wagtail-modeltranslation Documentation

Release stable

May 12, 2023

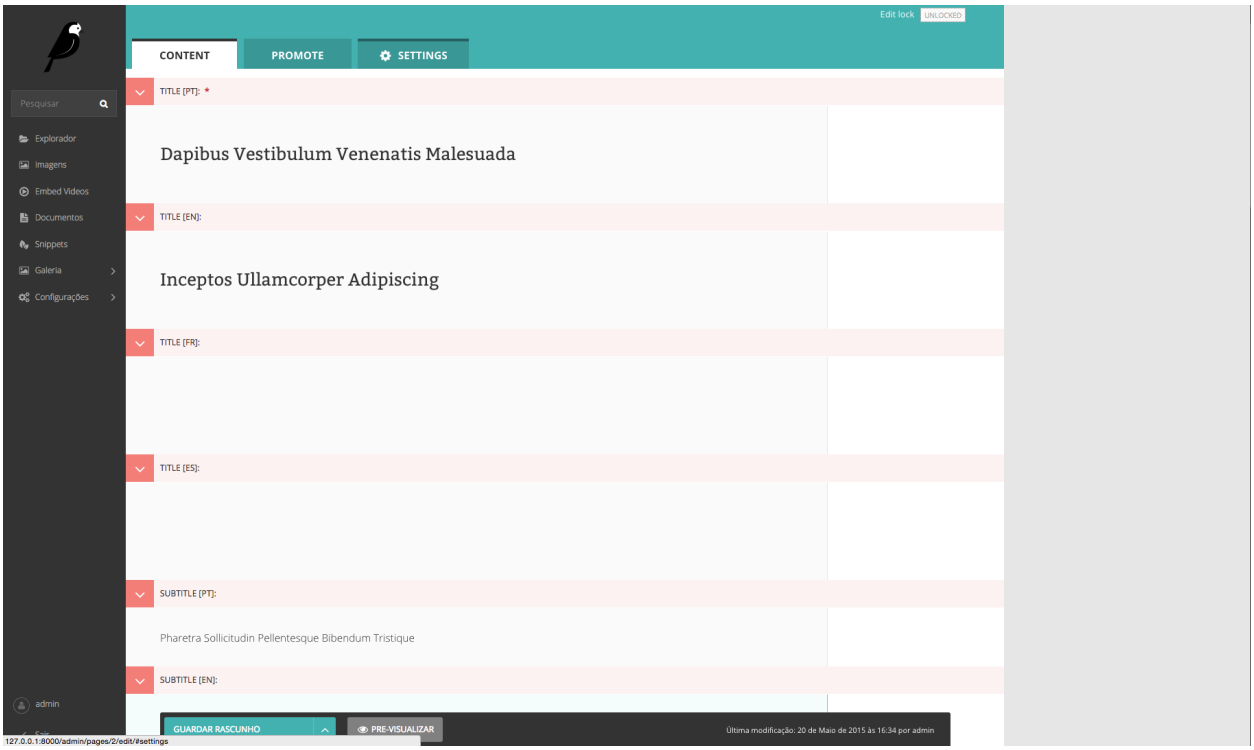
Contents

This app is built using core features of django-modeltranslation: https://github.com/deschler/django-modeltranslation

It's an alternative approach for i18n support on Wagtail CMS websites.

The modeltranslation application is used to translate dynamic content of existing Wagtail models to an arbitrary number of languages, without having to change the original model classes. It uses a registration approach (comparable to Django's admin app) to add translations to existing or new projects and is fully integrated into the Wagtail admin UI.

The advantage of a registration approach is the ability to add translations to models on a per-app basis. You can use the same app in different projects, whether or not they use translations, and without touching the original model class.



CHAPTER 1

Features

- Add translations without changing existing models or views
- Translation fields are stored in the same table (no expensive joins)
- Supports inherited models (abstract and multi-table inheritance)
- Handle more than just text fields
- Wagtail admin integration (for Page, BaseSiteSetting and Snippet models)
- Flexible fallbacks, auto-population and more!
- Default Page model fields has translatable fields by default

CHAPTER 2

Contents

2.1 Introduction

2.1.1 Creating multilingual sites

I18n

Django and Wagtail CMS have implemented Internationalisation (I18n) in their frameworks. Hooks are provided for translating strings such as literals. Furthermore, **locale language files** are included. This is where the translated text of the frameworks is stored.

