
SODAR Core Documentation

Release 0.6.1

Mikko Nieminen

Jun 21, 2019

Contents:

1	SODAR and SODAR Core	3
1.1	Getting Started	3
1.2	Projectroles App	5
1.3	Adminalerts App	40
1.4	Bgjobs App	41
1.5	Filesfolders App	42
1.6	Userprofile App	45
1.7	Siteinfo App	46
1.8	Sodarcache App	47
1.9	Taskflow Backend	53
1.10	Timeline App	55
1.11	Development	63
1.12	Breaking Changes	82
2	Indices and tables	89
	Python Module Index	91
	Index	93

This documentation provides instructions for integration, usage and development of reusable SODAR Core apps for projects built on the Django web server.

SODAR and SODAR Core

SODAR (System for Omics Data Access and Retrieval) is a specialized system for managing data in omics research projects.

This repository contains reusable and non-domain-specific apps making up the core of the SODAR system. These apps can be used for any Django application which wants to make use of the following features:

- Project-based user access control
- Dynamic app content management
- Advanced project activity logging
- Small file uploading and browsing
- Managing server-side background jobs
- Caching and aggregation of data from external services
- Tracking site information and statistics

Basics of Django site setup and instructions for third party packages used are considered out of scope for this documentation. Please refer instead to official documentation of Django and/or the packages in question.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

NOTE: To view this document in the rendered form during development, run `make html` in the `docs` directory of the repository. You can find the rendered HTML in `docs/build`. You will have to install system and Python dependencies, including ones in `requirements/local.txt` for this to work. See [SODAR Core Development](#).

1.1 Getting Started

Basic concepts of SODAR Core apps are detailed in this document.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

1.1.1 Repository Contents

The following Django apps will be installed when installing the `django-sodar-core` package:

- **projectroles**: Base app for project access management and dynamic app content management. All other apps require the integration of projectroles.
- **adminalerts**: Site app for displaying site-wide messages to all users.
- **bgjobs**: Project app for managing background jobs.
- **siteinfo**: Site app for displaying site information and statistics for administrators.
- **sodarcache**: Generic caching and aggregation of data referring to external services.
- **taskflowbackend**: Backend app providing an API for the optional `sodar_taskflow` transaction service.
- **timeline**: Project app for logging and viewing project-related activity.
- **userprofile**: Site app for viewing user profiles.

The following packages are included in the repository for development and as examples:

- **config**: Example Django site configuration
- **docs**: Usage and development documentation
- **example_backend_app**: Example SODAR Core compatible backend app
- **example_project_app**: Example SODAR Core compatible project app
- **example_site**: Example/development Django site
- **example_site_app**: Example SODAR Core compatible site-wide app
- **requirements**: Requirements for SODAR Core and development
- **utility**: Setup scripts for development

1.1.2 Requirements

Major requirements for integrating projectroles and other SODAR Core apps into your Django site and/or participating in development are listed below. For a complete requirement list, see the `requirements` and `utility` directories in the repository.

- Ubuntu 16.04 Xenial (**NOTE**: Older releases no longer supported)
- Library requirements (see the `utility` directory and/or your own Django project)
- Python 3.6+ (**NOTE**: Python 3.5 no longer supported)
- Django 1.11.20+ (**NOTE**: 2.x not currently supported)
- PostgreSQL 9.6+ and psycopg2-binary
- Bootstrap 4.3.1
- JQuery 3.3.1
- Shepherd 1.8.1 with Tether 1.4.4
- Clipboard.js 2.0.0
- DataTables 1.10.18 with JQuery UI, FixedColumns, FixedHeader, Buttons, KeyTables

For more detailed instructions on what to install for local development, see [SODAR Core Development](#).

1.1.3 Next Steps

To proceed with using the SODAR Core framework in your Django site, you must first install and integrate the `projectroles` app. See the [projectroles app documentation](#) for instructions.

Once `projectroles` has been integrated into your site, you may proceed to install other apps as needed.

1.2 Projectroles App

The `projectroles` app is the base app for building a SODAR Core based Django site. It provides a framework for project access management, dynamic content including with django-plugins, models and tools for SODAR-compatible apps plus a default template and CSS layout.

Other Django apps which intend to use aforementioned functionalities depend on `projectroles`. While inclusion of other SODAR Core apps can be optional, having `projectroles` installed is **mandatory** for working with the SODAR project and app structure.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

1.2.1 Projectroles Basics

The basic concepts and functionalities of the `projectroles` app are detailed in this document.

Projects

The `projectroles` app groups project-specific data, user access roles and other features into **projects** and **categories**. These can be nested in a tree structure with the `category` type working as a container for sub-projects with no project content of its own.

User Roles in Projects

User access to projects is granted by per-project assigning of roles. In each project, a user can have one role at a time. New types of roles can be defined by extending the default model’s database table.

The default setup of role types used in SODAR sites:

- **project owner**
 - Full read/write access to project data and roles
 - Can create sub-projects under owned categories
 - One per project
 - Must be specified upon project creation
- **project delegate**
 - Full read/write access to project data
 - Can modify roles except for owner and delegate
 - One per project (as default, the limit can be increased in site settings)
 - Assigned by owner

- **project contributor**
 - Can read and write project data
 - Can modify and delete own data
- **project guest**
 - Read only access to project data

Note: Django **superuser** status overrides project role access.

The projectroles app provides the following features for managing user roles in projects:

- Adding/modifying/removing site users as project members
- Inviting people not yet using the site by email
- Importing members from other projects (**NOTE:** disabled pending update)
- Automated emailing of users regarding role changes
- Mirroring user roles to/from an external projectroles-enabled site

Note: Currently, only superusers can assign owner roles for top-level categories.

Remote Project Sync

SODAR Core allows optionally reading and synchronizing project metadata between multiple SODAR-based Django sites. A superuser is able to set desired levels of remote access for specific sites on a per-project basis.

A SODAR site must be set in either **source** or **target** mode.

- **Source** site is one expecting to (potentially) serve project metadata to an arbitrary number of other SODAR sites.
- **Target** site can be linked with exactly one source site, from which it can retrieve project metadata. Creation of local projects can be enabled or disabled according to local configuration.

Among the data which can be synchronized:

- General project information such as title, description and readme
- Project category structure
- User roles in projects
- User accounts for LDAP/AD users (required for the previous step)

Rule System

Projectroles uses the [django-rules](#) package to manage permissions for accessing data, apps and functionalities within projects based on the user role. Predicates for project roles are provided by the projectroles app and can be used and extended for developing rules for your other project-specific Django apps.

Plugins

Projectroles provides a plugin framework to enable integrating apps and content dynamically to a projectroles-enabled Django site. Types of plugins currently included:

- **Project apps:** Apps tied to specific projects, making use of project roles, rules and other projectroles functionalities.
- **Site apps:** Site-wide Django apps which are not project-specific
- **Backend apps:** Backend apps without GUI entry points or (usually) views, imported and used dynamically by other SODAR-based apps for e.g. connectivity to external resources.

Existing apps can be modified to conform to the plugin structure by implementing certain variables, functions, views and templates within the app. For more details, see the app development documents.

Other Features

Other features in the projectroles app:

- **App settings:** Setting values for project or user specific variables, which can be defined in project app plugins
- **Project starring:** Ability for users to star projects as their favourites
- **Project search:** Functionality for searching data within projects using functions implemented in project app plugins
- **Tour help:** Inline help for pages
- **Project readme:** README document for each project with Markdown support
- **Custom user model:** Additions to the standard Django user model
- **Multi-Domain LDAP/AD support:** Support for LDAP/AD users from multiple domains
- **SODAR Taskflow and SODAR Timeline integration:** Included but disabled unless backend apps for Taskflow and Timeline are integrated in the Django site

Templates and Styles

Projectroles provides views and templates for all GUI-related functionalities described above. The templates utilize the plugin framework to provide content under projects dynamically. The project also provides default CSS stylings, base templates and a base layout which can be used or adapted as needed. See the usage and app development documentation for more details.

1.2.2 Projectroles Integration

This document provides instructions and guidelines for integrating projectroles and other SODAR Core apps into your Django site.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Installation on a New Site

If you want to set up a new site for integrating projectroles, see the recommended options in this section.

SODAR Django Site Template (Recommended)

When setting up a new SODAR Core based site, it is strongly recommended to use [sodar_django_site](#) as a template. The repository contains a minimal Django site pre-configured with projectroles and other SODAR Core apps. The master branch of this project always integrates the latest stable release of SODAR Core and projectroles.

To set up your site with this template, clone the repository and follow the installation instructions in the README.rst file.

To modify default SODAR Core and projectroles settings, see the [Projectroles Django Settings](#) document.

Once you have your site set up, you can look into [customization tips](#) and start *developing your SODAR Core compatible apps*.

Cookiecutter-Django

If the SODAR Django site template does not suit your needs, it is also possible to set up your site using [cookiecutter-django](#). In this case, follow the instructions in the following section as if you were integrating SODAR Core to an existing Django site.

Warning: Currently, SODAR Core only supports Django 1.11.x, while the latest versions of cookiecutter-django set up Django 2.0.x by default. It is strongly recommended to use Django 1.11 LTS for time being. Compatibility with 2.0 and upwards is not guaranteed! Integration into the last official [1.11 release](#) of cookiecutter-django has been tested and verified to be working.

Note: The latest cookiecutter-django 1.11 release has dependencies which are already out of date. Please update them to match the requirements of the django-sodar-core package.

Note: For any other issues regarding the cookiecutter-django setup, see the cookiecutter-django documentation.

Installation on an Existing Site

Instructions for setting up projectroles and SODAR Core on an existing Django site or a fresh site generated with cookiecutter-django are detailed in this chapter.

Warning: In order to successfully set up projectroles, you are expected to **follow all the instructions here in the order they are presented**. Please note that leaving out steps may result in a non-working Django site! Attempting to run the site before implementing all of the steps may (and probably will) result in errors.

Warning: The rest of this section assumes that your Django project has been set up using a [1.11 release of cookiecutter-django](#). Otherwise details such as directory structures and settings variables may differ.

