froide Documentation

Release 1.0a1

Stefan Wehrmeyer

March 01, 2013

CONTENTS

Froide is a Freedom of Information platform written in Python using the Django Web framework. It manages public bodies and FoI requests. Users can send emails to public bodies and receive answers via the platform.

It was developed to power Frag den Staat – the German Freedom of Information Portal.

CHAPTER

ONE

ABOUT

Froide was designed to mimic the functionality of What do they know – a Freedom of Information portal in the UK written in Ruby on Rails 2.3. At the time when a German FoI portal was needed, the general FoI solution forked from WDTK called Alaveteli was hard to install and not ready for reuse. That's why Froide was developed as a fresh start, fully internationalized and configurable written in Django 1.3 to power Frag den Staat.

1.1 Development Goals

Froide has some development goals that are listed below. Some of them are a continuous effort, some are achieved, on others development is still ongoing.

- Internationalization (i18n): Keep the code fully internationalized and localized.
- Flexible and Configurable: Many aspects of an FoI platform depend on local customs and laws. These aspects should be either configurable via settings or easily replaceable.
- Easy to install: Keep dependencies to one language environment (Python) and use abstraction layers for backends like search, caching etc. to enable different setups.
- Maintain a test suite with a high test coverage.

1.2 Features

- Froide uses many of the built-in Django features like the Admin interface to manage and update entities in the system, the internationalization system, and the user management and authentication.
- Freedom of Information Laws and Public Entities are connected through a many-to-many relationship. That allows for a Public Body to be accountable under different laws.
- A Public Body can have a parent to represent hierarchies from the real world. They can also be categorized into classifications (e.g. ministry, council) and topics (e.g. environment, military) which can be defined separately.
- Users can create requests without a Public Body so that others can suggest an appropriate recipient later.
- Requests can optionally be kept private by users and published at a later point (e.g. after a related article has been published).
- Requests are mailed to Public Bodies through the platform via a special, request-unique email address (using SMTP) and the platform will receive answers on that mail address (by accessing an IMAP account).
- Search functionality for Requests and Public Bodies.
- Error Reporting interface via Sentry.

1.3 Dependencies

A detailed list of Python package dependencies can be found in *requirements.txt*, but here is a general overview:

- Django 1.3 - the Web framework
- South 0.7.3 - the database migration framework (development dependency)
- Sphinx 1.0.7 - the documentation tool (development dependency)
- Celery 2.2.5 - task queue for background processing
- Haystack 1.2.0 - abstraction layer for search

A development goal is that, even though a task queue (like Celery) and a search server (like Solr) are highly recommended, they are not necessary for either development or production setup and can be replaced with Cronjobs and database queries respectively (results/performance will probably degrade, but it should work).

1.4 Name

Froide stems from “Freedom of Information de” (de for Germany) and sounds like the German word “Freude” which means joy.

CHAPTER

TWO

GETTING STARTED

This is a guide that will get you started with Froide in no time. Some more advanced features are discussed at the end.

2.1 Set up the development environment

You should be using *virtualenv* and it is suggested you also use *virtualenvwrapper*. Setup a virtual environment for development like so:

```
mkvirtualenv --no-site-packages froide
```

Get the source code with Git from the official GitHub repository or from your fork:

```
git clone git://github.com/stefanw/froide.git
cd froide
```

Install the requirements inside the virtual env with *pip*:

```
which pip
<should display your virtual env pip>
pip install -r requirements.txt
```

If only your global *pip* is available, run *easy_install pip*. The dependency installation takes a couple of seconds, but after that everything is in place.

Copy *local_settings.py.example* to *local_settings.py*:

```
cd froide
cp local_settings.py.example local_settings.py
```

The development environment uses SQLite. You can change that in *local_settings.py*, if you want, but you don't have to. Sync and migrate and *do NOT* create a superuser just yet:

```
python manage.py syncdb --noinput --migrate
```

Now you can create a superuser account:

```
python manage.py createsuperuser
```

That's it for a setup that basically works. Run this:

```
python manage.py runserver
```

and go to http://localhost:8000. You should be greeted by a working Froide installation. It doesn't have any data inside, but you can change that by going to http://localhost:8000/admin/ and logging in with your superuser account.