When writing your own apps, it is recommended that you use Il18n. If you need guidance, you can read the Django Internalization Documentation.

Wagtail-modeltranslation

Another important component in the translation equation is the content stored in database fields. This is where wagtail-modeltranslation comes into play.

Wagtail-modeltranslation uses django-modeltranslation to register which fields need to be translated, and provides the integration with the wagtail admin interface so that translation fields are displayed and edited together on the same page. Translated fields can be used in your templates and as you would use any other field.

Some of the advantages of wagtail-modeltranslation

- The same template is used for multiple languages
- The document tree is simpler with no need to have a separate branch for each language
- Languages can be added without changing existing models or views
- Translation fields are stored in the same table (no expensive joins)
- Can handle more than just text fields
- Wagtail admin integration

- Flexible fallbacks, auto-population and more!
- Default Page model has translatable fields by default
- StreamFields are supported
- Easy to implement

Examples used in this document

We will be using a fictitious model `foo` in the coding examples.

Wagtail-modeltranslation and Django-Modeltranslation

This document only covers the integration of the translated fields in the wagtail admin and simple model field registration. For more advanced usage of field registering functionalities please check django-modeltranslation documentation..

2.2 Installation

2.2.1 Requirements

- Wagtail >= 1.12

Installing using Pip

```
$ pip install wagtail-modeltranslation
```

Installing using the source

- From github: **git clone** https://github.com/infoportugal/wagtail-modeltranslation.git
  - Copy wagtail_modeltranslation folder in project tree

  OR
- Download ZIP file on Github.com from infoportugal/wagtail-modeltranslation
  - Unzip and copy wagtail_modeltranslation folder in project tree

2.2.2 Quick Setup

To setup the application please follow these steps:

1. In your settings file:
   - Add 'wagtail_modeltranslation' to `INSTALLED_APPS`

```
INSTALLED_APPS = (
    ...
    'wagtail_modeltranslation',
    'wagtail_modeltranslation.makemigrations',
    'wagtail_modeltranslation.migrate',
)
```

- Add 'django.middleware.locale.LocaleMiddleware' to `MIDDLEWARE` (`MIDDLEWARE_CLASSES` before django 1.10).

```
MIDDLEWARE = (
    ...
    'django.middleware.locale.LocaleMiddleware',  # should be after␣
↪SessionMiddleware and before CommonMiddleware
)
```

- Set `USE_I18N = True`
- Configure your `LANGUAGES` setting.

  The `LANGUAGES` variable must contain all languages you will use for translation. The first language is treated as the *default language*.

  Modeltranslation uses the list of languages to add localized fields to the models registered for translation. For example, to use the languages Portuguese, Spanish and French in your project, set the `LANGUAGES` variable like this (where `pt` is the default language). In required fields the one for the default language is marked as required (for more advanced usage check django-modeltranslation required_languages.)

  Warning: When the `LANGUAGES` setting isn't present in `settings.py` (and neither is `MODELTRANSLATION_LANGUAGES`), it defaults to Django's global LANGUAGES setting instead, and there are quite a few languages in the default!

Note: To learn more about preparing Wagtail for Internationalisation check the Wagtail i18n docs.

2. Create a `translation.py` file in your app directory and register `TranslationOptions` for every model you want to translate and for all subclasses of Page model.

```
from .models import foo
from modeltranslation.translator import TranslationOptions
from modeltranslation.decorators import register

@register(foo)
class FooTR(TranslationOptions):
    fields = (
       'body',
    )
```

3. Run `python manage.py makemigrations` followed by `python manage.py migrate`. This will add the tranlation fields to the database, repeat every time you add a new language or register a new model.
4. Run `python manage.py sync_page_translation_fields`. This will add translation fields to Wagtail's `Page` table, repeat every time you add a new language.

5. If you're adding `wagtail-modeltranslation` to an existing site run `python manage.py update_translation_fields`.
6. Define the panels for the original fields, as you normally would, as wagtail-modeltranslation will generate the panels for the translated fields.

2.3 Registering models for translation

`Modeltranslation` can translate model fields of any model class.

Registering models and their fields used for translation requires the following steps:

1. Create **translation.py** in your app directory.
2. Define the models you want to use, import django-modeltranslation's **TranslationOptions** and the django-modeltranslation **register** decorator
3. Create a translation option class for every model you want to translate and precede the class with the **@register** decorator.

