TransManager Documentation

Release 0.2.1

Andreu Vallbona

Jul 05, 2018

Contents

TransManager is a Django app that deals with the translation tasks of the django-hvad based models. It handles the translations tasks of the translatable fields.

CHAPTER 1

Installation

In order to install **TransManager** you need to do it via `pip install transmanager`.

Then you've to append `transmanager` and `django_tables2` to `INSTALLED_APPS` in your settings.

```
INSTALLED_APPS = (
    ...
    'transmanager',
    'django_tables2',
    'haystack',
)
```

Execute the migrate command in order to create the models.

```
python manage.py migrate transmanager
```

If you install **TransManager** in an app with already existing data, you can use the following command in order to generate the translations tasks.

```
python manage.py generate_tasks
```

As **Transmanager** comes with a translations suggestion system enable by default, we have to generate the index of the suggestions, which relies on a haystack search index. First we need to add the HAYSTACK_CONNECTIONS to the main app settings:

```
HAYSTACK_CONNECTIONS = {
    'default': {
        'ENGINE': 'haystack.backends.whoosh_backend.WhooshEngine',
        'PATH': os.path.join(os.path.dirname(__file__), 'whoosh_index'),
    },
}
HAYSTACK_SIGNAL_PROCESSOR = 'haystack.signals.RealtimeSignalProcessor'
```

It's recommended to activate the realtime signal processos in order to have the search index updated. You can also cron the command `python manage.py update_index`. Take a look to the Django-haystack documentation for further information.

CHAPTER 2

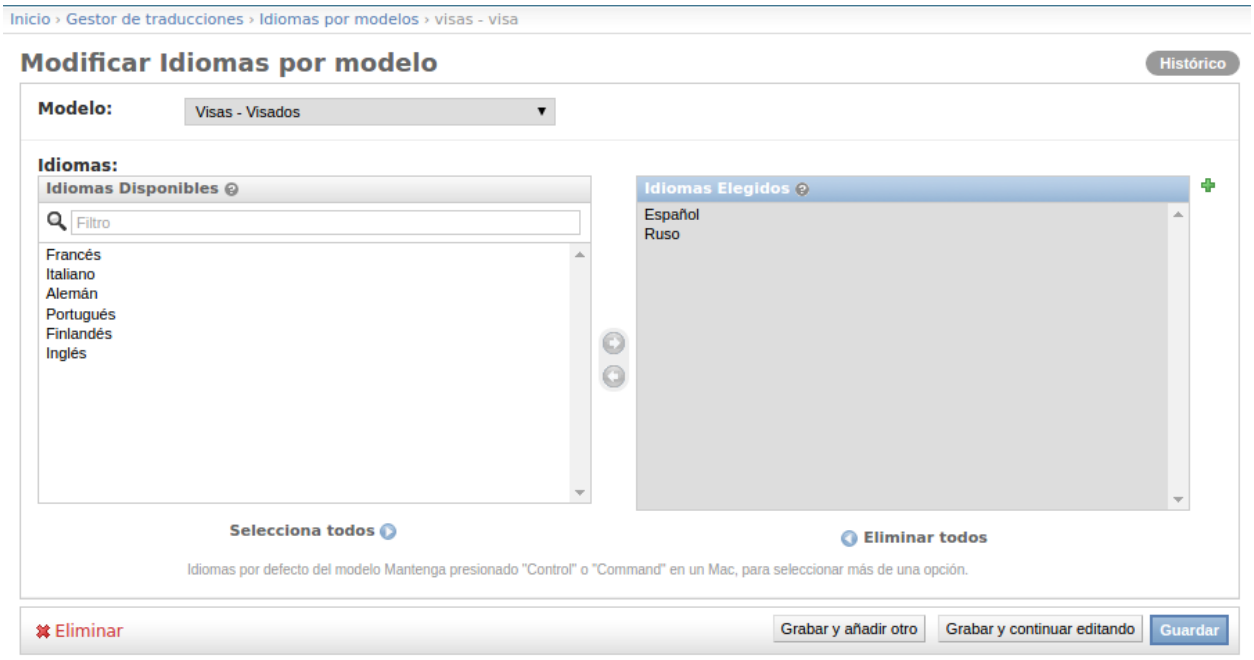
How it works

TransManager its based on the handling pre_save django signal. This allow to detect the changes done in the translatable fields of a model. For each one of the fields that have their original content changed we generate as many translation tasks as the configured languages per app or model we're editing the fields on.