For more information on the different models you find in the admin visit *Froide Data Model*.

2.2 Run tests

Froide has a test suite. Copy *test_settings.py.example* to *test_settings.py*. *test_settings.py* does not import your *local_settings.py* changes.

You can then run the shell script for tests:

```
sh runtests.sh
```

This also does timing and a test coverage analysis that you can then find at *htmlcov/index.html*. Note some tests will not work without a search engine like solr running.

2.3 Search with Haystack and Solr

2.4 Background Processing with Celery

CHAPTER

THREE

FROIDE DATA MODEL

The following will describe the most important entities, their important fields and their relation to each other. Froide uses South for database migrations, so if you find yourself in need of an additional field to cover some data point, do not hesitate to add it to your fork, migrating is easy.

3.1 Public Body

3.2 FoiLaw

3.3 FoiRequest

3.4 FoiMessage

3.5 FoiAttachment

3.6 PublicBodySuggestion

3.7 FoiRequestFollower

CHAPTER

FOUR

CONFIGURATION

Froide can be configured in many ways to reflect the needs of your local FoI portal.

All configuration is kept in the Django *settings.py* file. Individual settings can be overwritten by placing a *local_settings.py* file on the Python path (e.g. in the same directory) and redefining the configuration key in there.

4.1 Froide Configuration

There is a dictionary called *FROIDE_CONFIG* inside settings.py that acts as a namespace for some other configurations. These settings are also available in the template via the name *froide* through the context processor *froide.helper.context_processors.froide*.

The following keys in that dictionary must be present:

create_new_publicbody *boolean* Are users allowed to create new public bodies when making a request by filling in some details? Newly created public bodies must be approved by an administrator before the request is sent.

publicbody_empty *boolean* Can users leave the public body empty on a request, so other users can suggest an appropriate public body later?

users_can_hide_web *boolean* Can users hide their name on the portal? Their name will always be sent with the request, but may not appear on the website.

public_body_officials_public *boolean* Are the names of responding public body officials public and visible on the Web?

public_body_officials_email_public *boolean* Are the email addresses of public body officials public and visible on the Web?

currency *string* The currency in which payments (if at all) occur

default_law *integer* The id of the Freedom of Information law in the database that is used by default (e.g. 1)

4.2 Greeting Regexes

To detect names and beginning and endings of letters the standard settings define a list of common English letter greeting and closing regexes that also find the name:

```python
import re
rec = re.compile
# define your greetings and closing regexes
POSSIBLE_GREETINGS = [rec(u"Dear (?:Mr\.?|Ms\.? .*?)")]
POSSIE_CLOSINGS = [rec(u"Sincerely yours,?")]
```

You should replace this with a list of the most common expressions in your language.

4.3 Index Boosting of Public Bodies

Some Public Bodies are more important and should appear first in searches (if their name and description match the search terms). You can provide a mapping of public body classifications (e.g. ministry, council etc.) to their search boost factor via the *FROIDE_PUBLIC_BODY_BOOSTS* setting:

```
# boost public bodies by their classification
FROIDE_PUBLIC_BODY_BOOSTS = {
    u"Ministry": 1.9,
    u"Council": 0.8
}
```

4.4 Settings for Sending E-Mail

You must adapt the standard Django parameters for sending email. Configure the backend depending on your environment (development vs. production):

```
# development environment:
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
# production environment:
EMAIL_BACKEND = 'djcelery_email.backends.CeleryEmailBackend'
```

Define the standard Django SMTP parameters for sending regular email notifications (not FoI request emails to public bodies):

```
EMAIL_HOST = "smtp.example.com"
EMAIL_PORT = 587
EMAIL_HOST_USER = "mail@foi.example.com"
EMAIL_HOST_PASSWORD = "password"
EMAIL_USE_TLS = True
```

