slim.helpers.`**`get_default_language`**()

Gets default language.

Return str

`slim.helpers.`**`get_languages`**()

Gets available languages.

Return iterable

`slim.helpers.`**`get_languages_keys`**()

Returns just languages keys.

Return list

`slim.helpers.get_language_from_request(request, default='af')`

Gets language from HttpRequest

Parameters

- **django.http.HttpRequest** –
- **default** (*str*) –

Return str

`slim.helpers.get_languages_dict()`

Returns just languages dict.

Return dict

`slim.helpers.admin_change_url(app_label, module_name, object_id, extra_path='', url_title=None)`

Gets an admin change URL for the object given.

Parameters

- **app_label** (*str*) –
- **module_name** (*str*) –
- **object_id** (*int*) –
- **extra_path** (*str*) –
- **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

`slim.helpers.admin_add_url(app_label, module_name, extra_path='', url_title=None)`

Gets an admin edit URL for the object given.

Parameters

- **app_label** (*str*) –
- **module_name** (*str*) –
- **extra_path** (*str*) –
- **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

`slim.helpers.smart_resolve(var, context)`

Resolves variable from context in a smart way. First trying to resolve from context and when result is None checks if variable is not None and returns just variable when not. Otherwise returns None.

Parameters

- **var** (*str*) –
- **context** (*Context*) –

Return mixed

7.8 `utils` Module

`slim.utils.`**`locale_url_is_installed`**`()`

Checks if localeurl is installed in the Django project.

Return bool

7.9 `translations` Module

`slim.translations.`**`short_language_code`**`(`*code=None*`)`

Extracts the short language code from its argument (or return the default language code).

Parameters code (*str*) –

Return str

from django.conf import settings >>> short_language_code('de') 'de' >>> short_language_code('de-at') 'de'

`slim.translations.`**`is_primary_language`**`(`*language=None*`)`

Returns true if current or passed language is the primary language for this site. (The primary language is defined as the first language in settings.LANGUAGES.)

Parameters language (*str*) –

Return bool

7.10 `slim_tags` Module

`slim.templatetags.slim_tags.`**`get_translated_object_for`**`(`*parser*, *token*`)`

Gets translated object for the object given.

Syntax:: {% get_translated_object_for [object] language=[language] as [var_name] %}

Example usage:: {% get_translated_object_for article as translated_article %} {% get_translated_object_for article language=ru as translated_article %}

`slim.templatetags.slim_tags.`**`get_translated_objects_for`**`(`*parser*, *token*`)`

Gets translations available for the given object.

Syntax:: {% get_translated_objects_for [object] as [var_name] %}

Example usage:: {% get_translated_objects_for article as translated_article %}

`slim.templatetags.slim_tags.`**`set_language`**`(`*parser*, *token*`)`

Sets current language code.

FIXME: This is actually a hack.

Syntax:: {% set_language [language] %}

Example:: {% set_language ru %}

`slim.templatetags.slim_tags.`**`multiling_is_enabled`**`(`*parser*, *token*`)`

Checks if multiling shall be enabled (in templates). Simply, if LANGUAGES tuple contains more than one language, we return boolean True; otherwise - boolean False.

Syntax:: {% multiling_is_enabled as [var_name] %}

Example:: {% multiling_is_enabled as multiling_is_enabled %}

`slim.templatetags.slim_tags.`**`slim_language_name`**(*lang_code*)

Not all languages are available in Django yet. It might happen that support for your own precious language is not yet available in Django and many apps rely on languages list defined in *django.conf.global_settings* module.

Thus, to have translations for your own language available, the following approach is introduced: - Pick the language code closest to your language but name it differently: ((‘ar’, ugettext(‘Armenian’)),). - Instead of using Django’s *language_name* filter (of *i18n* module) use *slim_language_name* filter just the

same way.

This filter would get your item tranlation based on your project translations.

Parameters **lang_code** (*str*) –

Return str

CHAPTER 8

Indices and tables

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

S

A

C

D

F

G

I

L

M

O

P

Q

S

T

V