The django-modeltranslation application reads the **translation.py** file in your app directory thereby triggering the registration of the translation options found in the file.

A translation option is a class that declares which model fields are needed for translation. The class must derive from **modeltranslation.translator.TranslationOptions** and it must provide a **field** attribute storing the list of field names. The option class must be registered with the **modeltranslation.decorators.register** instance.

To illustrate this let's have a look at a simple example using a **Foo** model. The example only contains an **introduction** and a **body** field.

Instead of a **Foo** model, this could be any Wagtail model class:

```
from .models import Foo
from modeltranslation.translator import TranslationOptions
from modeltranslation.decorators import register

@register(Foo)
class FooTR(TranslationOptions):
    fields = (
        'introduction',
        'body',
    )
```

In the above example, the **introduction** and **body** language fields will be be added for each language defined in LANGUAGES in the settings file , **base.py**, when the database is updated with **./manage.py makemigrations** and **./manage.py migrate.**

At this point you are mostly done and the model classes registered for translation will have been added some auto-magical fields. For more under-the-hood details check django-modeltranslation docs.

Now you can define the panels for the model as you normally would and wagtail-modeltranslation will take care of the creation of the panels for all the supported languages.

```
# Indicate fields to include in Wagtail admin panel(s)
Foo.content_panels = [
    FieldPanel('title', classname="full title"),
    FieldPanel('introduction', classname="full"),
    FieldPanel('body', classname="full"),
    ]
```

(continues on next page)

(continued from previous page)

```
#or with a custom edit_handler
edit_handler = TabbedInterface([
    ObjectList(content_panels, heading='Content'),
    ObjectList(sidebar_content_panels, heading='Sidebar content'),
    ObjectList(Page.promote_panels, heading='Promote'),
    ObjectList(Page.settings_panels, heading='Settings', classname="settings"),
])
```

The panel creation is available for **Page**, **BaseSiteSetting** and **Snippet** models.

2.3.1 Precautions regarding registration approach

Be aware that registration approach (as opposed to base-class approach) to models translation has a few caveats, though (despite many pros).

First important thing to note is the fact that translatable models are being patched - that means their fields list is not final until the modeltranslation code executes. In normal circumstances it shouldn't affect anything - as long as `models.py` contain only models' related code.

For example: consider a project where a `ModelForm` is declared in `models.py` just after its model. When the file is executed, the form gets prepared - but it will be frozen with old fields list (without translation fields). That's because the `ModelForm` will be created before modeltranslation would add new fields to the model (`ModelForm` gathers fields info at class creation time, not instantiation time). Proper solution is to define the form in `forms.py`, which wouldn't be imported alongside with **models.py** (and rather imported from views file or urlconf).

Generally, for seamless integration with modeltranslation (and as sensible design anyway), the models.py `should contain only bare models and model related logic`.

2.3.2 Committing fields to database

Modeltranslation supports the migration system introduced by Django 1.7. Besides the normal workflow as described in Django's Migration Docs, you should do a migration whenever one of the following changes have been made to your project:

- Added or removed a language through `settings.LANGUAGES` or `settings.MODELTRANSLATION LANGUAGES`.
- Registered or unregistered a field through `TranslationOptions`.

It doesn't matter if you are starting a fresh project or change an existing one, it's always:

1. `python manage.py makemigration` to create a new migration with the added or removed fields.
2. `python manage.py migrate` to apply the changes.
3. If you've added a new language `python manage.py sync_page_translation_fields` to add *Page* translation fields.

2.3.3 Required fields

By default, only the default language of a required field is marked as required (eg. if you have field bar and the default language is pt the only required field will be bar_pt). This behavior can be customized using required_languages.

2.3.4 Supported fields

The list of all suported fields is available here.

