django-klingon Documentation

Release 0.0.7

Angel Velasquez

February 21, 2017

Contents

Contents:

CHAPTER 1

klingon

Welcome to the documentation for django-klingon!

django-klingon is an attempt to make django model translations suck but with no integrations pain in your app!

Setup & Integration

In your settings files: add django-klingon to INSTALLED_APPS:

```
INSTALLED_APPS = (
    ...
    'klingon',
    ...
)
```

specify a default language if you want to use your fields to store the default language:

```
KLINGON_DEFAULT_LANGUAGE = 'en'
```

Extend you models to add API support: first add Translatable to your model Class definition. This will add the API functions:

```
from klingon.models import Translatable
...
class Book(models.Model, Translatable):
...
```

in the same model add an attribute to indicate which fields will be translatables:

```
...
translatable_fields = ('title', 'description')
...
```

your model should look like this:

```
class Book(models.Model, Translatable):
    title = models.CharField(max_length=100)
    description = models.TextField()
    publication_date = models.DateField()

    translatable_fields = ('title', 'description')
```

Add admin capabilities:

you can include an inline to your model admin and a custom action to create the translations. To do this in your ModelAdmin class do this:

```
from klingon.admin import TranslationInline, create_translations
...
class BookAdmin(admin.ModelAdmin):
    ...
    inlines = [TranslationInline]
    actions = [create_translations]
```

- see full example in example_project folder of source code of klingon

Using Specific Widgets in the TranslationInline form of the admin:

You can specify the widget to be use on an inline form by passing a dictionary to TranslationInlineForm. So, you might want to extend the TranslationInline with a new form that will a "widgets" dictionary, where you can specify the widget that each filds has to use, for example:

```
class RichTranslationInlineForm(TranslationInlineForm):
    widgets = {
        'CharField': forms.TextInput(attrs={'class': 'klingon-char-field'}),
        'TextField': forms.Textarea(attrs={'class': 'klingon-text-field'}),
    }

class RichTranslationInline(TranslationInline):
    form = RichTranslationInlineForm
```

and then you just simply use the RichTranslationInline class on your AdminModels, for example:

```
class BookAdmin(admin.ModelAdmin):
    inlines = [RichTranslationInline]
```

- see full example in example_project folder of source code of klingon

Using the API

To create the translation you can do the follwing:

Suppose that you have and object called book:

```
> book = Book.objects.create(
    title="The Raven",
    description="The Raven is a narrative poem",
    publication_date=datetime.date(1845, 1, 1)
)
```

you can create translation for that instances like this:

```
> book.set_translation('es', 'title', 'El Cuervo')
> book.set_translation('es', 'description', 'El Cuervo es un poema narrativo')
```

a translation could be access individually:

```
> self.book.get_translation('es', 'title')
'El Cuervo'
> book.get_translation('es', 'description')
'El Cuervo es un poema narrativo'
```

or you can get all translations together:

```
> self.book.translations('es')
{
    'title': self.es_title,
    'description': self.es_description,
}
```

Installation:

pip install django-klingon

Running the Tests

You can run the tests with via:

```
python setup.py test
```

or:

```
python runtests.py
```

CHAPTER 2

Installation

At the command line:

```
$ easy_install django-klingon
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django-klingon
$ pip install django-klingon
```

CHAPTER 3

Usage

Using Specific Widgets in the TranslationInline form of the admin:

You can specify the widget to be use on an inline form by passing a dictionary to TranslationInlineForm. So, you might want to extend the TranslationInline with a new form that will a "widgets" dictionary, where you can specify the widget that each filds has to use, for example:

```
class RichTranslationInlineForm(TranslationInlineForm):
    widgets = {
        'CharField': forms.TextInput(attrs={'class': 'klingon-char-field'}),
        'TextField': forms.Textarea(attrs={'class': 'klingon-text-field'}),
    }

class RichTranslationInline(TranslationInline):
    form = RichTranslationInlineForm
```

and then you just simply use the RichTranslationInline class on your AdminModels, for example:

```
class BookAdmin(admin.ModelAdmin):
    inlines = [RichTranslationInline]
```

- see full example in example_project folder of source code of klingon

Using the API

To create the translation you can do the follwing:

Suppose that you have and object called book:

```
> book = Book.objects.create(
    title="The Raven",
    description="The Raven is a narrative poem",
    publication_date=datetime.date(1845, 1, 1)
)
```

you can create translation for that instances like this:

```
> book.set_translation('es', 'title', 'El Cuervo')
> book.set_translation('es', 'description', 'El Cuervo es un poema narrativo')
```

a translation could be access individually:

```
> self.book.get_translation('es', 'title')
'El Cuervo'
> book.get_translation('es', 'description')
'El Cuervo es un poema narrativo'
```

or you can get all translations together:

```
> self.book.translations('es')
{
    'title': self.es_title,
    'description': self.es_description,
}
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/angvp/django-klingon/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

django-klingon could always use more documentation, whether as part of the official django-klingon docs, in doc-strings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/angvp/django-klingon/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *django-klingon* for local development.

1. Fork the *django-klingon* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-klingon.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv django-klingon
$ cd django-klingon/
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 klingon tests
$ python setup.py test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website to the `develop` repo.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, and 3.4, and for PyPy. Check https://travis-ci.org/angvp/django-klingon/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ python -m unittest tests.test_klingon
```

Rafael Capdevielle Angel Velasquez

CHAPTER 5

History

0.0.7 (2017-1-7)

- Removed support for Django 1.5 and 1.6 now klingon works from Django 1.7 version in advance

0.0.4 (2015-1-2)

- Add translatable_slug and a painless integration with klingon + django-autoslug.