First, add the `django-plugins` and `django-sodar-core` package requirements into your `requirements/base.txt` file. Make sure you are pointing to the desired release tag.

```
-e git://github.com/mikkonie/django-plugins.
→git@1bc07181e6ab68b0f9ed3a00382eb1f6519e1009#egg=django-plugins
-e git://github.com/bihealth/sodar_core.git@v0.4.2#egg=django-sodar-core
```

Install the requirements for development:

```
$ pip install -r requirements/local.txt
```

If any version conflicts arise between django-sodar-core and your existing site, you will have to resolve them before continuing.

Hint: You can always refer to either the `sodar_django_site` repository or `example_site` in the SODAR Core repository for a working example of a Cookiecutter-based Django site integrating SODAR Core. However, note that some aspects of the site configuration may vary depending on the cookiecutter-django version used on your site.

Django Settings

Next you need to modify your default Django settings file, usually located in `config/settings/base.py`. For sites created with an older cookiecutter-django version the file name may also be `common.py`. Naturally, you should make sure no settings in other configuration files conflict with ones set here.

For values retrieved from environment variables, make sure to configure your env accordingly. For development and testing, using `READ_DOT_ENV_FILE` is recommended.

Required and optional Django settings are described in the [Projectroles Django Settings](#) document.

User Configuration

In order for SODAR Core apps to work on your Django site, you need to extend the default user model.

Extending the User Model

In a cookiecutter-django project, an extended user model should already exist in `{SITE_NAME}/users/models.py`. The abstract model provided by the projectroles app provides the same model with critical additions, most notably the `sodar_uuid` field used as an unique identifier for SODAR objects including users.

If you have not added any of your own modifications to the model, you can simply **replace** the existing model extension with the following code:

```
from projectroles.models import SODARUser

class User(SODARUser):
    pass
```

If you need to add your own extra fields or functions (or have existing ones already), you can add them in this model.

After updating the user model, create and run database migrations.

```
$ ./manage.py makemigrations
$ ./manage.py migrate
```

Note: You probably will need to edit the default unit tests under `{SITE_NAME}/users/tests/` for them to work after making these changes. See `example_site.users.tests` in this repository for an example.

Populating UUIDs for Existing Users

When integrating projectroles into an existing site with existing users, the `sodar_uuid` field needs to be populated. See [instructions in Django documentation](#) on how to create the required migrations.

Synchronizing User Groups for Existing Users

To set up user groups for existing users, run the `syncgroups` management command.

```
$ ./manage.py syncgroups
```

User Profile Site App

The `userprofile` site app is installed with SODAR Core. It adds a user profile page in the user dropdown. Use of the app is not mandatory but recommended, unless you are already using some other user profile app. See the [userprofile app documentation](#) for instructions.

Add Login Template

You should add a login template to `{SITE_NAME}/templates/users/login.html`. If you're OK with using the projectroles login template, the file can consist of the following line:

```
{% extends 'projectroles/login.html' %}
```

If you intend to use projectroles templates for user management, you can delete other existing files within the directory.

URL Configuration

In the Django URL configuration file, usually found in `config/urls.py`, add the following lines under `urlpatterns` to include projectroles URLs in your site.

```
urlpatterns = [
    # ...
    url(r'^api/auth/', include('knox.urls')),
    url(r'^project/', include('projectroles.urls')),
]
```

If you intend to use projectroles views and templates as the basis of your site layout and navigation (which is recommended), also make sure to set the site's home view accordingly:

```
from projectroles.views import HomeView

urlpatterns = [
    # ...
```

(continues on next page)

(continued from previous page)

```
url(r'^$', HomeView.as_view(), name='home'),
]
```

Finally, make sure your login and logout links are correctly linked. You can remove any default allauth URLs if you're not using it.

```
from django.contrib.auth import views as auth_views

urlpatterns = [
    # ...
    url(r'^login/$', auth_views.LoginView.as_view(
        template_name='users/login.html'), name='login'),
    url(r'^logout/$', auth_views.logout_then_login, name='logout'),
]
```

Base Template for Your Django Site

In order to make use of Projectroles views and templates, you should set the base template of your site accordingly in `{SITE_NAME}/templates/base.html`.

For a supported example, see `projectroles/base_site.html`. It is strongly recommended to use this as the base template for your site, either by extending it or copying the content into `{SITE_NAME}/templates/base.html` and modifying it to suit your needs.

If you do not need to make any modifications, the most simple way is to replace the content of the `{SITE_NAME}/templates/base.html` file with the following line:

```
{% extends 'projectroles/base_site.html' %}
```

Note: CSS and Javascript includes in `site_base.html` are **mandatory** for Projectroles-based views and functionalities.

Note: The container structure defined in the example `base.html`, along with including the `{STATIC}/projectroles/css/projectroles.css` are **mandatory** for Projectroles-based views to work without modifications.

Site Error Templates

The projectroles app contains default error templates to use on your site. These are located in the `projectroles/error/` template directory. You can use them by entering `{% extends 'projectroles/error/*.html' %}` in the corresponding files found in the `{SITE_NAME}/templates/` directory. You have the options of extending or replacing content on the templates, or simply implementing your own.

All Done!

After following all the instructions above, you should have a working Django site with Projectroles access control and support for SODAR app. To test the site locally execute the supplied shortcut script:

```
$ ./run.sh
```

Or, run the standard Django runserver command:

```
$ ./manage.py runserver
```

You can now browse your site locally at `http://127.0.0.1:8000`. You are expected to log in to view the site. Use e.g. the superuser account you created when setting up your cookiecutter-django site.

You can now continue on to create apps or modify your existing apps to be compatible with the SODAR Core framework. See the [development section](#) for app development guides. Also see the [customization documentation](#) for tips for modifying the default appearance of SODAR Core.

1.2.3 Projectroles Django Settings

This document describes the Django settings for the `projectroles` app, which also control the configuration of other apps in a SODAR Core based site.

These settings are usually found in `config/settings/*.py`, with `config/settings/base.py` being the default configuration other files may override or extend.

If your site is based on `sodar_django_site`, mandatory settings are already set to their default values. In that case, you only need to modify or customize them where applicable.

If you are integrating `django-sodar-core` with an existing Django site or building your site from scratch without the recommended template, make sure to add all mandatory settings into your project.

For values retrieved from environment variables, make sure to configure your env accordingly. For development and testing, using `DJANGO_READ_DOT_ENV_FILE` is recommended.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Site Package and Paths

The site package and path configuration should be found at the beginning of the default configuration file. Substitute `{SITE_NAME}` with the name of your site package.

```
import environ
SITE_PACKAGE = '{SITE_NAME}'
ROOT_DIR = environ.Path(__file__) - 3
APPS_DIR = ROOT_DIR.path(SITE_PACKAGE)
```

Apps

Apps installed from `django-sodar-core` are placed in `THIRD_PARTY_APPS`. The following apps need to be included in the list in order for SODAR Core to work:

```
THIRD_PARTY_APPS = [
    # ...
    'crispy_forms',
    'rules.apps.AutodiscoverRulesConfig',
    'djangoplugins',
    'pagedown',
```

(continues on next page)

(continued from previous page)

```
'markupfield',
'rest_framework',
'knox',
'projectroles.apps.ProjectrolesConfig',
'dal',
'dal_select2',
]
```

Database

Under DATABASES, the setting below is recommended:

```
DATABASES['default']['ATOMIC_REQUESTS'] = False
```

Note: If this conflicts with your existing set up, you can modify the code in your other apps to use e.g. `@transaction.atomic`.

Note: This setting mostly is used for the `sodar_taskflow` transactions supported by `projectroles` but not commonly used, so having this setting as `True` *may* cause no issues. However, it is not officially supported at this time.

Templates

Under `TEMPLATES['OPTIONS']['context_processors']`, add the `projectroles` URLs processor:

```
'projectroles.context_processors.urls_processor',
```

Email

Under `EMAIL_CONFIGURATION` or `EMAIL`, configure email settings:

```
EMAIL_SENDER = env('EMAIL_SENDER', default='noreply@example.com')
EMAIL_SUBJECT_PREFIX = env('EMAIL_SUBJECT_PREFIX', default='')
```

Authentication

`AUTHENTICATION_BACKENDS` should contain the following backend classes:

```
AUTHENTICATION_BACKENDS = [
    'rules.permissions.ObjectPermissionBackend',
    'django.contrib.auth.backends.ModelBackend',
]
```

Note: The default setup by `cookiecutter-django` adds the `allauth` package. This can be left out of the project if not needed, as it mostly provides adapters for e.g. social media account logins. If removing `allauth`, you can also remove unused settings variables starting with `ACCOUNT_*`.

The following settings remain in your auth configuration:

```
AUTH_USER_MODEL = 'users.User'
LOGIN_REDIRECT_URL = 'home'
LOGIN_URL = 'login'
```

Django REST Framework

To enable `django-rest-framework` API views and `knox` authentication, these values should be added under `DEFAULT_AUTHENTICATION_CLASSES`:

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework.authentication.BasicAuthentication',
        'rest_framework.authentication.SessionAuthentication',
        'knox.auth.TokenAuthentication',
    ),
}
```

General Site Settings

For display in `projectroles` based templates, customize related variables to describe your site. `SITE_INSTANCE_TITLE` may be used to e.g. differentiate between site versions used for deployment or staging, use in different organizations, etc.