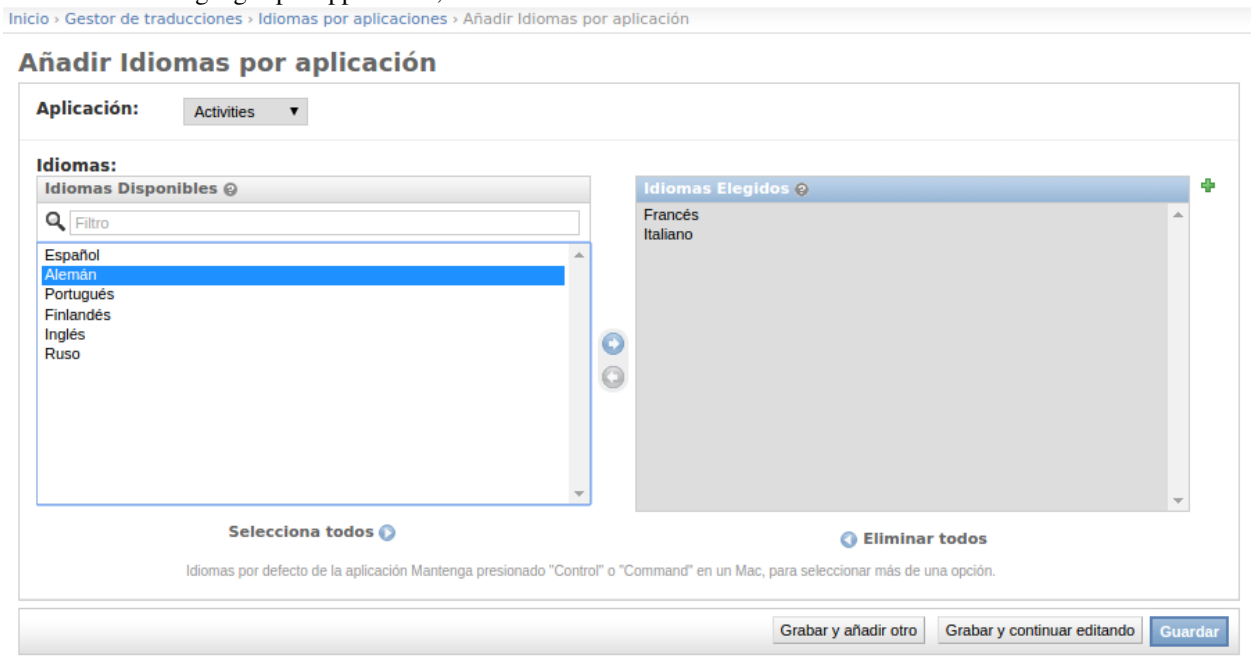
Initial setup

The system has to be configured as follows, we have to create the languages we want the translations into and define the which one of the is the original language of the content.

Then every multilang model has to have set the languages we want the translations into.

