froide Documentation

Release 3.0.0

Stefan Wehrmeyer

March 14, 2013

CONTENTS

Froide is a Freedom of Information portal written in Python using the Django Web framework. It manages public bodies, FoI requests and much more. 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 is a Freedom of Information portal software written in Python with Django 1.5.

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.
- A read/write REST-API
- Redaction of PDFs

1.3 Dependencies

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

- Django 1.5 - the Web framework
- Celery 3.X - task queue for background processing
- Haystack 2.X-beta - 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 will work nonetheless).

1.4 History

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 in spring 2011 as a fresh start, fully internationalized and configurable written in Django 1.3 to power Frag den Staat which launched in August 2011.

1.5 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 a Python virtual environment. Setup a virtual environment for development with ‘virtualenv‘like so:

```
virtualenv --no-site-packages ~/pyenv
. ~/pyenv/bin/activate
```

Get the source code with Git from the GitHub repository:

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

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

```
pip install -r requirements.txt
```

The dependency installation may take a couple of minutes, but after that everything is in place.

Sync and migrate and *do NOT* create a superuser just yet:

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

Now you can already start the web server:

```
python manage.py runserver
```

Visit http://localhost:8000 and there is your running Froide instance!

You can quit the server (Ctrl+C) and create a superuser account:

```
python manage.py createsuperuser
```

2.2 Add basic database objects

Run the web server again and login to the admin interface at http://localhost:8000/admin/ with the credentials of your superuser.

The first thing you should do is create a jurisdiction. Click on "Jurisdiction" in the "Publicbody" section and then on "Add Jurisdiction". Name your jurisdiction however you want (e.g. Federal). If you only ever intend to have one, the name will not even show up. Click "Save" to continue.

Go back into the public body section and add an FOI law. Give it a name (e.g. Freedom of Information Act) and choose a jurisdiction. There are many more things that can be configured, but you can leave them for now.

Now you can add your first public body by going back to the public body section and clicking on "Add" next to "Public Bodies". Give it a name (e.g. Ministry of the Interior). Click on the little plus-sign next to topic to add a topic for this public body. The classification is to distinguish different areas of government (e.g. "Ministry", "Council"). If you want to make a request to this public body, it needs to have an email address. Select your previously created jurisdiction and FOI law and save.

You should also fill out your user details like your name in the user section of the admin.

Now you are ready to make your first request. Go back to the front page and give it a go. You can also find out more about the `admin`.

2.3 Custom configuration

By default the Django Web server uses the *settings.py* file in the froide directory (the *froide.settings* Python module). This will be fine for your first experiments but if you want to customize your froide instance you will want your own settings file.

Go into the froide directory and copy the *custom_settings.py.example* to *custom_settings.py*:

```
cd froide
cp custom_settings.py.example custom_settings.py
```

Now you can customize that settings file to your liking. Note that it imports the original *settings.py* at the top, so you only have to change the values you need to change.

In order for Django to start your web server with these new settings you either have to set the environment variable *DJANGO_SETTINGS_MODULE* or give the settings module as a command line argument.

Either type:

```
export DJANGO_SETTINGS_MODULE=froide.custom_settings
python manage.py runserver
```

The environment variable will be set for this shell session. If you open a new shell, you have to export the settings again. Or:

```
python manage.py runserver --settings=froide.custom_settings
```

Since it can be annoying to type this (or to remember to export the environment variable), you can create a shortcut shell script, create an alias, put an export in your virtualenv's *activate* file or use virtualenvwrapper and put the export in your *postactivate* hook.

2.4 Search with Haystack

In order to get a real search engine running you need to override the *HAYSTACK_CONNECTIONS* setting with the details of your search engine. Find out how to configure your search engine at the Haystack Docs.

An example configuration for solr would look like this:

```
HAYSTACK_CONNECTIONS = {
    'default': {
        'ENGINE': 'haystack.backends.solr_backend.SolrEngine',
        'URL': 'http://127.0.0.1:8983/solr/froide'
    }
}
```

2.5 Background Tasks with Celery

From the standard settings file everything is already setup for background tasks except that they are not running in the background.

You need to change the *CELERY_ALWAYS_EAGER* setting to *False* in your custom settings:

```
CELERY_ALWAYS_EAGER = False
```

You need a broker for Celery. Find out more at the Celery Docs.