```
SITE_TITLE = 'Name of Your Project'
SITE_SUBTITLE = env.str('SITE_SUBTITLE', 'Beta')
SITE_INSTANCE_TITLE = env.str('SITE_INSTANCE_TITLE', 'Deployment Instance Name')
```

Projectroles Settings

Mandatory `projectroles` app settings are explained below:

- `PROJECTROLES_SITE_MODE`: Site mode for remote project metadata synchronization, either `SOURCE` (allow others to read local projects) or `TARGET` (read projects from another site)
- `PROJECTROLES_TARGET_CREATE`: Whether or not local projects can be created if site is in `TARGET` mode. If your site is in `SOURCE` mode, this setting has no effect.
- `PROJECTROLES_INVITE_EXPIRY_DAYS`: Days until project email invites expire (int)
- `PROJECTROLES_SEND_EMAIL`: Enable/disable email sending (bool)
- `PROJECTROLES_ENABLE_SEARCH`: Whether you want to enable SODAR search on your site (boolean)
- `PROJECTROLES_DEFAULT_ADMIN`: User name of the default superuser account used in e.g. replacing an unavailable user or performing backend admin commands (string)

Example:

```
# Projectroles app settings
PROJECTROLES_SITE_MODE = env.str('PROJECTROLES_SITE_MODE', 'TARGET')
PROJECTROLES_TARGET_CREATE = env.bool('PROJECTROLES_TARGET_CREATE', True)
PROJECTROLES_INVITE_EXPIRY_DAYS = env.int('PROJECTROLES_INVITE_EXPIRY_DAYS', 14)
PROJECTROLES_SEND_EMAIL = env.bool('PROJECTROLES_SEND_EMAIL', False)
```

(continues on next page)

(continued from previous page)

```
PROJECTROLES_ENABLE_SEARCH = True
PROJECTROLES_DEFAULT_ADMIN = env.str('PROJECTROLES_DEFAULT_ADMIN', 'admin')
```

Optional Projectroles Settings

The following projectroles settings are **optional**:

- **PROJECTROLES_SECRET_LENGTH**: Character length of secret token used in projectroles (int)
- **PROJECTROLES_SEARCH_PAGINATION**: Amount of search results per each app to display on one page (int)
- **PROJECTROLES_HELP_HIGHLIGHT_DAYS**: Days for highlighting tour help for new users (int)
- **PROJECTROLES_DISABLE_CATEGORIES**: If set True, disable categories and only allow a list of projects on the root level (boolean) (see note)
- **PROJECTROLES_HIDE_APP_LINKS**: Apps hidden from the project sidebar and dropdown menus for non-superusers. The app views and URLs are still accessible. The names should correspond to the `name` property in each project app's plugin (list)
- **PROJECTROLES_DELEGATE_LIMIT**: The number of delegate roles allowed per project. The amount is limited to 1 per project if not set, unlimited if set to 0. Will be ignored for remote projects synchronized from a source site (int)
- **PROJECTROLES_BROWSER_WARNING**: If true, display a warning to users using Internet Explorer (bool)
- **PROJECTROLES_ALLOW_LOCAL_USERS**: If true, roles for local non-LDAP users can be synchronized from a source during remote project sync if they exist on the target site. Similarly, local users will be selectable in member dropdowns when selecting users (bool)

Example:

```
# Projectroles app settings
# ...
PROJECTROLES_SECRET_LENGTH = 32
PROJECTROLES_SEARCH_PAGINATION = 5
PROJECTROLES_HELP_HIGHLIGHT_DAYS = 7
PROJECTROLES_DISABLE_CATEGORIES = True
PROJECTROLES_HIDE_APP_LINKS = ['filesfolders']
PROJECTROLES_DELEGATE_LIMIT = 1
PROJECTROLES_BROWSER_WARNING = True
PROJECTROLES_ALLOW_LOCAL_USERS = True
```

Warning: Regarding **PROJECTROLES_DISABLE_CATEGORIES**: In the current SODAR core version remote site access and remote project synchronization are disabled if this option is used! Use only if a simple project list is specifically required in your site.

Warning: Regarding **PROJECTROLES_ALLOW_LOCAL_USERS**: Please note that this will allow synchronizing project roles to local non-LDAP users based on their **user name**. You should personally ensure that the users in question are authorized for these roles. Furthermore, only roles for **existing** local users will be synchronized. New local users will have to be added manually through the Django admin or shell on the target site.

Backend App Settings

The `ENABLED_BACKEND_PLUGINS` settings lists backend plugins implemented using `BackendPluginPoint` which are enabled in the configuration. For more information see [Backend App Development](#).

```
ENABLED_BACKEND_PLUGINS = env.list('ENABLED_BACKEND_PLUGINS', None, [])
```

SODAR API Settings (Optional)

There are also settings for providing and extending the general SODAR API, which is currently in development.

The API uses accept header versioning. The `SODAR_API_MEDIA_TYPE` setting is by default set to the SODAR Core API media type, but should preferably be changed to your organization and API identification if API views are modified or introduced. The `SODAR_API_DEFAULT_HOST` setting should post to the externally visible host of your server and be configured in your environment settings.

These settings are **optional**. Default values will be used if they are unset.

Example:

```
SODAR_API_DEFAULT_VERSION = '0.1'
SODAR_API_ACCEPTED_VERSIONS = [SODAR_API_DEFAULT_VERSION]
SODAR_API_MEDIA_TYPE = 'application/vnd.bihealth.sodar-core+json' # Change this
SODAR_API_DEFAULT_HOST = SODAR_API_DEFAULT_HOST = env.url('SODAR_API_DEFAULT_HOST',
↪ 'http://0.0.0.0:8000')
```

LDAP/AD Configuration (Optional)

If you want to utilize LDAP/AD user logins as configured by `projectroles`, you can add the following configuration. Make sure to also add the related env variables to your configuration.

This part of the setup is **optional**.

Note: In order to support LDAP, make sure you have installed the dependencies from `utility/install_ldap_dependencies.sh` and `requirements/ldap.txt`! For more information see [SODAR Core Development](#).

Note: If only using one LDAP/AD server, you can leave the “secondary LDAP server” values unset.

```
ENABLE_LDAP = env.bool('ENABLE_LDAP', False)
ENABLE_LDAP_SECONDARY = env.bool('ENABLE_LDAP_SECONDARY', False)

if ENABLE_LDAP:
    import itertools
    import ldap
    from django_auth_ldap.config import LDAPSearch

    # Default values
    LDAP_DEFAULT_CONN_OPTIONS = {ldap.OPT_REFERRALS: 0}
    LDAP_DEFAULT_FILTERSTR = '(sAMAccountName=%(user)s)'
    LDAP_DEFAULT_ATTR_MAP = {
        'first_name': 'givenName',
```

(continues on next page)

(continued from previous page)

```

        'last_name': 'sn',
        'email': 'mail',
    }

    # Primary LDAP server
    AUTH_LDAP_SERVER_URI = env.str('AUTH_LDAP_SERVER_URI', None)
    AUTH_LDAP_BIND_DN = env.str('AUTH_LDAP_BIND_DN', None)
    AUTH_LDAP_BIND_PASSWORD = env.str('AUTH_LDAP_BIND_PASSWORD', None)
    AUTH_LDAP_CONNECTION_OPTIONS = LDAP_DEFAULT_CONN_OPTIONS

    AUTH_LDAP_USER_SEARCH = LDAPSearch(
        env.str('AUTH_LDAP_USER_SEARCH_BASE', None),
        ldap.SCOPE_SUBTREE,
        LDAP_DEFAULT_FILTERSTR,
    )
    AUTH_LDAP_USER_ATTR_MAP = LDAP_DEFAULT_ATTR_MAP
    AUTH_LDAP_USERNAME_DOMAIN = env.str('AUTH_LDAP_USERNAME_DOMAIN', None)
    AUTH_LDAP_DOMAIN_PRINTABLE = env.str(
        'AUTH_LDAP_DOMAIN_PRINTABLE', AUTH_LDAP_USERNAME_DOMAIN
    )

    AUTHENTICATION_BACKENDS = tuple(
        itertools.chain(
            ('projectroles.auth_backends.PrimaryLDAPBackend',),
            AUTHENTICATION_BACKENDS,
        )
    )

    # Secondary LDAP server (optional)
    if ENABLE_LDAP_SECONDARY:
        AUTH_LDAP2_SERVER_URI = env.str('AUTH_LDAP2_SERVER_URI', None)
        AUTH_LDAP2_BIND_DN = env.str('AUTH_LDAP2_BIND_DN', None)
        AUTH_LDAP2_BIND_PASSWORD = env.str('AUTH_LDAP2_BIND_PASSWORD', None)
        AUTH_LDAP2_CONNECTION_OPTIONS = LDAP_DEFAULT_CONN_OPTIONS

        AUTH_LDAP2_USER_SEARCH = LDAPSearch(
            env.str('AUTH_LDAP2_USER_SEARCH_BASE', None),
            ldap.SCOPE_SUBTREE,
            LDAP_DEFAULT_FILTERSTR,
        )
        AUTH_LDAP2_USER_ATTR_MAP = LDAP_DEFAULT_ATTR_MAP
        AUTH_LDAP2_USERNAME_DOMAIN = env.str('AUTH_LDAP2_USERNAME_DOMAIN')
        AUTH_LDAP2_DOMAIN_PRINTABLE = env.str(
            'AUTH_LDAP2_DOMAIN_PRINTABLE', AUTH_LDAP2_USERNAME_DOMAIN
        )

        AUTHENTICATION_BACKENDS = tuple(
            itertools.chain(
                ('projectroles.auth_backends.SecondaryLDAPBackend',),
                AUTHENTICATION_BACKENDS,
            )
        )

```

Modifying SODAR_CONSTANTS (Optional)

String identifiers used globally in SODAR project management are defined in the `SODAR_CONSTANTS` dictionary. It can be imported into your app code with the import:

```
from projectroles.models import SODAR_CONSTANTS
```

If you need to update or extend the constants for use your site, you can import the default dictionary into your Django settings and modify it as necessary with the following snippet:

```
from projectroles.constants import get_sodar_constants
SODAR_CONSTANTS = get_sodar_constants(default=True)
# Your changes here..
```

Warning: Modifying existing default constants may result in unwanted issues, especially on a site which already contains created projects. Proceed with caution!

Logging (Optional)

It is recommended to add “projectroles” under `LOGGING['loggers']`. For production, INFO debug level is recommended.

1.2.4 Projectroles Usage

This document provides instructions for using the `projectroles` app which has been integrated into your Django site.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Hint: Detailed instructions for many pages can be found in an interactive tour by clicking the “Help” link in the right side of the top navigation bar.

Before reading this document, be sure to see *Projectroles Basics* for basic concepts regarding the use of this app.

Logging In

Apart from specific public or token-enabled views, user login is **mandatory** for using a SODAR Core based Django site.

One can either log in using a local Django user or, if LDAP/AD is enabled, their LDAP/AD credentials from a supported site. In the latter case, the user domain must be appended to the user name in form of `user@DOMAIN`.

User Interface

Basics

Upon login into a SODAR Core based Django site using default templates and CSS, the general view of your site is split into the following elements:

Log In

Please log in.

Fig. 1: SODAR login form

- **Top navigation bar:** Contains the site logo and title, search element, help link and the user dropdown menu.
- **User dropdown menu:** Contains links to user management, admin site and site-wide apps the user has access to.
- **Project sidebar:** Shortcuts to project apps and project management pages
- **Project navigation:** Project structure breadcrumb (disabled for site apps)
- **Content:** Actual app content goes in this element
- **Footer:** Optional footer with e.g. site info and version

SODAR Core Example Site Beta

Search term [Help](#)

Home

Create Category

Home

Available Projects

Project	Description	Your Role
Test Category	A test category	Superuser
Test Project	A test project	Superuser

Set the content for your footer in include/_footer.html. Example Site v0.1.0 / SODAR Core v0.2.1+93.gf7850ed.dirty

Fig. 2: Home view

Home View

As content within a SODAR Core based site is by default sorted into projects, the home view displays a tree view of categories and projects to choose from. You can filter the list with a search term or restrict display to your starred

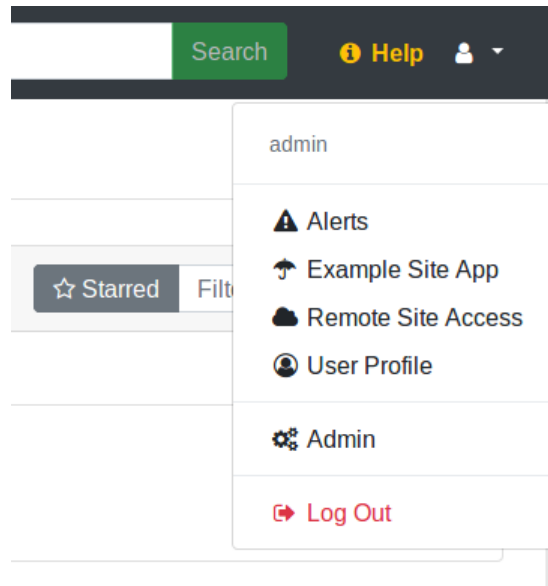


Fig. 3: User dropdown

projects.

Project Detail View

The project detail page dynamically imports elements from installed project apps, usually showing e.g. overview of latest additions to app data, statistics and/or shortcuts to app functionalities. Here you can also access project apps from the project sidebar. For project apps, the sidebar link leads to the app entry point view as defined in the app plugin.

For each page in a project app which extends the default projectroles template layout, the **project title bar** is displayed on the top of the page. This contains the project title and description and a link to “star” the project into your favourites. Below this, the **project app title bar** with possible app-specific controls is usually displayed.

Category and Project Management

In SODAR based sites, data is split into **categories** and **projects**. Categories may be freely nested and are used as containers of projects. They may contain a description and readme, but project apps and user roles beyond owner are disabled for categories. Projects can not be nested within each other.

Creating a Top Level Category

Currently, only users with a superuser status can create a top level category. This can be done by navigating to the *home view* and clicking the **Create Category** link. To create a category, a name and owner must be supplied, along with optional description and/or a readme document. All of these may be modified later.

Note: Currently, only users already previously logged into the system can be added as the owner of a category or project. The ability to invite users not yet on the site as owners will be added later.

SODAR Core Example Site Beta

Search term [Help](#)

> Test Category / Test Project

Test Project A test project

ReadMe

No ReadMe is currently set for this project. [You can update the ReadMe here.](#)

Project Timeline Overview

Timeline of project events

Timestamp	Event	User	Description	Status
2018-10-16 14:26:14	update_remote	admin	update remote access for site Example Dev Target to READ_ROLES)	OK
2018-10-15 18:39:34	project_create	admin	create project with admin as owner	OK

Example Project App Overview

This is a minimal example for a project app

This is the project overview card for [example_project_app](#)

Fig. 4: Project detail view

Hint: When setting up a new site, think about what kind of category and project structure makes sense for your team and organization. Moving projects and categories under different categories is possible, but is not recommended and can currently only be done via the admin view or directly in the Django shell.

Creating Projects

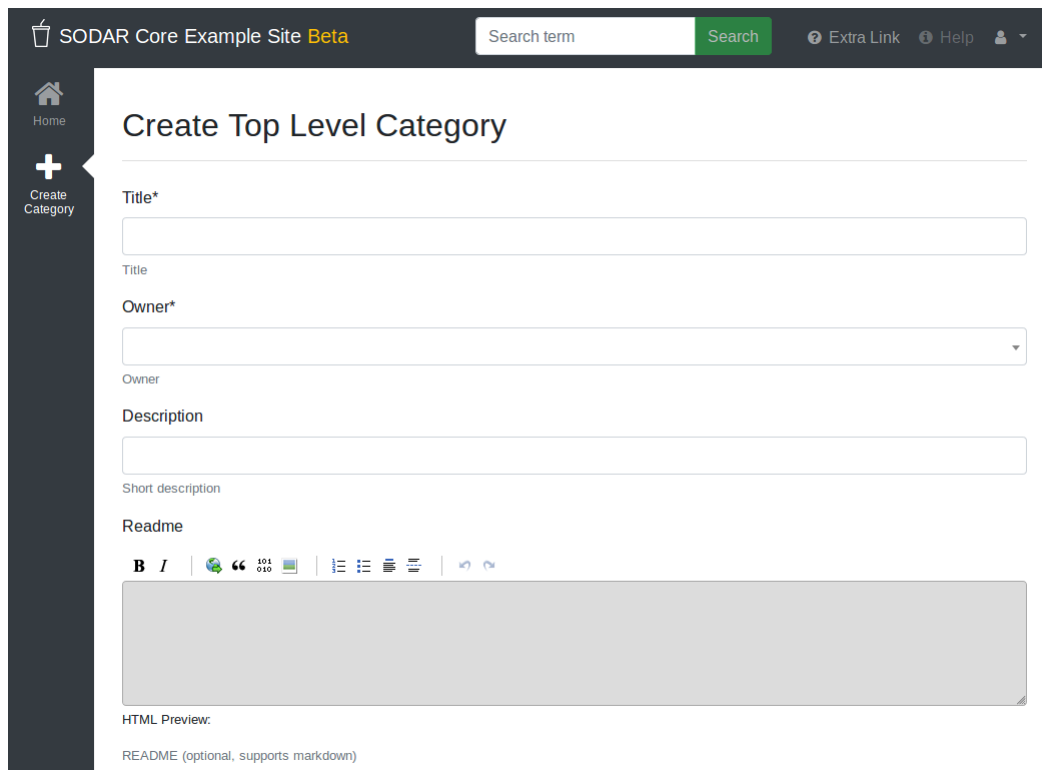
Once navigating into a category, a user with sufficient access will see the **Create Project or Category** link in the side bar. This opens up a form for adding a project or a nested category under the current category. The form is identical to top level category creation, except that you can also choose between creating a project or a category.

Updating Projects

An existing project or category can be updated from the **Update Project/Category** link in the side bar. Again, a similar form as before will be presented to the user.

App Settings

Project apps may define project or user specific settings, modifiable by users with sufficient project access. Widgets for project specific settings will show up in the project creation and updating form. User specific settings will be displayed in the *Userprofile app*.



The screenshot shows the 'Create Top Level Category' form in the SODAR Core interface. The header bar includes the site name 'SODAR Core Example Site Beta', a search bar, and links for 'Extra Link', 'Help', and a user profile. The left sidebar has a 'Home' link and a 'Create Category' link with a plus icon. The main form area is titled 'Create Top Level Category' and contains the following fields:

- Title***: A text input field.
- Owner***: A dropdown menu.
- Description**: A text input field.
- Readme**: A rich text editor with formatting tools (bold, italic, link, unlink, list, ul, ol, table, image, quote, code, link, unlink) and a large text area. Below the editor is an 'HTML Preview' section.

Below the HTML preview, it says 'README (optional, supports markdown)'.

Fig. 5: Category/project creation form

Note: Currently, project specific app settings are also enabled for categories but do not actually do anything. The behaviour regarding this (remove settings / inherit by nested projects / etc) is TBD.

Member Management

Project member roles can be viewed and modified through the **Project Members** link on the sidebar. Modification requires a sufficient role in the project (owner or delegate) or superuser status.

Adding Members

There are two ways to add new members to a project:

- **Add Member** is used to add member roles to system users.
- **Invite Member** is used to send email invites to users not yet registered in the system.

Addition or modification of users sends an email notification to the user in question if email sending is enabled on your Django server. The emails can be previewed in corresponding forms.

Hint: As of SODAR Core v0.4.5, it is also possible to create an invite in the “add member” form. Inviting is enabled when inputting an email address not found among the system users.

SODAR Core Example Site Beta

Search term Search

Extra Link Help

Home / Test Core Category / Test Core Project

Test Core Project ☆ Updating description

Update Project

Title*

Test Core Project

Owner*

admin

Description

This is a test project

Readme

TODO

- * Add readme here
- * Update documentation screenshots

HTML Preview:

Fig. 6: Category/project updating form

SODAR Core Example Site Beta

Search term Search Help

> Test Category / Test Project

Test Project ☆ A test project

Project Members

User	Name	Email	Role
admin		admin@example.com	project owner

Member Operations

- Add Member
- Send Invite
- View Members
- View Invites
- Import Members

Fig. 7: Project member list view

Modifying Members

Changing or removing user roles can be done from links next to each role on the member list. The exception for this is the *project owner* role which can only be modified on the **Update Project** page.

Invites

Invites are accepted by the responding user clicking on a link supplied in their invite email and logging in to the site with their LDAP/AD credentials. Invites expire after a certain time and can be reissued or revoked on the **Project Invites** page.

Remote Projects

It is possible to sync project metadata and member roles between multiple SODAR Core based Django sites. Remote sites and access can be managed in the **Remote Site Access** site app, found in the user dropdown menu in the top navigation bar.

In the current implementation, your django site must either be in **source** or **target** mode. A source site can define one or multiple target sites where project data can be provided. A target site can define exactly one source site, from which project data can be retrieved from.

Note: These are arbitrary restrictions which may be relaxed in the future, if use cases warrant it.

To enable remote project data reading, you must first set up either a target or a source site depending on the role of your own SODAR site.

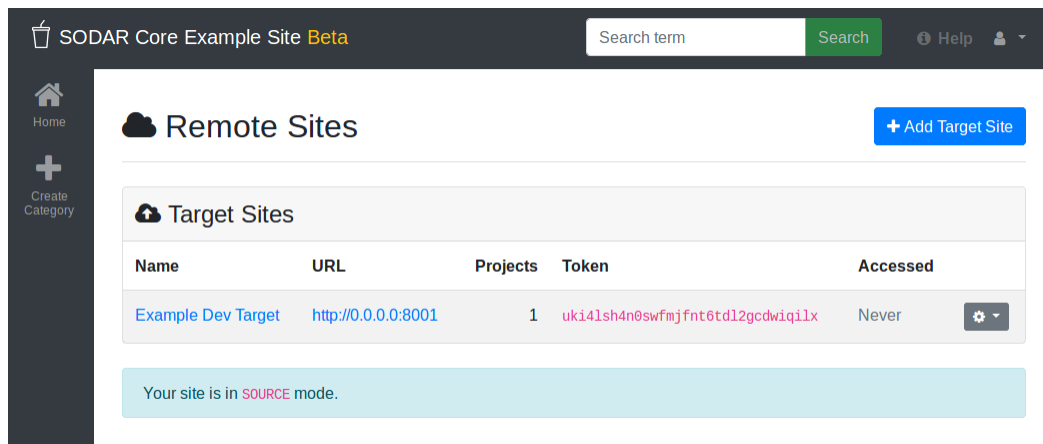


Fig. 8: Remote site list in source mode

As Source Site

Navigate to the **Remote Site Access** site app and click on the *Add Target Site* link. You will be provided with a form for specifying the remote site. A secret string is generated automatically and you need to provide this to the administrator of the target site in question for accessing your site.

Once created, you can access the list of projects on your site in regards to the created target site. For each project, you may select an access level, of which two are currently implemented:

- **No access:** No access on the remote site (default)
- **Read roles:** This allows for the target site to read project metadata *and* user roles in order to synchronize project access remotely.

Note: The *read roles* access level also provides metadata of the categories above the selected project so that the project structure can be maintained.

Note: Only LDAP/AD user roles and local administrator *owner* roles are provided to the target site. Other local user roles are ignored.

Note: Access levels for purely checking the existence of the project and only reading project metadata (title, description..) without member roles are implemented in the data model and backend, but currently disabled in the UI.

Once desired access to specific projects has been granted and confirmed, the target site will sync the data by sending a request to the source site.

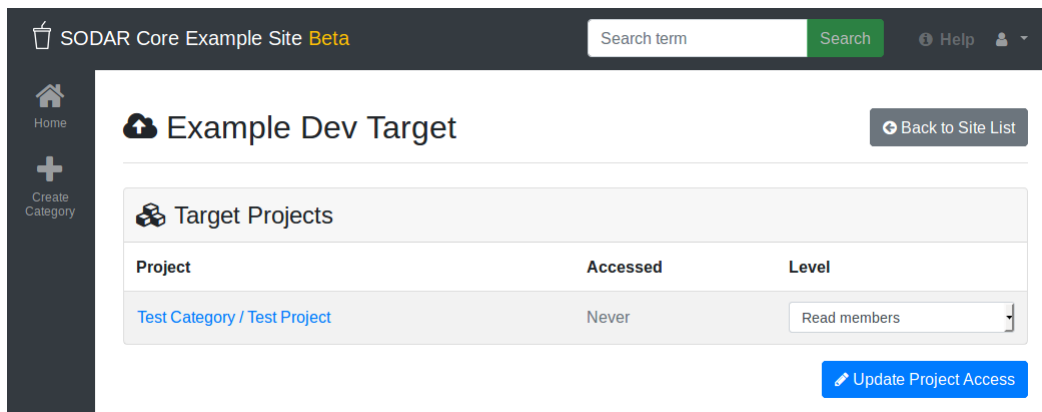


Fig. 9: Remote project list in source mode

As Target Site

The source site should be set up as above using the *Set Source Site* link, using the provided secret string as the access token.

After creating the source site, remote project metadata and member roles (for which access has been granted) can be accessed using the *Synchronize* link.

Alternatively, the following management command can be used:

```
$ ./manage.py syncremote
```

Note: If categories or projects with the same name within the same parent exist under a different UUID, they or their child projects will **not** be synchronized.

Note: If a local user is the owner of a synchronized project on the source site, the user defined in the `PROJECTROLES_DEFAULT_ADMIN` will be given the owner role. Hence you **must** have this setting defined if you are implementing a SODAR site in target mode.

Search

The search form is displayed in the top navigation bar if enabled. It currently takes one string as a search parameter, followed by optional keyword argument. At this time, the keyword of `type` has been implemented, used to limit the search to a certain data type as specified in app plugins.

Search results are split into results from different apps. For example, entering `test` will return all objects from all apps containing this string. Alternatively, entering `test type:project` will provide results from any app configured to produce results of type *project*. By default, this will result in the `projectroles` app listing projects which contain the search string in their name and/or description.

Note: Multiple search terms, complex search strings, full-text search and additional keywords/operators will be defined in the future.

1.2.5 Projectroles Customization

Here you can find some customization instructions and tips for `projectroles` and SODAR Core.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

CSS Overrides

If some of the CSS definitions in `{STATIC}/projectroles/css/projectroles.css` do not suit your purposes, it is possible to override them in your own includes. It is still recommended to include the “*Flexbox page setup*” section as provided.

In this chapter are examples of overrides you can place e.g. in `project.css` to change certain defaults.

Hint: While not explicitly mentioned, some parameters may require the `!important` argument to take effect on your site.

Warning: In the future we may instead offer a full Bootstrap 4 theme, which may deprecate current overriding/extending CSS classes.

Static Element Coloring

If you wish to recolor the background of the static elements on the page (title bar, side bar and project navigation breadcrumb), add the following CSS overrides.

```
.sodar-base-navbar, .sodar-pr-sidebar, .sodar-pr-sidebar-nav {
    background-color: #ff00ff;
}

.sodar-pr-navbar {
    background-color: #00ff00;
}
```

Sidebar Width

If the sidebar is not wide enough for your liking or e.g. a name of an app overflowing, the sidebar can be resized with the following override:

```
.sodar-pr-sidebar {
    width: 120px;
}
```

Title Bar

You can implement your own title bar by replacing the default base.html include of projectroles/_site_titlebar.html with your own HTML or include.

When doing this, it is possible to include elements from the default title bar separately:

- Search form: projectroles/_site_titlebar_search.html
- Site app and user operation dropdown: projectroles/_site_titlebar_dropdown.html

See the templates themselves for further instructions.

Additional Title Bar Links

If you want to add additional links *not* related to apps in the title bar, you can implement in the template file {SITE_NAME}/templates/include/_titlebar_nav.html. This can be done for e.g. documentation links or linking to external sites. Example:

```
{# Example extra link #}
<li class="nav-item">
  <a href="#" class="nav-link" id="site-extra-link-x" target="_blank">
    <i class="fa fa-fw fa-question-circle"></i> Extra Link
  </a>
</li>
```

Site Icon

An optional site icon can be placed into {STATIC}/images/logo_navbar.png to be displayed in the default Projectroles title bar.

Project Breadcrumb

To add custom content in the end of the default project breadcrumb, use {% block nav_sub_project_extend %} in your app template.

The entire breadcrumb element can be overridden by declaring `{% block nav_sub_project %} block` in your app template.

Footer

Footer content can be specified in the optional template file `{SITE_NAME}/templates/include/_footer.html`.

Project and Category Display Names

If the *project* and *category* labels don't match your use case, it is possible to change the labels displayed to the user by editing `SODAR_CONSTANTS` in your Django site settings file. Example:

```
SODAR_CONSTANTS = get_sodar_constants(default=True)
SODAR_CONSTANTS['DISPLAY_NAMES']['CATEGORY'] = {
    'default': 'not-a-category',
    'plural': 'non-categories',
}
SODAR_CONSTANTS['DISPLAY_NAMES']['PROJECT'] = {
    'default': 'not-a-project',
    'plural': 'non-projects',
}
```

See more about overriding `SODAR_CONSTANTS` [here](#).

To print out these values in your views or templates, call the `get_display_name()` function, which is available both as a template tag through `projectroles_common_tags.py` and a general utility function in `utils.py`. Capitalization and pluralization are handled by the function according to arguments. See the [API documentation](#) for details.

Note: These changes will **not** affect role names or IDs and descriptions of Timeline events.

1.2.6 Projectroles API Documentation

This document contains API documentation for the `projectroles` app. Included are functionalities and classes intended to be used by other applications.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Plugins

SODAR plugin point definitions and helper functions for plugin retrieval are detailed in this section.

Plugin point definitions and plugin API for apps based on projectroles

```
class projectroles.plugins.BackendPluginPoint
    Projectroles plugin point for registering backend apps