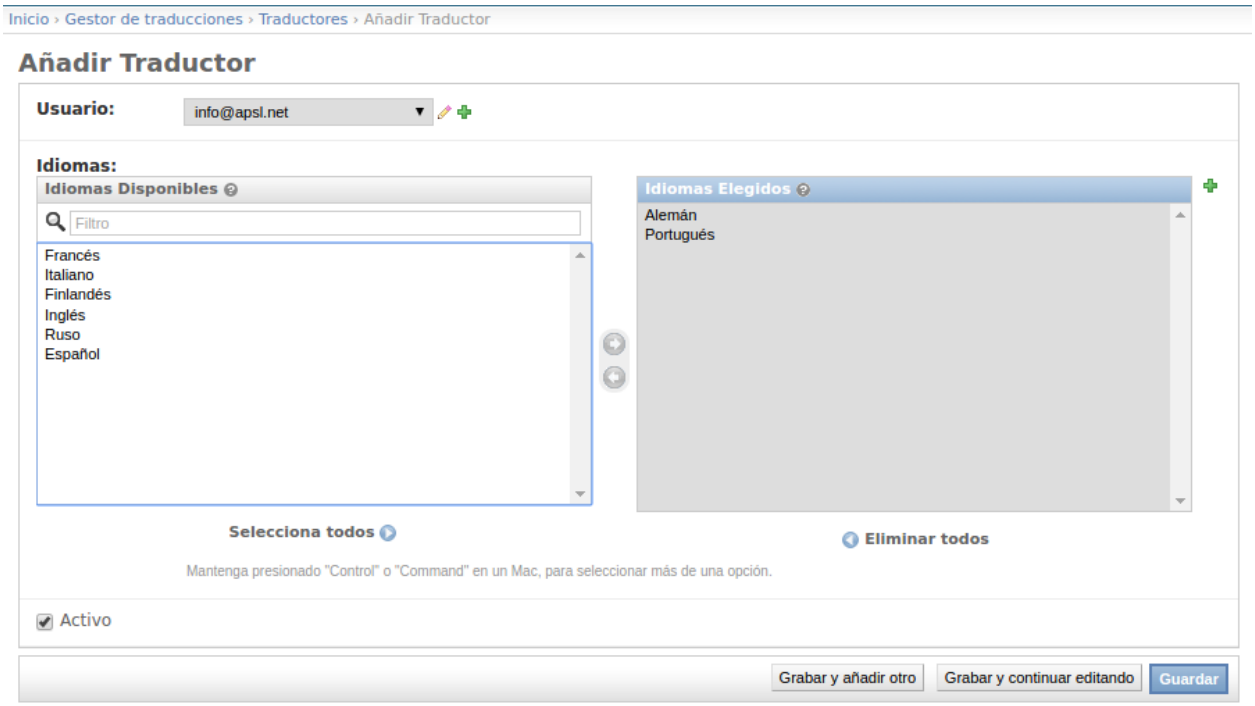


We can set the languages per app as well,



in this case if a model has no languages configured we'll take the model app specified languages.

Besides, every language defined on the TransManager admin has to have an user assigned. An user can have more than one languages assigned. If there is a language without a user, the translations tasks will not be created. Every translator is associated with a django user.



Creation of new record

When a new record is created we create a new translation task for every translatable not empty field in every language configured for the model we're the new record.

Modification of a record

When we modify a register, we will create a translation task for every modified translatable field in the original language.

Translating the tasks

Once the tasks are created they have to be translated. When we translate the task, having the main language original text as reference in the edition form, at the moment of saving the task the main object will be updated. This way it have the advantage that the translator user does not have to have access to the main content models, avoiding undesired deletions or modifications on these main models.

Translation a specific records

In Transmanager, every time we insert or modify a content in the main language, a task is generated for every language defined in the model we are editing.

Nevertheless, we have the possibility to order a translation in one or more languages ONLY for an specific record of the model, even if the language(s) are not set by default in the model we're editing . This can be done throught the "Translations menu" that we can add to our edit template via TransManager template tag

```
{% load transmanager_tags %}
{% if object %}
    {% show_list_translations object %}
{% endif %}
```

From the translation menu, we can order a translation for one o more language for the specific item we're editing. We can also delete the translation tasks ordered before in one o more languages for the specific item we're editing. We see as well the default languages for the model (the languages in which the translations tasks will be generated automatically), these default language that have the grey background.

Finally, you can see the default languages for the specific item as well.

Child models

When working with specific records, it can happen that we edit a form that includes a formset. The records of the formset are children of the main form specific record. When we order a translation task for the main specific record we order as well the translation tasks for the records of the related formset.