2.3.5 Supported panels

The creation of panels for the translation fields supports the following panel classes:

- **FieldPanel**
- **MultiFieldPanel**
- **InlinePanel**

2.4 Advanced Settings

Besides the django-modeltranslation settings, documented here this app provides the following custom settings:

2.4.1 WAGTAILMODELTRANSLATION_CUSTOM_SIMPLE_PANELS

Default: `[]` (empty list)

This setting is used to add custom "simple panel" classes (all panels that contain directly a field value, like FieldPanel) that need patching but are not included by default, resulting in not being created translated versions of that panel in wagtail admin. If, for example, you're using wagtail-embedvideos the EmbedVideoChooserPanel is not patched by default so you'd need to include the fully qualified class name like the example below. This setting must be a list of fully qualified class names as strings.

```
WAGTAILMODELTRANSLATION_CUSTOM_SIMPLE_PANELS = ['wagtail_embed_videos.edit_handlers.
↪EmbedVideoChooserPanel']
```

2.4.2 WAGTAILMODELTRANSLATION_CUSTOM_COMPOSED_PANELS

Default: `[]` (empty list)

This setting behaves as the above but should be used for panels that are composed by other panels (MultiFieldPanel or FieldRowPanel for example).

```
WAGTAILMODELTRANSLATION_CUSTOM_COMPOSED_PANELS = ['app_x.module_y.PanelZ']
```

2.4.3 WAGTAILMODELTRANSLATION_CUSTOM_INLINE_PANELS

Default: `[]` (empty list)

This setting behaves as the above but should be used for panels that inherit InlinePanel.

```
WAGTAILMODELTRANSLATION_CUSTOM_INLINE_PANELS = ['app_x.module_y.PanelZ']
```

2.4.4 WAGTAILMODELTRANSLATION_TRANSLATE_SLUGS

Default: `True`

This setting makes slug and url_path localized. If True, each page will have a slug and url_path per language. If a slug field is not translated it will be automatically populated when the page title of it's language is filled.

```
WAGTAILMODELTRANSLATION_TRANSLATE_SLUGS = True
```

2.4.5 WAGTAILMODELTRANSLATION_LOCALE_PICKER

Default: `True`

This setting injects a locale picker in the editor interface, so that only selected locale fields are shown.

```
WAGTAILMODELTRANSLATION_LOCALE_PICKER = True
```

2.4.6 WAGTAILMODELTRANSLATION_LOCALE_PICKER_DEFAULT

Default: `None`

This setting specifies, which languages should initially be enabled on the edit pages when the locale picker is used. If not set, just the default language from `MODELTRANSLATION_DEFAULT_LANGUAGE` is initially enabled.

```
WAGTAILMODELTRANSLATION_LOCALE_PICKER_DEFAULT = None            # only default␣
↪language initially enabled
WAGTAILMODELTRANSLATION_LOCALE_PICKER_DEFAULT = [ ]             # all languages␣
↪initially disabled
WAGTAILMODELTRANSLATION_LOCALE_PICKER_DEFAULT = [ 'en', 'de' ]  # these languages␣
↪initially enabled
```

2.4.7 WAGTAILMODELTRANSLATION_LOCALE_PICKER_STORE

Default: `False`

If set to true, the language picker will restore language selection on each page. Otherwise, the default will be used

```
WAGTAILMODELTRANSLATION_LOCALE_PICKER_RESTORE = False # the default will be used on␣
↪each page
WAGTAILMODELTRANSLATION_LOCALE_PICKER_RESTORE = True  # the last used language will␣
↪be used on each page
```

2.5 Template Tags

2.5.1 change_lang

Use this template tag to get the url of the given page in another language. The parameteres of this template tag are the language code and page object. Below is an example where we want to get the url of the current page in portuguese.

```
{% load wagtail_modeltranslation %}
{% change_lang 'pt' page %}
```

2.5.2 slugurl_trans

Use this template tag as a replacement for `slugurl`.

```
{% load wagtail_modeltranslation %}
{% slugurl_trans 'default_lang_slug' %}
{# or #}
```

2.5.3 get_available_languages_wmt

Use this template tag to get the current languages from MODELTRANSLATION_LANGUAGES (or LANGUAGES) from your setting file (or the default settings).

```
{% get_available_languages_wmt as languages %}
{% for language in languages %}
...
{% endfor %}
{% slugurl_trans 'pt_lang_slug' 'pt' %}
```

2.6 Management Commands

2.6.1 wagtail_modeltranslation

wagtail_modeltranslation module adds the following management commands.

The `update_translation_fields` Command

This command is a proxy to `django-modeltranslation`'s own `update_translation_fields`, for more details read the corresponding documentation on django-modeltranslation docs.

In case modeltranslation was installed in an existing project and you have specified to translate fields of models which are already synced to the database, you have to update your database schema.

