cleanship

Release 0.1

Jun 11, 2020

Contents:

Cleanship is a citizen participation (Bürgerbeteiligung) / complaints & suggestion management (Anliegenmanagement) / … online platform that allows you to submit issues about the public infrastructure. The local administration will then review your note and the solution process is public visible.

CHAPTER 1

Basics

It is the successor of *Klarschiff*, a platform where citizens report problems / ideas concerning public infrastructure. The local administration will then review your note and the solution process is public visible.

The code is Python3 using the Django 2 framework and bootstrap4 webfrontend toolkit.

Warning: Currently **alpha** , so expect that we will break your installation / data / modules / . . . !

Migrations will cause data lost!

Note: Currently we port only existing features, while avoiding old bottlenecks. New features will be introduced starting with version 0.3. New features will be introduced starting with version 0.3 (*IX*)

Note:

This version is a preview with a lot of limitations:

- frontend UI not polished
- read-only API
- some performance issues
- hardcoded settings

CHAPTER 2

Features

- issues with georeference
- groups to maintain / delegate issues
- staff frontend (office)
- admin backend
- API compatible with *CitySDK*
- import *Klarschiff* legacy data

../img/clean[illegible]ailexample.png

CHAPTER 3

Usage

- start `python3 manage.py runserver --settings cleanship.settings.local`
- enter `localhost:8000/admin` to maintain issues
- enter `localhost:8000/office` for staff backoffice
- enter `localhost:8000/citysdk` for REST API
- enter `localhost:8000/feed` for GeoRSS feed

3.1 Setup

On Linux you need to follow this steps to get a working instance

3.1.1 Setup postgres DBMS with geoextension

```
sudo apt install pgadmin3 postgresql postgresql-10-postgis-2.4  postgresql-10-postgis-
↪scripts
sudo -u postgres psql
```

```
CREATE USER cleanship WITH PASSWORD 'mysecretpass';
CREATE DATABASE cleanship OWNER cleanship;
ALTER ROLE cleanship CREATEDB SUPERUSER;  /*setting up test-dbs with GIS extension␣
↪requires high privileges)*/
```

You quit with ‘q’. Now work on specific ‘cleanship’ DB:

```
psql cleanship
```

```
CREATE EXTENSION postgis;
```

3.1.2 Setup python virtualenv

```
sudo apt install python3-dev libpq-dev binutils libproj-dev gdal-bin
mkvirtualenv -p /usr/bin/python3 cleanship
workon cleanship
```

3.1.3 Init codebase

```
git clone cleanship
pip install -R requirements/base.txt (dev.txt for contributing)
```

3.1.4 Configure instance

- `cp /cleanship/settings/example.py /cleanship/settings/local.py`
- Adapt your settings in `/cleanship/settings/local.py`
- Apply DB tables with `python3 manage.py migrate --settings cleanship.settings.local`
- Gather static assets to ./static `python3 manage.py collectstatic --settings cleanship.settings.local`
- Test startup with `python3 manage.py runserver --settings cleanship.settings.local`
- Create first admin user with `python3 manage.py createsuperuser --settings cleanship.settings.local`
- Create `/municipality_area.json` which contains the outer border as polygon in CRS:4326 (e.g. of Rostock
- Create `/eigentumsangaben.geojson` which contains disjunct polygones CRS:25833 with char field `eigentumsangabe` about landowners
- Perform single tests with e.g. `python3 manage.py test legacy/tests -v 2 --settings cleanship.settings.local`

3.1.5 Assign users to groups

Could be done via admin frontend or programmatically via python shell

```
from common.models import User, Group
myself = User.objects.get(username='test')
group = Group.objects.get(name='a group')')
group = Group.objects.get(name='a group')')
group.user_set.add(myself)
group.save()
```

CHAPTER 4

Klarschiff migration

You can transfer your existing issues from Klarschiff (tested v1.9) to cleanship including issues, categories, groups. We highly recommend a fresh cleanship setup to avoid troubles!

This migration is a more complex process and will take some time for tuning till you get the desired results. Please start with a cloned Klarschiff copy to proceed step by step:

- import categories - create the ideas / problems / tipps hierachy
- import issues - create issue with all details and creates user groups
- export old user details and manually merge them
- import users - create unified user that are referenced
- import comments
- import feedback

The order of the steps is essential, as e.g. comments refer users, issues refer categories, ...

When you have a working mapping and reproducible import steps, you can run all steps in the right order, using `python3 manage.py import --settings cleanship.settings.local`.