In order to achieve the behaviour described above, we have to add the method `get_parent` to the formset model, e.g.:

```
class CardCharacteristic(TranslatableModel):

    card = models.ForeignKey(Card, verbose_name=_('Card Experience'), related_name=
↪'characteristics')
    type = models.ForeignKey(CardCharacteristicType, verbose_name=_('Tipo'))
    translations = TranslatedFields(
        value=models.TextField(verbose_name=_('Valor'))
    )

    def __str__(self):
        try:
            return '{} - {} - {}'.format(self.card_id, self.type, self.lazy_
↪translation_getter('value'))
        except AttributeError:
            return '[{}]'.format(self.pk)

    class Meta:
        ordering = ('card', 'type')
        verbose_name = _('Característica Card Experience')
        verbose_name_plural = _('Características Card Experience')

    def get_parent(self):
        return self.card
```

This way when we order a translation task, the generation process will know which the children model is.

Enabling/Disabling a specific record translations

When working with specific records, it can happen that we work on the content in several times and we don't want to generate the translation tasks every time we save the record. In order to accomplish this behaviour, we can add an attribute to the model we're editing that allows us to know if the record is "enabled" or "disabled". **Enabled** means the edition of the record is finished and ready to generate the translation tasks. **Disabled** means the record is not ready yet and we don't want the translation tasks to be generated.

The name of the model attribute can be configured in the settings, via the *TM_DEFAULT_ENABLED_ATTRIBUTE_NAME* constant.

The watch of the value of the attribute above is done by adding the mixin `TranslationMixin` to the model that have the `enabled` attribute:

```
class Card(TranslationTasksMixin, SafeLocationMixin, ModelUniqueIdMixin,␣
↪TranslatableModel):

    translations = TranslatedFields(
        name=models.CharField(max_length=250, verbose_name=_('Nombre')),
        selling_text=models.TextField(verbose_name=_('Texto vendedor'), blank=True,␣
↪null=True),
        short_description=models.TextField(verbose_name=_('Descripción corta'),␣
↪blank=True, null=True),
        large_description=models.TextField(verbose_name=_('Descripción larga'),␣
↪blank=True, null=True),
        important_info=models.TextField(verbose_name=_('Información importante'),␣
↪blank=True, null=True),
    )
    code = models.CharField(max_length=50, verbose_name=_('Código'), unique=True, db_
↪index=True)
    enabled = models.BooleanField(_('Activo'), default=True)

    def __str__(self):
        try:
            return '{} - {}'.format(self.code, self.lazy_translation_getter('name'))
        except AttributeError:
            return '[{}]'.format(self.pk)

    class Meta:
        verbose_name = _('Card Experience')
        verbose_name_plural = _('Cards Experience')
```

The mixin controls the addition/deletion of the translations tasks.