Unfortunately the newly added translation fields on the model will be empty then, and your templates will show the translated value of the fields which will be empty in this case. To correctly initialize the default translation field you can use the `update_translation_fields` command:

```
$ python manage.py update_translation_fields
```

The `sync_page_translation_fields` Command

New in version 0.8.

This command compares the database and translated Page model definition (finding new translation fields) and provides SQL statements to alter `wagtailcore_page` table. You should run this command after installation and after adding a new language to your `settings.LANGUAGES`.

```
$ python manage.py sync_page_translation_fields
```

The `makemigrations_translation` Command

New in version 0.8.

`wagtail-modeltranslation` patches Wagtail's `Page` model and as consequence Django's original `makemigrations` commmand will create migrations for `Page` which may create conflicts with other migrations. To circumvent this issue `makemigrations_translation` hides any `Page` model changes and creates all other migrations as usual. Use this command as an alternative to Django's own `makemigrations` or consider using *The makemigrations Command*.

```
$ python manage.py makemigrations_translation
```

The `migrate_translation` Command

New in version 0.8.

Since *The makemigrations_translation Command* hides any `Page` model changes, Django's own `migrate` command won't be able to update `wagtailcore_page` table with new translation fields. In order to correctly update the database schema a combination of `migrate` followed by sync_page_translation_fields is usually required. `migrate_translation` provides a shortcut to running these two commands. Use this as an alternative to Django's own `migrate` or consider using *The migrate Command*.

```
$ python manage.py migrate_translation
```

The `set_translation_url_paths` Command

Updates url_path translation fields for all pages.

```
$ python manage.py set_translation_url_paths
```

2.6.2 wagtail_modeltranslation.makemigrations

To use `wagtail_modeltranslation.makemigrations` module commands add `'wagtail_modeltranslation.makemigrations,'` to `INSTALLED_APPS`. This module adds the following management commands.

The `makemigrations` Command

This command is a proxy for *The makemigrations_translation Command*. It has the added benefit of overriding Django's own `makemigrations` allowing you to run `makemigrations` safely without creating spurious `Page` migrations.

```
$ python manage.py makemigrations
```

The `makemigrations_original` Command

Since Django's `makemigrations` is overriden by `wagtail-modeltranslation`'s version use `makemigrations_original` to run the Django's original `makemigrations` command. Please note this will likely create invalid `Page` migrations, do this only if you know what you're doing.

```
$ python manage.py makemigrations_original
```

2.6.3 wagtail_modeltranslation.migrate

To use `wagtail_modeltranslation.migrate` module commands add `'wagtail_modeltranslation.migrate,'` to `INSTALLED_APPS`. This module adds the following management commands.

The `migrate` Command

This command is a proxy for *The migrate_translation Command*. It has the added benefit of overriding Django's own `migrate` saving the need to additionally run sync_page_translation_fields. See issue #175 to understand how this command can be used to create translation fields in a test database.

```
$ python manage.py migrate
```

The `migrate_original` Command

Since Django's `migrate` is overriden by `wagtail-modeltranslation`'s version use `migrate_original` to run the Django's original `migrate` command. Please note this will not update `wagtailcore_page` table with new translation fields, use sync_page_translation_fields for that.

```
$ python manage.py migrate_original
```

2.7 Caveats

2.7.1 Wagtail's `Page` patch

`wagtail-modeltranslation` patches Wagtail's `Page` model with translation fields `title_xx`, `slug_xx`, `seo_title_xx`, `search_description_xx` and `url_path_xx` where "xx" represents the language code for each translated language. This is done without migrations through *The sync_page_translation_fields Command*. Since `Page` model belongs to Wagtail it's within the realm of possibility that one day Wagtail may add a conflicting field to `Page` thus interfering with `wagtail-modeltranslation`.

See also *The makemigrations_translation Command* to better understand how migrations are managed with `wagtail-modeltranslation`.

2.7.2 Wagtail's slugurl

Wagtail's `slugurl` tag does not work across languages. To work around this `wagtail-modeltranslation` provides a drop-in replacement tag named *slugurl_trans* which by default takes the slug parameter in the default language.

Replace any usages of Wagtail's `{% slugurl 'default_lang_slug' %}` for

```
{% load wagtail_modeltranslation %}
...
{% slugurl_trans 'default_lang_slug' %}
```

2.8 Upgrade considerations (v0.10.8)

- Template tag `change_lang` now needs a second parameter, `page`

2.9 Upgrade considerations (v0.8)

This version includes breaking changes as some key parts of the app have been re-written:

- The most important change is that `Page` is now patched with translation fields.
- `WAGTAILMODELTRANSLATION_ORIGINAL_SLUG_LANGUAGE` setting has been deprecated.