    get_api()
        Return API entry point object.
```


get_statistics()

Return backend statistics as a dict. Should take the form of {id: {label, value, url (optional), description (optional)}}.

Returns Dict

class projectroles.plugins.**ProjectAppPluginPoint**

Projectroles plugin point for registering project specific apps

get_extra_data_link(*_extra_data*, *_name*)

Return a link for the given timeline label that stars with "extra:".

get_object(*model*, *uuid*)

Return object based on the model class and the object's SODAR UUID.

Parameters

- **model** – Object model class
- **uuid** – sodar_uuid of the referred object

Returns Model object or None if not found

Raise NameError if model is not found

get_object_link(*model_str*, *uuid*)

Return the URL for referring to a object used by the app, along with a label to be shown to the user for linking.

Parameters

- **model_str** – Object class (string)
- **uuid** – sodar_uuid of the referred object

Returns Dict or None if not found

get_project_list_value(*column_id*, *project*)

Return a value for the optional additional project list column specific to a project.

Parameters

- **column_id** – ID of the column (string)
- **project** – Project object

Returns String (may contain HTML) or None

get_statistics()

Return app statistics as a dict. Should take the form of {id: {label, value, url (optional), description (optional)}}.

Returns Dict

get_taskflow_sync_data()

Return data for synchronizing taskflow operations.

Returns List of dicts or None.

search(*search_term*, *user*, *search_type=None*, *keywords=None*)

Return app items based on a search term, user, optional type and optional keywords.

Parameters

- **search_term** – String
- **user** – User object for user initiating the search

- **search_type** – String
- **keywords** – List (optional)

Returns Dict

update_cache (*name=None, project=None, user=None*)

Update cached data for this app, limitable to item ID and/or project.

Parameters

- **name** – Item name to limit update to (string, optional)
- **project** – Project object to limit update to (optional)
- **user** – User object to denote user triggering the update (optional)

urls = []

App URLs (will be included in settings by.djangoplugins)

class projectroles.plugins.**RemoteSiteAppPlugin**

Site plugin for remote site and project management

app_permission = 'userprofile.update_remote'

Required permission for displaying the app

description = 'Management of remote SODAR sites and remote project access'

Description string

entry_point_url_id = 'projectroles:remote_sites'

Entry point URL ID

icon = 'cloud'

FontAwesome icon ID string

name = 'remotesites'

Name (slug-safe, used in URLs)

title = 'Remote Site Access'

Title (used in templates)

urls = []

App URLs (will be included in settings by.djangoplugins)

class projectroles.plugins.**SiteAppPluginPoint**

Projectroles plugin point for registering site-wide apps

get_messages (*user=None*)

Return a list of messages to be shown to users.

Parameters **user** – User object (optional)

Returns List of dicts or and empty list if no messages

projectroles.plugins.**change_plugin_status** (*name, status, plugin_type='app'*)

Change the status of a selected plugin in the database.

Parameters

- **name** – Plugin name (string)
- **status** – Status (int, see.djangoplugins)
- **plugin_type** – Type of plugin (“app”, “backend” or “site”)

Raise ValueError if plugin_type is invalid or plugin with name not found

`projectroles.plugins.get_active_plugins(plugin_type='project_app')`

Return active plugins of a specific type.

Parameters `plugin_type` – “project_app”, “site_app” or “backend” (string)

Returns List or None

Raise `ValueError` if `plugin_type` is not recognized

`projectroles.plugins.get_app_plugin(plugin_name)`

Return active app plugin.

Parameters `plugin_name` – Plugin name (string)

Returns `ProjectAppPlugin` object or None if not found

`projectroles.plugins.get_backend_api(plugin_name, force=False)`

Return backend API object.

Parameters

- **plugin_name** – Plugin name (string)
- **force** – Return plugin regardless of status in `ENABLED_BACKEND_PLUGINS`

Returns Plugin object or None if not found

Models

Projectroles models are used by other apps for project access and metadata management as well as linking objects to projects.

class `projectroles.models.AppSettings(*args, **kwargs)`

Project and users settings value.

The settings are defined in the “app_settings” member in a SODAR project app’s plugin. The scope of each setting can be either “USER” or “PROJECT”, defined for each setting in `app_settings`. Project AND user-specific settings or settings which don’t belong to either are currently not supported.

exception `DoesNotExist`

exception `MultipleObjectsReturned`

app_plugin

App to which the setting belongs

get_value()

Return value of the setting in the format specified in ‘type’

name

Name of the setting

project

Project to which the setting belongs

save(*args, **kwargs)

Version of `save()` to convert ‘value’ data according to ‘type’

sodar_uuid

`AppSetting` SODAR UUID

type

Type of the setting

user
Project to which the setting belongs

value
Value of the setting

class projectroles.models.**AppSettingManager**

Manager for custom table-level AppSetting queries

get_setting_value (*app_name, setting_name, project=None, user=None*)
Return value of setting_name for app_name in project or for user.

Note that either project or user must be None but not both.

Parameters

- **app_name** – App plugin name (string)
- **setting_name** – Name of setting (string)
- **project** – Project object or pk
- **user** – User object or pk

Returns Value (string)

Raise AppSetting.DoesNotExist if setting is not found

class projectroles.models.**Project** (**args, **kwargs*)

A SODAR project. Can have one parent category in case of nested projects. The project must be of a specific type, of which “CATEGORY” and “PROJECT” are currently implemented. “CATEGORY” projects are used as containers for other projects

exception DoesNotExist

exception MultipleObjectsReturned

description
Short project description

get_children ()
Return child objects for the Project sorted by title

get_delegates ()
Return RoleAssignments for delegates

get_depth ()
Return depth of project in the project tree structure (root=0)

get_full_title ()
Return full title of project (just an alias for __str__())

get_members ()
Return RoleAssignments for members of project excluding owner and delegates

get_owner ()
Return RoleAssignment for owner or None if not set

get_parents ()
Return an array of parent projects in inheritance order

get_source_site ()
Return source site or None if this is a locally defined project

has_role (*user*, *include_children=False*)
Return whether user has roles in Project. If *include_children* is True, return True if user has roles in ANY child project

is_remote ()
Return True if current project has been retrieved from a remote SODAR site

parent
Parent category if nested, otherwise null

readme
Project README (optional, supports markdown)

save (**args*, ***kwargs*)
Version of *save()* to include custom validation for Project

sodar_uuid
Project SODAR UUID

submit_status
Status of project creation

title
Project title

type
Type of project ("CATEGORY", "PROJECT")

class `projectroles.models.ProjectInvite` (**args*, ***kwargs*)
Invite which is sent to a non-logged in user, determining their role in the project.

exception `DoesNotExist`

exception `MultipleObjectsReturned`

active
Status of the invite (False if claimed or revoked)

date_created
DateTime of invite creation

date_expire
Expiration of invite as DateTime

email
Email address of the person to be invited

issuer
User who issued the invite

message
Message to be included in the invite email (optional)

project
Project to which the person is invited

role
Role assigned to the person

secret
Secret token provided to user with the invite

sodar_uuid
ProjectInvite SODAR UUID

```
class projectroles.models.ProjectManager
    Manager for custom table-level Project queries

    find (search_term, keywords=None, project_type=None)
        Return projects with a partial match in full title or, including titles of parent Project objects, or the descrip-
        tion of the current object. Restrict to project type if project_type is set. :param search_term: Search term
        (string) :param keywords: Optional search keywords as key/value pairs (dict) :param project_type: Project
        type or None :return: List of Project objects

class projectroles.models.ProjectUserTag (*args, **kwargs)
    Tag assigned by a user to a project

    exception DoesNotExist

    exception MultipleObjectsReturned

    name
        Name of tag to be assigned

    project
        Project to which the tag is assigned

    sodar_uuid
        ProjectUserTag SODAR UUID

    user
        User for whom the tag is assigned

class projectroles.models.RemoteProject (*args, **kwargs)
    Remote project relation

    exception DoesNotExist

    exception MultipleObjectsReturned

    date_access
        DateTime of last access from/to remote site

    get_project ()
        Get the related Project object

    level
        Project access level

    project
        Related project object (if created locally)

    project_uuid
        Related project UUID

    site
        Related remote SODAR site

    sodar_uuid
        RemoteProject relation UUID (local)

class projectroles.models.RemoteSite (*args, **kwargs)
    Remote SODAR site

    exception DoesNotExist

    exception MultipleObjectsReturned

    description
        Site description
```

```
get_access_date ()
    Return date of latest project access by remote site

get_url ()
    Return sanitized site URL

mode
    Site mode

name
    Site name

save (*args, **kwargs)
    Version of save() to include custom validation

secret
    Secret token used to connect to the master site

sodar_uuid
    RemoteSite relation UUID (local)

url
    Site URL

class projectroles.models.Role (*args, **kwargs)
    Role definition, used to assign roles to projects in RoleAssignment

exception DoesNotExist

exception MultipleObjectsReturned

description
    Role description

name
    Name of role

class projectroles.models.RoleAssignment (*args, **kwargs)
    Assignment of an user to a role in a project. One role per user is allowed for each project. Roles of project owner
    and project delegate assignments might be limited (to PROJECTROLES_DELEGATE_LIMIT) per project.

exception DoesNotExist

exception MultipleObjectsReturned

project
    Project in which role is assigned

role
    Role to be assigned

save (*args, **kwargs)
    Version of save() to include custom validation for RoleAssignment

sodar_uuid
    RoleAssignment SODAR UUID

user
    User for whom role is assigned

class projectroles.models.RoleAssignmentManager
    Manager for custom table-level RoleAssignment queries

get_assignment (user, project)
    Return assignment of user to project, or None if not found
```

```
class projectroles.models.SODARUser(*args, **kwargs)
    SODAR compatible abstract user model

    get_full_name()
        Return full name or username if not set

    sodar_uuid
        User SODAR UUID

projectroles.models.assign_user_group(sender, user, **kwargs)
    Signal for user group assignment

projectroles.models.handle_ldap_login(sender, user, **kwargs)
    Signal for LDAP login handling
```

App Settings

Projectroles provides an API for getting or setting project and user specific settings.

```
class projectroles.app_settings.AppSettingAPI

    classmethod get_all_defaults(scope)
        Get all default settings for a scope.

        Parameters scope –

        Returns

    classmethod get_all_settings(project=None, user=None)
        Return all setting values. If the value is not found, return the default.

        Parameters

        • project – Project object (can be None)

        • user – User object (can be None)

        Returns Dict

        Raise ValueError if neither project nor user are set

    classmethod get_app_setting(app_name, setting_name, project=None, user=None)
        Return app setting value for a project or an user. If not set, return default.

        Parameters

        • app_name – App name (string, must correspond to “name” in app plugin)

        • setting_name – Setting name (string)

        • project – Project object (can be None)

        • user – User object (can be None)

        Returns String or None

        Raise KeyError if nothing is found with setting_name

    classmethod get_default_setting(app_name, setting_name)
        Get default setting value from an app plugin.