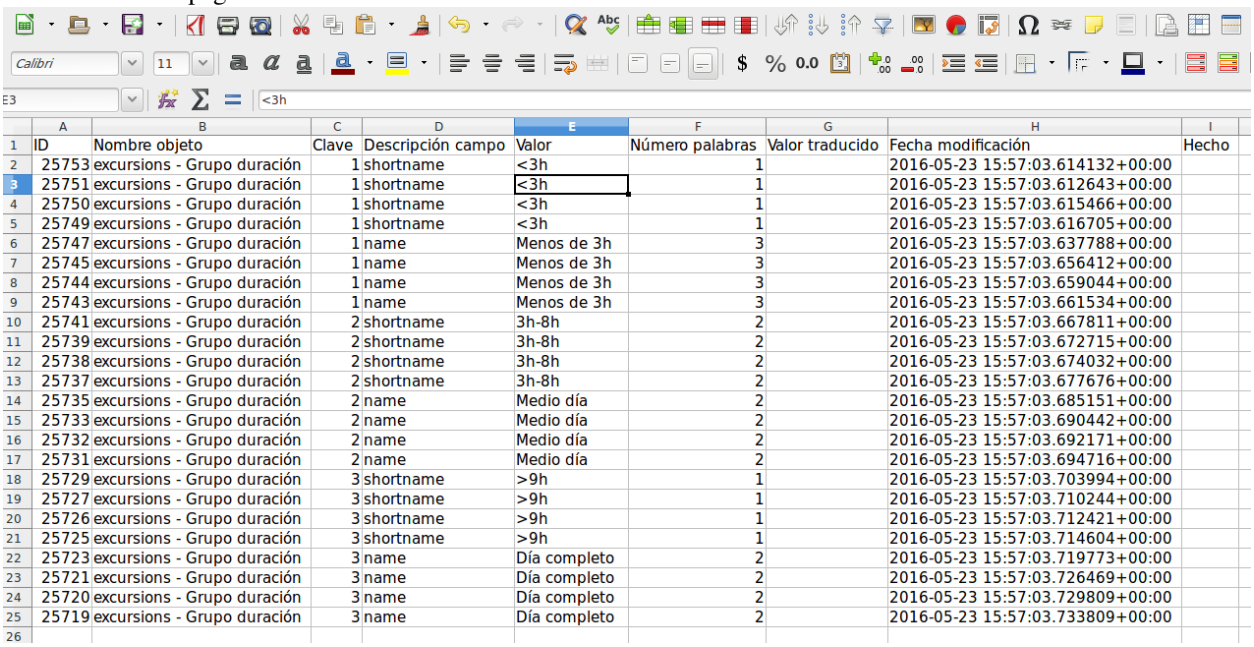
```
class TranslationTasksMixin(object):
"""
Mixin that allows to create/delete the translations tasks when the instance of the␣
↪model is enabled/disabled
"""

def save(self, force_insert=False, force_update=False, using=None, update_
↪fields=None):

    create = False
    delete = False
```

```
    # get the previous instance
    if self.pk:
        prev = self.__class__.objects.get(pk=self.pk)
    else:
        prev = None

    # decide what to do
    if not self.pk and self.enabled:
        # new instance
        create = True
    elif self.pk and self.enabled and prev and not prev.enabled:
        # from disabled to enabled
        create = True
    elif self.pk and not self.enabled and prev and prev.enabled:
        # from enabled to disabled
        delete = True

    super().save(force_insert, force_update, using, update_fields)

    if create:
        create_translations_for_item_and_its_children.delay(self.__class__, self.pk)
    elif delete:
        delete_translations_for_item_and_its_children.delay(self.__class__, self.pk)
```

Because there may be lots of translations tasks to generate, the process is called asyncronously, via the python-rq workers.

Export to Excel

Translation tasks can be listed and filtered by several criteria, and then the list can be exported to excel via the export button on the list page.