We recommend RabbitMQ as broker. Install it and then start it in a different terminal like this:

```
rabbitmq-server
```

After you started the broker open yet another terminal, activate your virtual environment and run the celery worker like this:

```
python manage.py celeryd -l INFO -B
```

Now your server will send background tasks to Celery. Lots of common tasks are designed as background tasks so that an ongoing HTTP request can send a response more quickly. The following things are designed as background tasks:

- Search Indexing: Updates to database objects are indexed in the background
- Email Sending: When an action triggers an email, it's sent in the background
- Denormalized Counts on database objects

Celery also takes the role of *cron* and handles periodic tasks. You should set up periodic tasks in the admin under "Djcelery - Periodic tasks". Here is a recommended configuration:

- Fetch Mail: every minute
- Detect Overdue at Midnight: 0 0 * * * (m/h/d/dM/MY)
- Batch Update Followers every 24 hours: 0 0 * * * (m/h/d/dM/MY)
- Remind users to classify there requests: 0 7 6 * * (m/h/d/dM/MY)

CHAPTER

THREE

IMPORTING PUBLIC BODIES

While it is possible to create public bodies individually through the admin interface, it might be more advisable to scrape a list from a website or crowdsource them with other people in a Google doc. You can then import these public bodies as a CSV file into Froide.

The format of the CSV file should be like in this Google Docs Spreadsheet. You could for example copy it and either fill in public bodies collaboratively or programmatically.

3.1 Format

The format is quite simple.

name (required) The name of the public body.

email (optional) If you give no email, users will not be able to make requests to this public body. You can fill in email addresses later through the admin.

jurisdiction__slug (required) Give the slug of the jurisdiction this public body belongs to.

other_names (optional) Possible other, alternative names for the public body separated by commas.

description (optional) A text description of the public body.

topic__slug (optional) Slug for this public body's topic. If not given, the public body topic with the highest rank attribute will be chosen.

url (optional) Website for this public body.

parent__name (optional) if this public body has a parent, give it's name here. The parent must be specified before in the CSV file.

classification (required) Give a classification (e.g. "ministry"). If you don't want have the information, just fill in something general like "public body".

contact (optional) Contact information apart from post address and email. E.g. phone or fax number. May contain line breaks.

address (optional) Postal address of this public body.

website_dump (optional) Any further text that can be used to described this public body. This is used for search indexing and will not be displayed publicly.

request_note (optional) Display this text for this public body when making a request to it.

If during import a public body with the same slug is found, it is skipped and not overwritten.

3.2 Importing through Admin

The admin interface that lists public bodies has an import form at the very bottom of the page. Give a HTTP or HTTPS url of your CSV file and press the import button. The file will be downloaded and imported. Any errors will be shown to you.

3.3 Importing via command line

The management command *import_csv* takes a URL or a path to a CSV file:

```
python manage.py import_csv public_bodies.csv --settings=froide.custom_settings
```

CHAPTER

FOUR

CONFIGURATION

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

The *custom_settings.py.example* file that comes with froide has all the settings from the *settings.py* file but they are commented out. You can copy this file to *custom_settings.py*

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)

search_engine_query *string* 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. The default is a Google search.

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:

```
import re
rec = re.compile
# define your greetings and closing regexes

FROIDE_CONFIG.update(
    dict(
        greetings=[rec(u"Dear (?:Mr\.?|Ms\.? .*?)")],
        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 *public_body_boosts* key in the *FROIDE_CONFIG* setting:

```
# boost public bodies by their classification
FROIDE_CONFIG.update(
    'public_body_boosts': {
        u"Ministry": 1.9,
        u"Council": 0.8
    }
})
```

4.4 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_CONFIG.update(
    dict(
        dryrun=False,
        dryrun_domain="testmail.example.com"
    )
)
```

This converts public body email addresses from

public-body@example.com

to

public-body+example.com@testmail.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.

4.5 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/testing 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"
```

4.6 Some more settings

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

```
SITE_NAME = 'FroIde'
SITE_URL = 'http://localhost:8000'
SITE_EMAIL = 'info@example.com'
```

More suggestions of settings you can change can be found in the *custom_settings.py.example* file that comes with froide.

4.7 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"
}
```

It's also recommended to protect the admin 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
```

Make sure that your frontend server transports the information that HTTPS is used to the web server.

CHAPTER

FIVE

THEMING FROIDE

If want to customize the look of your own Froide instance or add other pages to it, you can create a theme and install it in your Froide instance.

See the FragDenStaat.de Theme as an example.

5.1 Basics

A theme is normal Python Package and Django App. The app's templates, static files and urls will be found first be Froide and therefore override the normal files of Froide.

5.2 URLs

You can add custom URLs to your Froide instance by placing an *urls.py* file in the app. The url patterns will be hooked to the root of the Froide URLs and are the first to be considered for routing. An example might look like this:

```
from django.conf.urls import patterns, url
from django.http import HttpResponseRedirect

urlpatterns = patterns('fragdenstaat_de.views',
    url(r'^nordrhein-westfalen/', lambda request: HttpResponseRedirect('/nrw/'),
        name="jurisdiction-nrw-redirect")
)
```

This is simply a custom redirect for a jurisdiction URL but you can also hook up your own views.

5.3 Static files

You can override the serving of existing static files by placing a *static* folder in your theme app. If you put a file like *img/logo.png* in the *static* folder of your theme, Froide will serve the theme logo instead of the standard Froide logo. You can also override and add CSS and JavaScript files like this.

5.4 Templates

Most likely you want to change some parts of the site, but keep most of it the same. The Django template language allows you to extend base template and override blocks. However, if you override the base template you have to copy

over all blocks. To circumvent that Froide comes with the template tag *overextend*. It allows to override only the specific blocks in templates and keeps the other blocks the same.

An example for the *base.html* template could look like this:

```
{% overextends "base.html" %}
{% load i18n %}

{% block footer_description %}
<p>
  {% blocktrans with url=about_url %}Froide is a free and Open Source Project by <a href="http://www
</p>
{% endblock %}
```

This will only override the footer description of the *base.html* template.

Have a look at the Froide templates to find block you can override. If you need to override a specific part that is not enclosed in a block tag yet, open a pull request. More blocks are always welcome.

CHAPTER

SIX

FROIDE API

Froide has a RESTful API that allows structured access to the content inside your Froide instance.

The Froide API is available at */api/v1/* and the interactive Froide API documentation is available at */api/v1/docs/*.

There are additional search endpoints for Public Bodies and FOI Requests at */api/v1/publicbody/search/* and */api/v1/request/search/* respectively. Use *q* as the query parameter in a GET request.

GET requests do not need to be authenticated. POST, PUT and DELETE requests have to either carry a valid session cookie and a CSRF token or provide username (you find your username on your profile) and password via Basic Authentication.

CHAPTER

SEVEN

DEVELOPMENT

7.1 Run tests

Froide has a test suite. You have to install the test dependencies first:

```
pip install -r requirements-test.txt
```

The default test configuration requires a running elasticsearch instance. You either have to install elasticsearch or change the configuration to another search engine.

Then you can run the tests:

```
make test
```

This also does test coverage analysis. You can generate an HTML report with:

```
coverage html --omit="*/migrations/*"
```

You can then find the test HTML coverage report at *htmlcov/index.html*.

If you want to run tests for modules, classes or methods your can run them like this:

```
# Run everything
python manage.py test froide --settings=froide.settings_test
# Run only app tests
python manage.py test froide.foirequest --settings=froide.settings_test
# run only one test module in app
python manage.py test froide.foirequest.tests.test_admin.AdminActionTest --settings=froide.settings_t
# run only one method
python manage.py test froide.foirequest.tests.test_admin.AdminActionTest.test_approve --settings=froi
```

7.2 Build docs

The docs can be build with Sphinx but first you must install the theme. Excecute these commands from the top level of the froide directory:

```
git submodule init
git submodule update
```

Then change into the *docs* directory and type:

```
make html
```

The documentation will be build and you can find the resulting html in *docs/_build/html*.

CHAPTER

EIGHT

UPGRADE DESCRIPTIONS

8.1 Upgrading from 2.X to 3.X

The settings file has been restructured. All Froide related configuration (except email configuration) is now inside the *FROIDE_CONFIG* setting.

Django and other dependencies were upgraded to their latest versions.

8.2 Upgrading from 1.X to 2.X

Major Change was the move to Bootstrap theming. Themes have to be adapted.

CHAPTER

NINE

PRODUCTION SETUP

This is an example of a production setup, your personal flavor may vary.

- Nginx as a frontend server
- Supervisor as process manager and monitor
- Gunicorn as WSGI Server
- RabbitMQ as Background Queue Broker
- Solr as Search Engine

9.1 Protect admin site

Setup your front end server to serve the admin site behind basic authentication. Here is an example for Nginx:

```
location /custom-admin-url {
  auth_basic "Restricted";
  auth_basic_user_file  /var/www/froide/conf/htadminsitepasswd;
  proxy_pass http://127.0.0.1:29000;
  proxy_set_header Host $host;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
  proxy_set_header X-Forwarded-Protocol https;
}
```

Also note that we forward the HTTPS protocol. This sh

- Protect the admin by setting a Basic Authentication via Nginx for the URL
- Make the URL secret via the *SECRET_URLS* setting

9.2 Acces to Documents

Nginx is able to serve your uploads behind authentication/authorization. Activate the following settings:

```
# Use nginx to serve uploads authenticated
USE_X_ACCEL_REDIRECT = True
X_ACCEL_REDIRECT_PREFIX = '/protected'
```

Nginx will forward the request to Froide which will in turn check for authentication and authorization. If everything is good Froide replies to Nginx with an internal redirect and Nginx will then serve the file to the user.

A sample configuration looks like this:

```
location /protected {
  internal;
  alias /var/www/froide/public;
}
```

CHAPTER

TEN

INDICES AND TABLES

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