        Parameters

        • app_name – App name (string, must correspond to “name” in app plugin)
```


- **setting_name** – Setting name (string)

Returns Setting value (string, integer or boolean)

Raise KeyError if nothing is found with setting_name

classmethod **get_setting_defs** (*plugin, scope*)

Return app setting definitions of a specific scope from a plugin.

Parameters

- **plugin** – project app plugin object extending ProjectAppPluginPoint
- **scope** – PROJECT or USER

Returns Dict

Raise ValueError if scope is invalid

classmethod **set_app_setting** (*app_name, setting_name, value, project=None, user=None, validate=True*)

Set value of an existing project or user settings. Creates the object if not found.

Parameters

- **app_name** – App name (string, must correspond to “name” in app plugin)
- **setting_name** – Setting name (string)
- **value** – Value to be set
- **project** – Project object (can be None)
- **user** – User object (can be None)
- **validate** – Validate value (bool, default=True)

Returns True if changed, False if not changed

Raise ValueError if validating and value is not accepted for setting type

Raise ValueError if neither project nor user are set

Raise KeyError if setting name is not found in plugin specification

classmethod **validate_setting** (*setting_type, setting_value*)

Validate setting value according to its type.

Parameters

- **setting_type** – Setting type
- **setting_value** – Setting value

Raise ValueError if setting_type or setting_value is invalid

Common Template Tags

These tags can be included in templates with `{% load 'projectroles_common_tags' %}`.

Template tags provided by projectroles for use in other apps

`projectroles.template_tags.projectroles_common_tags.check_backend(name)`

Return True if backend app is available, else False

`projectroles.template_tags.projectroles_common_tags.core_version()`

Return the SODAR Core version

`projectroles.templatetags.projectroles_common_tags.force_wrap(s, length)`
Force wrapping of string

`projectroles.templatetags.projectroles_common_tags.get_class(obj, lower=False)`
Return object class as string

`projectroles.templatetags.projectroles_common_tags.get_display_name(key, title=False, count=1, plural=False)`
Return display name from SODAR_CONSTANTS

`projectroles.templatetags.projectroles_common_tags.get_full_url(request, url)`
Get full URL based on a local URL

`projectroles.templatetags.projectroles_common_tags.get_history_dropdown(project, obj)`
Return link to object timeline events within project

`projectroles.templatetags.projectroles_common_tags.get_info_link(content, html=False)`
Return info popover link icon

`projectroles.templatetags.projectroles_common_tags.get_project_by_uuid(sodar_uuid)`
Return Project by sodar_uuid

`projectroles.templatetags.projectroles_common_tags.get_project_link(project, full_title=False, request=None)`
Return link to project with a simple or full title

`projectroles.templatetags.projectroles_common_tags.get_project_title_html(project)`
Return HTML version of the full project title including parents

`projectroles.templatetags.projectroles_common_tags.get_remote_icon(project, request)`
Get remote project icon HTML

`projectroles.templatetags.projectroles_common_tags.get_setting(name, js=False)`
Return value of Django setting by name or None if it is not found. Return a Javascript-safe value if js=True.

`projectroles.templatetags.projectroles_common_tags.get_user_by_username(username)`
Return User by username

`projectroles.templatetags.projectroles_common_tags.get_user_html(user)`
Return standard HTML representation for a User object

`projectroles.templatetags.projectroles_common_tags.highlight_search_term(item, term)`
Return string with search term highlighted

`projectroles.templatetags.projectroles_common_tags.render_markdown(raw_markdown)`
Markdown field rendering helper

`projectroles.templatetags.projectroles_common_tags.site_version()`
Return the site version

`projectroles.templatetags.projectroles_common_tags.static_file_exists(path)`
Return True/False based on whether a static file exists

`projectroles.template_tags.projectroles_common_tags.template_exists(path)`
Return True/False based on whether a template exists

Utilities

General utility functions are stored in `utils.py`.

`projectroles.utils.build_invite_url(invite, request)`
Return invite URL for a project invitation.

Parameters

- **invite** – ProjectInvite object
- **request** – HTTP request

Returns URL (string)

`projectroles.utils.build_secret(length=32)`
Return secret string for e.g. public URLs.

Parameters **length** – Length of string if specified, default value from settings

Returns Randomized secret (string)

`projectroles.utils.get_app_names()`
Return list of names for locally installed non-django apps

`projectroles.utils.get_display_name(key, title=False, count=1, plural=False)`
Return display name from SODAR_CONSTANTS.

Parameters

- **key** – Key in SODAR_CONSTANTS['DISPLAY_NAMES'] to return (string)
- **title** – Return name in title case if true (boolean, optional)
- **count** – Item count for returning plural form, overrides plural=False if not 1 (int, optional)
- **plural** – Return plural form if True, overrides count != 1 if True (boolean, optional)

Returns String

`projectroles.utils.get_expiry_date()`
Return expiry date based on current date + INVITE_EXPIRY_DAYS

Returns DateTime object

`projectroles.utils.get_user_display_name(user, inc_user=False)`
Return full name of user for displaying.

Parameters

- **user** – User object
- **inc_user** – Include user name if true (boolean)

Returns String

`projectroles.utils.set_user_group(user)`
Set user group based on user name.

1.3 Adminalerts App

The `adminalerts` site app enables system administrators to display site-wide messages to all users with an expiration date.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

1.3.1 Basics

The app displays un-dismissable small alerts on the top of page content to all users. They can be used to e.g. warn users of upcoming downtime or highlight recently deployed changes.

Upon creation, an expiration date is set for each alert. Alerts can also be freely enabled, disabled or deleted by superuser on the app UI. Additional information regarding an alert can be provided with Markdown syntax and viewed on a separate details page.

1.3.2 Installation

Warning: To install this app you **must** have the `django-sodar-core` package installed and the `projectroles` app integrated into your Django site. See the [projectroles integration document](#) for instructions.

Django Settings

The `adminalerts` app is available for your Django site after installing `django-sodar-core`. Add the app into `THIRD_PARTY_APPS` as follows:

```
THIRD_PARTY_APPS = [
    # ...
    'adminalerts.apps.AdminalertsConfig',
]
```

Optional Settings

To alter default `adminalerts` app settings, insert the following **optional** variables with values of your choosing:

```
# Adminalerts app settings
ADMINALERTS_PAGINATION = 15      # Number of alerts to be shown on one page (int)
```

URL Configuration

In the Django URL configuration file, add the following line under `urlpatterns` to include `adminalerts` URLs in your site.

```
urlpatterns = [
    # ...
    url(r'^alerts/', include('adminalerts.urls')),
]
```

Migrate Database and Register Plugin

To migrate the Django database and register the adminalerts site app plugin, run the following management command:

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

In addition to the database migration operation, you should see the following output:

```
Registering Plugin for admimnalert.plugins.SiteAppPlugin
```

1.3.3 Usage

When logged in as a superuser, you can find the “Alerts” option in your user dropdown menu in the top right corner of the site. Using the UI, you can add, modify and delete alerts shown to users.

This application is not available for users with a non-superuser status.

1.4 Bgjobs App

The `bgjobs` app allows for the management of project-specific and asynchronous server-side background jobs.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

TODO: Docs to be filled out

1.4.1 Bgjobs Installation

This document provides instructions and guidelines for installing the `bgjobs` app to be used with your SODAR Core enabled Django site.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Warning: To install this app you **must** have the `django-sodar-core` package installed and the `projectroles` app integrated into your Django site. See the [projectroles integration document](#) for instructions.

Django Settings

The `bgjobs` app is available for your Django site after installing `django-sodar-core`. Add the app into `THIRD_PARTY_APPS` as follows:

```
THIRD_PARTY_APPS = [
    # ...
    'bgjobs.apps.BgjobsConfig',
]
```

URL Configuration

In the Django URL configuration file, add the following line under `urlpatterns` to include `bgjobs` URLs in your site.

```
urlpatterns = [  
    # ...  
    url(r'^bgjobs/', include('bgjobs.urls')),  
]
```

Migrate Database and Register Plugin

To migrate the Django database and register the `bgjobs` app and job type plugins, run the following management command:

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

In addition to the database migration operation, you should see the following output:

```
Registering Plugin for bgjobs.plugins.ProjectAppPlugin  
Registering Plugin for bgjobs.plugins.BackgroundJobsPluginPoint
```

Celery Setup

TODO

1.4.2 Bgjobs Usage

Usage instructions for the `bgjobs` app are detailed in this document.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

TODO

1.5 Filesfolders App

The `filesfolders` app enables uploading small files into the Django database and organizing them in folders. It also permits creating hyperlinks, providing public links to files and automated unpacking of ZIP archives.

The app is displayed as “*Small Files*” on the SODAR site.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

1.5.1 Filesfolders Installation

This document provides instructions and guidelines for installing the `filesfolders` app to be used with your SODAR Core enabled Django site.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Warning: To install this app you **must** have the `django-sodar-core` package installed and the `projectroles` app integrated into your Django site. See the [projectroles integration document](#) for instructions.

Django Settings

The `filesfolders` app is available for your Django site after installing `django-sodar-core`. Add the app, along with the prerequisite `django_db_storage` app into `THIRD_PARTY_APPS` as follows:

```
THIRD_PARTY_APPS = [
    # ...
    'filesfolders.apps.FilesfoldersConfig',
    'db_file_storage',
]
```

Next set the `db_file_storage` app as the default storage app for your site:

```
DEFAULT_FILE_STORAGE = 'db_file_storage.storage.DatabaseFileStorage'
```

Fill out `filesfolders` app settings to fit your site. The settings variables are explained below:

- `FILESFOLDERS_MAX_UPLOAD_SIZE`: Max size for an uploaded file in bytes (int)
- `FILESFOLDERS_MAX_ARCHIVE_SIZE`: Max size for an archive file to be unpacked in bytes (int)
- `FILESFOLDERS_SERVE_AS_ATTACHMENT`: If true, always serve downloaded files as attachment instead of opening them in browser (bool)
- `FILESFOLDERS_LINK_BAD_REQUEST_MSG`: Message to be displayed for a bad public link request (string)

Example of default values:

```
# Filesfolders app settings
FILESFOLDERS_MAX_UPLOAD_SIZE = env.int(
    'FILESFOLDERS_MAX_UPLOAD_SIZE', 10485760)
FILESFOLDERS_MAX_ARCHIVE_SIZE = env.int(
    'FILESFOLDERS_MAX_ARCHIVE_SIZE', 52428800)
FILESFOLDERS_SERVE_AS_ATTACHMENT = False
FILESFOLDERS_LINK_BAD_REQUEST_MSG = 'Invalid request'
```

URL Configuration

In the Django URL configuration file, add the following lines under `urlpatterns` to include `filesfolders` URLs in your site. The latter line is required by `db_file_storage` and should be obfuscated from actual users.