	A	B	C	D	E	F	G	H	I
1	ID	Nombre objeto	Clave	Descripción campo	Valor	Número palabras	Valor traducido	Fecha modificación	Hecho
2	25753	excursions - Grupo duración	1	shortname	<3h	1		2016-05-23 15:57:03.614132+00:00	
3	25751	excursions - Grupo duración	1	shortname	<3h	1		2016-05-23 15:57:03.612643+00:00	
4	25750	excursions - Grupo duración	1	shortname	<3h	1		2016-05-23 15:57:03.615466+00:00	
5	25749	excursions - Grupo duración	1	shortname	<3h	1		2016-05-23 15:57:03.616705+00:00	
6	25747	excursions - Grupo duración	1	name	Menos de 3h	3		2016-05-23 15:57:03.637788+00:00	
7	25745	excursions - Grupo duración	1	name	Menos de 3h	3		2016-05-23 15:57:03.656412+00:00	
8	25744	excursions - Grupo duración	1	name	Menos de 3h	3		2016-05-23 15:57:03.659044+00:00	
9	25743	excursions - Grupo duración	1	name	Menos de 3h	3		2016-05-23 15:57:03.661534+00:00	
10	25741	excursions - Grupo duración	2	shortname	3h-8h	2		2016-05-23 15:57:03.667811+00:00	
11	25739	excursions - Grupo duración	2	shortname	3h-8h	2		2016-05-23 15:57:03.672715+00:00	
12	25738	excursions - Grupo duración	2	shortname	3h-8h	2		2016-05-23 15:57:03.674032+00:00	
13	25737	excursions - Grupo duración	2	shortname	3h-8h	2		2016-05-23 15:57:03.677676+00:00	
14	25735	excursions - Grupo duración	2	name	Medio día	2		2016-05-23 15:57:03.685151+00:00	
15	25733	excursions - Grupo duración	2	name	Medio día	2		2016-05-23 15:57:03.690442+00:00	
16	25732	excursions - Grupo duración	2	name	Medio día	2		2016-05-23 15:57:03.692171+00:00	
17	25731	excursions - Grupo duración	2	name	Medio día	2		2016-05-23 15:57:03.694716+00:00	
18	25729	excursions - Grupo duración	3	shortname	>9h	1		2016-05-23 15:57:03.703994+00:00	
19	25727	excursions - Grupo duración	3	shortname	>9h	1		2016-05-23 15:57:03.710244+00:00	
20	25726	excursions - Grupo duración	3	shortname	>9h	1		2016-05-23 15:57:03.712421+00:00	
21	25725	excursions - Grupo duración	3	shortname	>9h	1		2016-05-23 15:57:03.714604+00:00	
22	25723	excursions - Grupo duración	3	name	Día completo	2		2016-05-23 15:57:03.719773+00:00	
23	25721	excursions - Grupo duración	3	name	Día completo	2		2016-05-23 15:57:03.726469+00:00	
24	25720	excursions - Grupo duración	3	name	Día completo	2		2016-05-23 15:57:03.729809+00:00	
25	25719	excursions - Grupo duración	3	name	Día completo	2		2016-05-23 15:57:03.733809+00:00	
26									

It can be usefull in order to check the number of words translated for some user in a certain period of time.

CHAPTER 3

Settings

TransManager provides setting variables in order to customize your installation.

TM_DISABLED

This property allow us to disable Transmanager. It can be useful for test environments. Default `False`.

TM_DEFAULT_LANGUAGE_CODE

The default language code in **ISO 639-1** format. Default `'es'`.

TM_DEFAULT_ENABLED_ATTRIBUTE_NAME

The name of the attribute that we have to watch in order to know if the record of the main model is enabled or not. Watching the state of this field we order to generate or delete the translations of the main model record. Default `'enabled'`. See the *Enabling/Disabling a specific record translations* section.

TM_API_URL

Url of the API of **TransManager**. Default `'/transmanager/api/task/'`.

TM_BRAND_LOGO_URL

Url of logo with which you can "brand" the list and edit tasks pages. Default `'transmanager/img/logo.png'`.

TM_ORIGINAL_VALUE_CHARS_NUMBER

Number of chars that we want to show from the value to translate in the translation task list. Default `100`.

TM_HAYSTACK_DISABLED

This property allow us to disable the enabled by default suggestion system via the "haystack" search engine. Default `False`.

TM_HAYSTACK_SUGGESTIONS_MAX_NUMBER

Maximum number of translation suggestions we want to show in the translation task edit form. Default `20`.

CHAPTER 4

Commands

There are several commands that help to keep clean and keen the translation tasks list.

Generate tasks

Generate the translation tasks from the existing data. It's useful if we have installed **TransManager** in an already existing and running application.

```
python manage.py generate_tasks
```

Because there may be lots of translations tasks to generate, the process is called asyncronously, via the python-rq workers.