Also define the parameters for sending FoI-Mails to public bodies. They might be different because they can either be sent from a fixed address and with a special *Reply-To* field or directly from a special address:

```
# Sends mail from a fixed from address with Reply-To field
FOI_EMAIL_FIXED_FROM_ADDRESS = True
FOI_EMAIL_HOST_USER = "foirelay@foi.example.com"
FOI_EMAIL_HOST_PASSWORD = "password"
FOI_EMAIL_HOST = "smtp.example.com"
FOI_EMAIL_PORT = 537
FOI_EMAIL_USE_TLS = True
```

Finally give the IMAP settings of the account that receives all FoI email. This account is polled regularly and the messages are processed and displayed on the website if their *To* field matches:

```
FOI_EMAIL_DOMAIN = "foi.example.com"
FOI_EMAIL_PORT_IMAP = 993
FOI_EMAIL_HOST_IMAP = "imap.example.com"
FOI_EMAIL_ACCOUNT_NAME = "foirelay@foi.example.com"
FOI_EMAIL_ACCOUNT_PASSWORD = "password"
```

4.5 Public Body E-Mail Dry-run

You can set your site up and test it out in a production environment while sending public body emails not to the public bodies but to another mail server. Use the following settings:

```
FROIDE_DRYRUN = True
FROIDE_DRYRUN_DOMAIN = "mymail.example.com"
```

This converts public body email addresses from

> public-body@example.com

to

> public-body+example.com@mymail.example.com

right before the mail is sent out (the changed address is not stored). This allows for some testing of sending and receiving mails to and from public bodies wihtout spamming them.

4.6 Setting Up Search with Solr

Froide uses *django-haystack* to interface with a search. Solr is recommended, but thanks to *django-haystack* you can use something else as well.

Haystack configuration for solr works like so:

```
HAYSTACK_SITECONF = 'froide.search_sites'
HAYSTACK_SEARCH_ENGINE = 'solr'
HAYSTACK_SOLR_URL = 'http://127.0.0.1:8983/solr'
```

If you have a solr multicore setup, your solr URL would probably look more like this:

```
HAYSTACK_SOLR_URL = 'http://127.0.0.1:8983/solr/froide'
```

For details, please refer to the Haystack Documentation.

4.7 Setting Up Background Processing with Celery

The following part in *settings.py* does the configuration of Celery. Overwrite the *CELERY** values with your own in *local_settings.py*:

```
import djcelery
djcelery.setup_loader()

CELERY_IMPORTS = ("foirequest.tasks", )

CELERY_RESULT_BACKEND = "database"
CELERY_RESULT_DBURI = "sqlite:///dev.db"

CELERYBEAT_SCHEDULER = "djcelery.schedulers.DatabaseScheduler"
```

For details please refer to the django-celery documentation.

4.8 Some more settings

Configure the name and default domain URL (without trailing slash) of your site with the following settings:

```
SITE_NAME = 'FroIde'
SITE_URL = 'http://localhost:8000'
```

You can give a URL with string formatting placeholders *query* and *domain* in them that will be presented to the user as the URL for web searches via the setting *SEARCH_ENGINE_QUERY*. The default is a Google search.

4.9 Securing your site

It may be a good idea to NOT use easily guessable URL paths for specific parts of the site, specifically the admin. To make these parts configurable by *local_settings* you can use the following setting:

```
SECRET_URLS = {
    "admin": "my-secret-admin",
    "sentry": "my-secret-sentry"
}
```

It's also recommended to protect the admin and sentry further via HTTP auth in your production reverse proxy (e.g. nginx).

The app djangosecure is part of Froide and it is highly recommended to deploy the site with SSL (get a free SSL certificate from StartSSL).

Some Django settings related to security and SSL:

```
CSRF_COOKIE_SECURE = True
CSRF_FAILURE_VIEW = 'froide.account.views.csrf_failure'

SESSION_COOKIE_AGE = 3628800 # six weeks for usability
SESSION_COOKIE_HTTPONLY = True
SESSION_COOKIE_SECURE = True
```

CHAPTER

FIVE

PRODUCTION SETUP

This page will describe how to setup Froide for a production ready setup. This is an example, your personal flavor of how to set this up may vary.

CHAPTER

SIX

INDICES AND TABLES

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