4.1 Preparations

The following instructions are tested to transfer your Klarschiff content:

- export old data as CSV via this shell-script at your current Klarschiff DB server

```
export PGPASSWORD="mypass"
psql -h localhost -d klarschiff -U admin -Atc "select tablename from pg_tables" |\
    while read TBL;
        do if [[ $TBL == *"klarschiff_"* ]]; then psql -h localhost -d klarschiff -U␣
↪admin -c "COPY $TBL TO STDOUT WITH (FORMAT CSV, HEADER);" > $TBL.csv
        fi done
```

- copy all full size photos to /media directory: `cp /srv/www/klarschiff/static/*_gross_*.jpg ./media`

4.2 Import Users

- create list of legacy user details: `python3 manage.py exportLegacyUsers --settings cleanship.settings.local`
- Use the resulting users.txt, which contains of 3 separated blocks (full names, emails, usernames) of all of your Klarschiff users, to manually create a new mapping file called users.csv. It will be used to create the listed users in the next step and needs to contain the following fields:

  You need to fill the lines using the information from the txt listing and maybe your Klarschiff administration interface. To merge a user from multiple old names, just add dublicated lines. This mapping is nessesary, to deal with Klarschiff different legacy user references when importing users / comments / feedback / edit history. The goal is to normalize old user datasets (encoding, old migration artifacts, ..) and match the new user-id (lowercase username) against your LDAP user-ids. You can also re-arrange users to the groups.
- Create users using this mapping file using `python3 manage.py import --users --settings cleanship.settings.local`

4.3 Import Categories

Import all categories: `python3 manage.py import --categories --settings cleanship.settings.local`

4.4 Import Issues

Note:

The import skips some checks to improve performance:

- is in boundary polygon
- updating location description
- updating landowner

- Start full import via `python3 manage.py import --issues --settings cleanship.settings.local`
- Import will take only a few minutes

4.5 Import Comments

Import all comments `python3 manage.py import --comments --settings cleanship.settings.local`.

With the first runs, you will noticed some warnings about non-existing users, which we can't find in the old_fullname mapping. You will need to find / add them the user mapping file, and run the user import again!

4.6 Import Feedback

Import all comments `python3 manage.py import --feedback --settings cleanship.settings.local`.

With the first runs, you will noticed some warnings about non-existing users, which we can't find in the email mapping. You will need to find / add them the user mapping file, and run the user import again! There are also Feedback with multiple, or no recipent, which don't need further attention.

CHAPTER 5

Concepts

Cleanship focus to be a enterprise-grade selfhosted solution for the public administrations. so it integrates in your existing IT and scales for huge amount of users and notes. You can receive notes with geolocation easily and maintain this issues to find a solution to this note step-by-step.

- LDAP support
- multilanguage
- scale with huge amount of issues (>40k tested so far)
- history and logging
- open API following *CitySDK* protocoll

To get a basic understanding of the internals, you might have a look at the base object definitions:

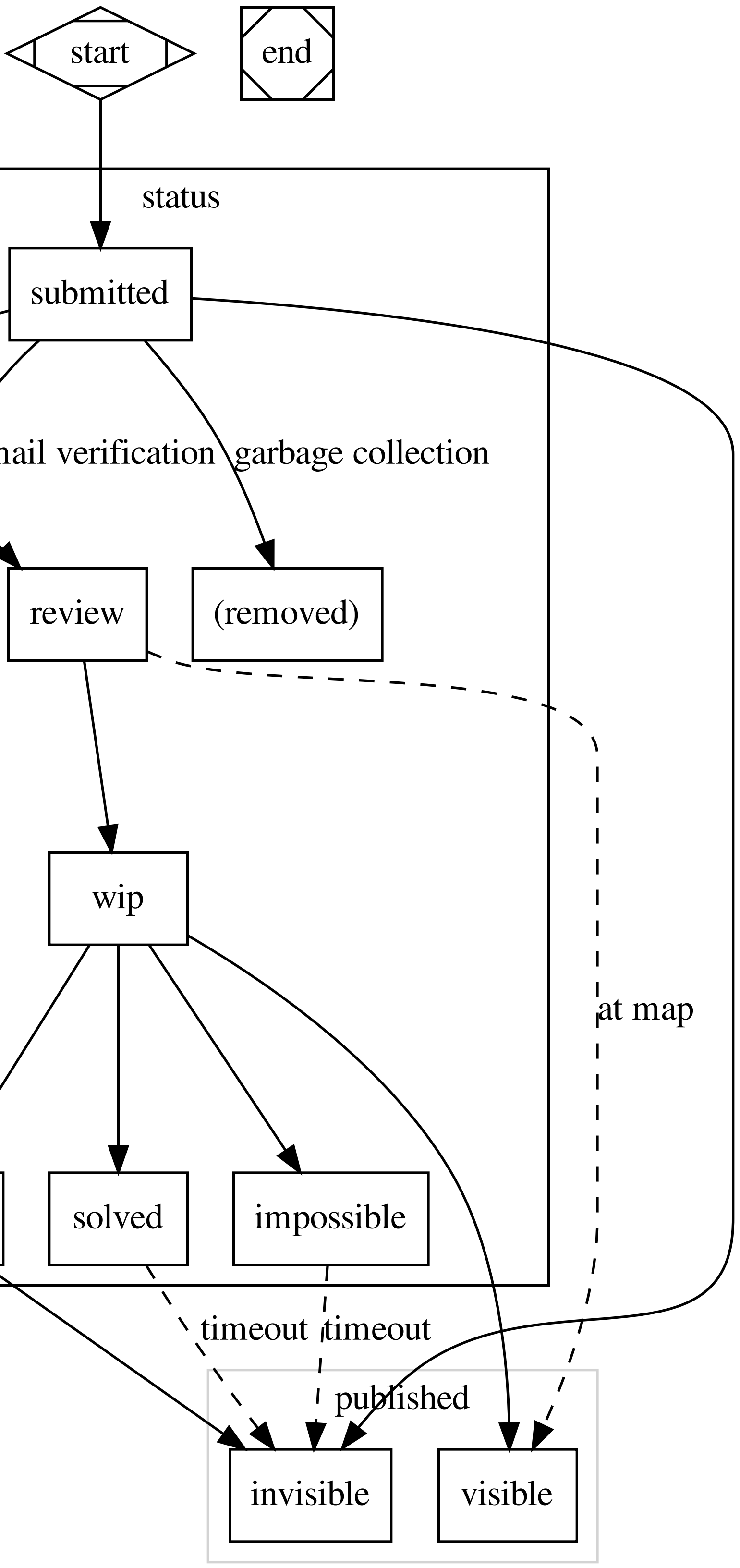
- **issue** - problem / idea / tipp for a location, submitted by external or interal authors. e.g. a pothole within a specific street. Focus is the reported damage, not the solution itself. It's allways assigned to a group of a organisation
- **category** - a 3 level categorisation by type (problem / idea / tipp), main-category and sub-category e.g. *problem - waste - bulky refuse*
- **role** - a overall qualification for a user e.g. admin, editors, field service
- **group** - a organisation unit of multiple users e.g. *civil engineering office*
- **user** - a member of a organsiation

The django project is splitted in different **apps** focussing on single aspects:

- **common** - general aspects esp. shared models, admin frontend
- **legacy** - compatibility features to migrate from old predecessor project *Klarschiff*
- **office** - internal frontend for staff
- **citysdk** - *CitySDK* API for public frontend / 3rd party apps & platforms

5.1 Status

A issue has a status which indicates it’s current progress and which transition can trigger various actions in detail.


start
end
status
submitted
email verification
garbage collection
review
(removed)
wip
at map
dublicate
solved
impossible
timeout
timeout
published
invisible
visible

- submitted - User submitted issue, but didn’t verified his email yet. Issue will be removed, if user don’t open confirmation link.
- review - User verified his email, but no internal group assigned and no person did a review of the issue content yet. Issue displayed on the map, but details and photo stay hidden.
- work in progress (wip) - A internal group is assigned and working on the issue. Details and photo become public.
- solved - Final state, the core issue could be solved. An explaination is in status text.
- impossible - Final state, the core issue couldn’t be solved. An explaination is in status text.

CHAPTER 6

Development

Please see readme.md and CONTRIBUTING.md!

CHAPTER 7

Glossary

CitySDK A REST like API which CitySDK participation component was implemented at Klarschiff. It is a updated version of the Open311 protocol, by exteding some core concepts. Klarschiff also added even more features, are completely covered here.

Klarschiff Is the previous (legacy) software suite for civic participation management. It was splitted in a public and internal frontend, both for mobile and desktop users. The database was managed by a backend component, which offered a *CitySDK* API to the frontends. See Wikipedia (Klarschiff) Cleanship replaces everything below the public frontends.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Index