To upgrade to this version you need to:

- Replace the `WagtailTranslationOptions` with `TranslationOptions` in all translation.py files
- Run `python manage.py sync_page_translation_fields` at least once to create `Page`'s translation fields
- Replace any usages of Wagtail's `{% slugurl ... %}` for `wagtail-modeltranslation`'s own `{% slugurl_trans ... %}`
- While optional it's recommended to add `'wagtail_modeltranslation.makemigrations'` to your INSTALLED_APPS. This will override Django's `makemigrations` command to avoid creating spurious `Page` migrations.

2.10 Upgrade considerations (v0.6)

This version has some important changes as there was a refactoring to include django-modeltranslation as a dependency instead of duplicating their code in our version. This allow us to focus on Wagtail admin integration features as django-modeltranslation is very well mantained and is very quickly to fix problems with the latest Django versions. This way we also keep all the django-modeltranslation features (if you want you can also customize django-admin, for example). We also provide a new class to create the translation options classes: **WagtailTranslationOptions** Most of the changes are related to imports as they change from wagtail-modeltranslation to modeltranslation.

To upgrade to this version you need to:

- Replace the `TranslationOptions` with `WagtailTranslationOptions` in all translation.py files
- The import of the register decorator is now `from modeltranslation.decorators import register`
- The import of translator is now `from modeltranslation.translator import translator`

2.11 Recommended reading

Although the contents of this documentation cover the basic usage of wagtail-modeltranslation and be enough for most cases, there are some changes in default behaviour by django-modeltranslation and also some advanced configurations for further customization. We recommend the reading of the following links.

- Field registration and model patching.
- Field access rules and fallbacks.
- Settings customization.

2.12 Release notes

2.12.1 wagtail-modeltranslation 0.6 rc2 release notes

Changelist:

- django-modeltranslation is now a dependency.
- added compatibility with Python 3 (3.3, 3.4, 3.5).
- dropped support for wagtail versions prior to 1.4.

Bug fixes:

- sometimes the required fields weren't marked as so, raising an Exception not caught on the form.
- patch of panels when a custom edit_handler is defined.
- set_url_method which caused to child of a page not being updated when the parent path changed.
- validation of duplicated slugs.

Upgrade considerations

This version has some important changes as there was a refactoring to include django-modeltranslation as a dependency instead of duplicating their code in our version. This allow us to focus on Wagtail admin integration features as django-modeltranslation is very well mantained and is very quickly to fix problems with the latest Django versions. This way we also keep all the django-modeltranslation features (if you want you can also customize django-admin, for example). We also provide a new class to create the translation options classes: **WagtailTranslationOptions** Most of the changes are related to imports as they change from wagtail-modeltranslation to modeltranslation.

To upgrade to this version you need to:

- Replace the `TranslationOption` with `WagtailTranslationOptions` in all translation.py files
- The import of the register decorator is now `from modeltranslation.decorators import register`
- The import of translator is now `from modeltranslation.translator import translator`

2.12.2 wagtail-modeltranslation 0.11.0 release notes

Changelist:

- dropped support for wagtail < 2.13
- dropped suport for python < 3.6
- added compatibility with Python 3.8, 3.9 and 3.10

2.12.3 wagtail-modeltranslation 0.12.0 release notes

Changelist:

- dropped support for wagtail < 3.0
- dropped suport for python < 3.7

2.12.4 wagtail-modeltranslation 0.13.0 release notes

Changelist:

- dropped support for wagtail < 4.0
- added support for wagtail >= 4.0

2.12.5 wagtail-modeltranslation 0.14.0 release notes

Changelist:

- dropped support for wagtail < 5.0
- added support for wagtail >= 5.0
- added tests for python 3.11

InfoPortugal, S.A. - https://github.com/infoportugal

2.13 Authors

2.13.1 Core Committers

- Alexandre Silva
- Diogo Marques
- Eduardo Nogueira
- Rui Martins
- Tiago Costa

2.13.2 Contributors

- Django-modeltranslation
- Django-linguo
- Alain Gelinas
- Karl Hobley
- Raphael Grill
- Tom Dyson
- Tim Tan
- Pomax