terra-backend-crud

Release 0.3.17

Oct 17, 2019

Contents

Terralego backend app

CHAPTER 1

Installation

1.1 Requirements

Must be installed, for the package to work:

- terracommon.terra
- template_model
- template_engines

1.2 With pip

From Pypi:

```
pip install xxxxxxxxxx-xxxxxxxxxxxx
```

From Github:

```
pip install -e https://github.com/Terralego/terra.backend.crud.git@master#egg=django-
↪template-engines
```

1.3 With git

```
git clone https://github.com/Terralego/terra.backend.crud.git
cd terra.backend.crud
python setup.py install
```

CHAPTER 2

Configuration

In your project :

- settings

```
INSTALLED_APPS = [
    ...
    # apps required by CRUD
    'geostore',  # store geographic data
    'template_model',  # store template in model
    'template_engines',  # generate odt and docx templates
    'rest_framework',  # if you want to try api HTML interface
    'django_json_widget',  # if you want to use django admin
    'reversion',  # used to store every change on data (run ./manage.py␣
↪createinitialrevisions first)
    # CRUD app
    'terra_geocrud',
    ...
]
...
TEMPLATES = [
    ...
    # if you want to render odt templates
    {'BACKEND': 'template_engines.backends.odt.OdtEngine'},
    # if you want to render docx templates
    {'BACKEND': 'template_engines.backends.docx.DocxEngine'},
]
```

- urls

```
urlpatterns = [
    ...
    # some urls in geostore are required by geocrud
    path('api/geostore/', include('geostore.urls')),
    path('api/crud/', include('terra_geocrud.urls')),
```

(continues on next page)

(continued from previous page)

```
    ...
]
```

You can customize default url and namespace by including terra_geocrud.views directly

Run migrations

```
./manage.py migrate
```

- ADMIN :

you can disable and / or customize admin

- SETTINGS :

Waiting for settings definition directly in models.

Settings should be overrided with TERRA_GEOCRUD settings in your project settings file:

```
...
TERRA_GEOCRUD = {
    # default value for map extent. API serialize this for layer extent if there is␣
↪no features in it (as default)
    'EXTENT': [-90.0, -180.0, 90.0, 180.0],
    # default storage for file stored in json properties. It is recommended to␣
↪configure a private web storage in your project (as S3Storage -> see django-
↪storages)
    'DATA_FILE_STORAGE_CLASS': 'django.core.files.storage.FileSystemStorage',
    # default mapbox style provided by api if no custom style defined in crud view
    'STYLES': {
        'line': {
            'type': 'line',
            'paint': {
                'line-color': '#000',
                'line-width': 3
            }
        },
        'point': {
            'type': 'circle',
            'paint': {
                'circle-color': '#000',
                'circle-radius': 8
            }
        },
        'polygon': {
            'type': 'fill',
            'paint': {
                'fill-color': '#000'
            }
        },
    }
}
...
```

CHAPTER 3

Example of use

- TERRA_GEOCRUD provide its own settings url to terralego apps via

example:

```
/api/settings

{
  'CRUD': '/api/crud/settings/'
}
```

Settings provide default config values, and formatted list of crud group and views, ready to be displayed in frontend left menu.

- There are 4 endpoint in GEOCRUD API:

:: settings/ -> get ordered menu with views classified by group or not, and basic map settings groups/ -> manage groups of CRUD views views/ -> manage CRUD views (a view creation create its associated layer) template/<template_pk>/render/<pk>/ -> fill a template with a feature

- A command is available to create default views for each existing layer

```
./manage.py create_default_crud_views
```

- START GUIDE
- First, you need to create crud views for your geostore layers with the command or the admin.
- These views can be grouped, and will be listed by the frontend api
- Then, you can customize default layer-schema by providing your own property groups, which will groups properties as json schema nested objects.

ADMIN

- Some classes are provided to help you to manage Crud views / groups / layers and feature through django admin.
- You need to register your wanted ModelAdmin in your project