Delete orphan tasks

This commands deletes the tasks that no longer have a related main object. It is a command that can be executed from time to time or it can be added to the *cron*.

```
python manage.py delete_orphan_tasks
```

Notify translators

Command that notifies the translators of pending translation tasks. Its a command that, if we want our translator users notified of the pending tasks, we have to *cron* in order to send the nofitication email to them.

```
python manage.py notify_translators
```

Update number of words

Update the number of words to translate in every task. It counts the number of words of the original text to translate.

```
python manage.py update_number_of_words
```

Delete disabled parent tasks

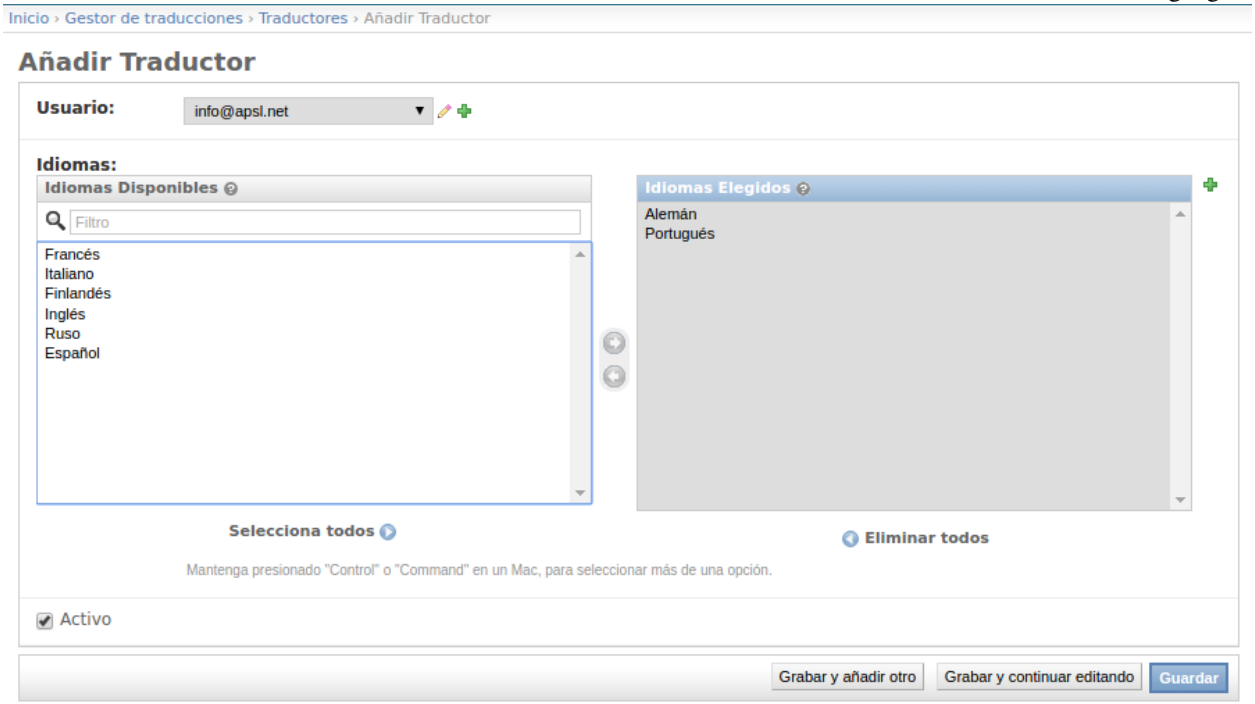
This commands deletes the tasks that no longer have a parent enabled object. It is a command that can be executed from time to time or it can be added to the *cron*.

```
python manage.py remove_tasks_for_disbled_parents
```

CHAPTER 5

Users

You have to create users that will make the translation work. At least there must be a user created for each language.



A user can handle more than one language, although normally there is one user per language.

Each time a translation task is generated it will be assigned to the user that handles the language specifid in that task.

Notification to user

Once every language we've defined in the application has its user, the users will begin to receive the nofitications of the pending translations tasks, once a day normally.

The notifications are set via cron, e.g:

```
# notify_translators
00 07 * * 1-5 python manage.py notify_translators  >>  $HOME/logs/cron_notify_
↪translators.log 2>&1
```

From the notification email, the user can access directly to the translation task by clicking on the detail task link.

Scope of the translator users

The users, also know as translators, only have access to the translation. None of the translators have access to the main content models. This way we avoid the risk of the deletion or modification of the original content and we have not to define translations rols into our main application. TransManager works in a complete detached way.