PO-Projects Documentation

Release 0.7.4

David THENON

February 19, 2015

1 Goal **3**

2 Features **5**

3 Links **7**

4 Table of contents **9**

4.1 Install 9

4.2 Usage 10

4.3 Changelog 11

PO Projects is a PO file management factory.

CHAPTER 1

Goal

Have a clean and ergonomic frontend to manage PO files for webapp projects and enable a REST API to deploy PO files into webapp projects.

CHAPTER 2

Features

- View to create new project from a PO/POT file;
- View to create new project translation and edit them;
- View to export project translations as PO files;
- View to export an archive (zip/tarball) of all translations as PO files (and their compiled MO files) from a project;
- Manage fuzzy mode for translations;
- Form to import a catalog (PO) to update a catalog;
- Nice frontend with Foundation;
- Permission restriction;
- Restricted API access with djangorestframework to get PO files or global project archive from external tools (like Optimus or a Django app from an external site) ?

CHAPTER 3

Links

- Read the documentation on Read the docs;
- Download his PyPi package;
- Clone it on his Github repository;

CHAPTER 4

Table of contents

4.1 Install

Add *PO Projects* to your installed apps in settings :

```
INSTALLED_APPS = (
    ...
    'po_projects'
    ...
)
```

Then import its default settings :

```
from po_projects.settings import *
```

If needed you can override some of its settings, see the original file to watch about available settings.

Finally mount its urls in your main `urls.py` :

```
urlpatterns = patterns('',
    ...
    (r'^po/', include('po_projects.urls', namespace='po_projects')),
    ...
)
```

4.1.1 External API access

To enable the rest API you will have to install djangorestframework in your settings :

```
INSTALLED_APPS = (
    ...
    'rest_framework'
    ...
)

REST_FRAMEWORK = {
    'PAGINATE_BY': 10,
    # Use hyperlinked styles by default.
    # Only used if the `serializer_class` attribute is not set on a view.
    'DEFAULT_MODEL_SERIALIZER_CLASS': (
        'rest_framework.serializers.HyperlinkedModelSerializer',
    ),
    # Use Django's standard `django.contrib.auth` permissions,
    # or allow read-only access for unauthenticated users.
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAdminUser',
        #'rest_framework.permissions.DjangoModelPermissionsOrAnonReadOnly',
    ),
}
```

And mount it in your `urls.py` :

```
urlpatterns = patterns('',
    ...
    (r'^api-auth/', include('rest_framework.urls', namespace='rest_framework')),
    ...
)
```

And so a rest API will be available on :

```
/po/rest/
```

It is browsable for authenticated users with admin rights (`is_staff` on True), also the client will need to access to the API with an user accounts with the admin rights.

PO-Projects-client is the client to use the API from your project.

4.2 Usage

4.2.1 Introducing to PO

PO file (`messages.po`) is a **gettext** format in plain text to manage texts translation management.

A PO file contains translations referenced with an identifiant (`msgid`) that is the text source marked for translation.

For each identifiant there is one message translation (`msgstr`) that should contain the translation of the text source. If the translation is empty, it is ignored and the text source will be used instead.

Also a translation can be marked as **fuzzy**, that means gettext has finded a translation has changed (its id or its message) but it is not sure about it. Until the *fuzzy* mark is keeped, the translation will be ignored, you have to remove it the translation message is right.

When a PO file is finished, a developer deploy it with compiling it to a MO file (`messages.mo`) that is ready to use by an application.

4.2.2 Workflow

Previously, translators and developers have to share PO files, deploy them and compile them manually.

With *PO-Projects* this is more simple if PO-Projects-client is installed and configured for the translation project.

Pushing new extracted translation

1. When templates or code has changed, developers extract again translation into the PO files;
2. Developers go into the webapp project and use the client's `push`;
3. It's done, new and updated translations sources are sended to the factory, the translators can work on them;

Pulling translation from the factory

1. Translators do translations onto a project;
2. When they have finished they ask a developer to deploy PO files;
3. Developers go into the webapp project and use the client's `pull` command (then restart the webserver if any);
4. It's done

4.3 Changelog

4.3.1 Version 0.7.4 - 20 February 2015

- Add forgotten locale directory;
- Fix bug on compiled PO in project's tarball archive so the client deploys correct PO and MO files;
- Starting new documentation;

4.3.2 Version 0.7.3 - 19 February 2015

- Better layout and ergonomy on all ressources;

4.3.3 Version 0.7.2 - 16 February 2015

- Better Catalog create form on project details page;

4.3.4 Version 0.7.1 - 16 February 2015

- Better translation source rendering in the translation update form;
- Remove the Project's slug field from the project update form;

4.3.5 Version 0.7.0 - 15 February 2015

- Add a more ergonomic interface for translation form with sticky fixed menu, translation statistics and a more readable layout;
- Add 'domain' attribut on Project model so now projects have an explicit domain name to use for write PO catalog files;
- Force gettext domain in forget Babel catalogs;
- Move forms and add crispies;
- Add assets for django-asets;
- Add Compass stuff to build SCSS;
- Add static files;