```
urlpatterns = [
    # ...
    url(r'^files/', include('filesfolders.urls')),
    url(r'^OBFUSCATED_STRING_HERE/', include('db_file_storage.urls')),
]
```

Migrate Database and Register Plugin

To migrate the Django database and register the `filesfolders` app plugin, run the following management command:

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

In addition to the database migration operation, you should see the following output:

```
Registering Plugin for filesfolders.plugins.ProjectAppPlugin
```

1.5.2 Filesfolders Usage

Usage instructions for the `filesfolders` app are detailed in this document.

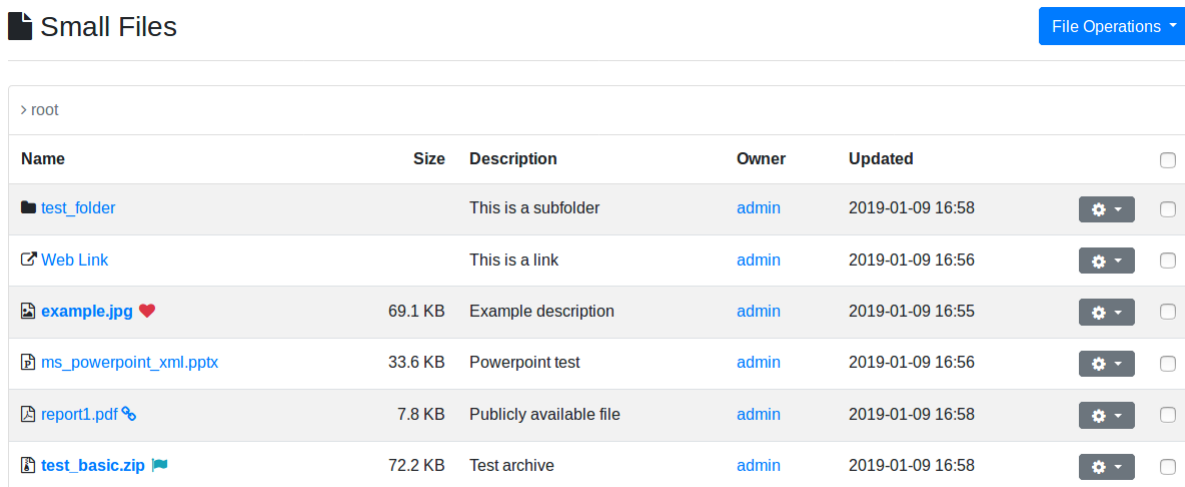
NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Filesfolders UI

You can browse and manage files in the app’s main view according to your permissions for each project. The “*File Operations*” menu is used to upload new files as well as add new folders or links. The menu also contains batch moving and deletion operations, for which items can be checked using the right hand side checkboxes.

Updating/deleting operations for single items can be accessed in the dropdown menus for each item. In the item create/update form, you can also *tag* items with a choice of icons and stylings to represent the item status.

When uploading a .zip archive, you may choose the “*Extract files from archive*” option to automatically extract archive files and folders into the filesfolders app. Note that overwriting of files is not currently allowed.



The screenshot shows the 'Small Files' view of the Filesfolders app. At the top right is a blue button labeled 'File Operations' with a dropdown arrow. Below the header is a table with columns: Name, Size, Description, Owner, Updated, and a checkbox. The table lists several items: a folder named 'test_folder', a 'Web Link', and five files: 'example.jpg', 'ms_powerpoint_xml.pptx', 'report1.pdf', and 'test_basic.zip'. Each row has a settings gear icon and a checkbox on the right.

Name	Size	Description	Owner	Updated	
> root					
test_folder		This is a subfolder	admin	2019-01-09 16:58	<input type="checkbox"/>
Web Link		This is a link	admin	2019-01-09 16:56	<input type="checkbox"/>
example.jpg	69.1 KB	Example description	admin	2019-01-09 16:55	<input type="checkbox"/>
ms_powerpoint_xml.pptx	33.6 KB	Powerpoint test	admin	2019-01-09 16:56	<input type="checkbox"/>
report1.pdf	7.8 KB	Publicly available file	admin	2019-01-09 16:58	<input type="checkbox"/>
test_basic.zip	72.2 KB	Test archive	admin	2019-01-09 16:58	<input type="checkbox"/>

Fig. 10: Filesfolders main view

App Settings

In the project create/update form, set the boolean setting `filesfolders.allow_public_links` true to allow providing public links to files, for people who can access the site but do not necessarily have a user account or project rights. Note that public link access still has to be granted for each file through its create/update form.

1.6 Userprofile App

The `userprofile` app is a site app which provides a user profile view for projectroles-compatible Django users and management of user specific settings.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

1.6.1 Installation

It is **strongly recommended** to install the `userprofile` app into your site when using `projectroles`, unless you require a specific user profile providing app of your own.

Warning: To install this app you **must** have the `django-sodar-core` package installed and the `projectroles` app integrated into your Django site. See the [projectroles integration document](#) for instructions.

Django Settings

The `userprofile` app is available for your Django site after installing `django-sodar-core`. Add the app into `THIRD_PARTY_APPS` as follows:

```
THIRD_PARTY_APPS = [
    # ...
    'userprofile.apps.UserprofileConfig',
]
```

URL Configuration

In the Django URL configuration file, add the following line under `urlpatterns` to include `userprofile` URLs in your site.

```
urlpatterns = [
    # ...
    url(r'^user/', include('userprofile.urls')),
]
```

Register Plugin

To register the app plugin, run the following management command:

```
$ ./manage.py syncplugins
```

You should see the following output:

```
Registering Plugin for userprofile.plugins.ProjectAppPlugin
```

1.6.2 Usage

After successful installation, the link for “User Profile” should be available in the user dropdown menu in the top-right corner of the website UI after you have logged in.

1.6.3 User Settings

User settings are configured in the `app_settings` dictionary in your project app plugins.

1.7 Siteinfo App

The `siteinfo` site app enables system administrators and developers to view site details and statistics gathered from project and backend apps.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

1.7.1 Basics

The app renders a site which displays information and statistics regarding the site and installed SODAR apps. Providing app statistics for siteinfo done via implementing the `get_statistics()` function in your app plugins. Currently, access to the app is limited to site administrators.

1.7.2 Installation

Warning: To install this app you **must** have the `django-sodar-core` package installed and the `projectroles` app integrated into your Django site. See the [projectroles integration document](#) for instructions.

Django Settings

The siteinfo app is available for your Django site after installing `django-sodar-core`. Add the app into `THIRD_PARTY_APPS` as follows:

```
THIRD_PARTY_APPS = [
    # ...
    'siteinfo.apps.SiteinfoConfig',
]
```

URL Configuration

In the Django URL configuration file, add the following line under `urlpatterns` to include siteinfo URLs in your site.

```
urlpatterns = [
    # ...
    url(r'^siteinfo/', include('siteinfo.urls')),
]
```

Migrate Database and Register Plugin

To migrate the Django database and register the siteinfo site app plugin, run the following management command:

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

In addition to the database migration operation, you should see the following output:

```
Registering Plugin for siteinfo.plugins.SiteAppPlugin
```

1.7.3 Usage

When logged in as a superuser, you can find the “Site Info” link in your user dropdown menu in the top right corner of the site.

This application is not available for users with a non-superuser status.

Providing App Statistics

In your project app or backend plugin, implement the `get_statistics()` function. It should return a dictionary containing, for each statistics item, a program friendly key and certain member fields:

- `label`: Human readable label for the statistics item.
- `value`: The value to be rendered
- `url`: The url to link to from the value for additional information (optional)
- `description`: Additional information (optional)

Example:

```
def get_statistics(self):
    return {
        'stat_id': {
            'label': 'Some statistic',
            'value': 9000,
            'url': reverse('home'),
            'description': 'More information here'
        }
    }
```

1.8 Sodarcache App

The `sodarcache` app provides a generic data caching functionality for a SODAR Core based site. This can be used to e.g. locally cache and aggregate data referring to external sources in order to speed up commonly repeated queries to databases other than the local Django PostgreSQL.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

1.8.1 Sodarcache Installation

This document provides instructions and guidelines for installing the `sodarcache` app to be used with your SODAR Core enabled Django site.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Warning: To install this app you **must** have the `django-sodar-core` package installed and the `projectroles` app integrated into your Django site. See the [projectroles integration document](#) for instructions.

Django Settings

The `sodarcache` app is available for your Django site after installing `django-sodar-core`. Add the app into `THIRD_PARTY_APPS` as follows:

```
THIRD_PARTY_APPS = [
    # ...
    'sodarcache.apps.SodarCacheConfig',
]
```

You also need to add the `sodarcache` backend plugin in enabled backend plugins.

```
ENABLED_BACKEND_PLUGINS = [
    # ...
    'sodar_cache',
]
```

URL Configuration

In the Django URL configuration file, add the following lines under `urlpatterns` to include `sodarcache` URLs in your site.

```
urlpatterns = [
    # ...
    url(r'^cache/', include('sodarcache.urls')),
]
```

Migrate Database and Register Plugin

To migrate the Django database and register the `sodarcache` app plugin, run the following management command:

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

In addition to the database migration operation, you should see the following output:

```
Registering Plugin for sodarcache.plugins.BackendPlugin
```

1.8.2 Sodar Cache Usage

Usage instructions for the `sodarcache` app are detailed in this document.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Backend API for Data Caching

The API for logging events is located in `sodarcache.api`. For the full API documentation, see [here](#).

Invoking the API

The API is accessed through a backend plugin. This means you can write calls to the API without any hard-coded imports and your code should work even if the `sodarcache` app has not been installed on the site.

Initialize the API using `projectroles.plugins.get_backend_api()` as follows:

```
from projectroles.plugins import get_backend_api
projectcache = get_backend_api('sodar_cache')

if projectcache:      # Only proceed if the backend was successfully initialized
    pass
```

Setting and getting Cache Items

Once you can access the `sodarcache` backend, you should set up the `update_cache()` function in the `ProjectAppPlugin` of the app with which you want to cache or aggregate data. The update process can be limited by two parameters: cached item name and project. If neither are specified, the function should update cached data for all known items within all projects.

```
def update_cache(self, name=None, project=None):
    """
    Update cached data for this app, limitable to item ID and/or project.

    :param project: Project object to limit update to (optional)
    :param name: Item name to limit update to (string, optional)
    """
    # TODO: Implement this in your app plugin
    return None
```

Updating a specific cache item within the `update_cache()` function (or elsewhere) should be done using `sodarcache.api.set_cache_item()`. A minimal example is as follows:

```
cache_item = projectcache.set_cache_item(
    project=project,          # Project object
    app_name=APP_NAME,       # Name of the current app
    user=request.user,       # The user triggering the cache update
    name='some_item',        # Cached item ID
    data_type='json',        # Data type ("json" currently supported)
    data={'key': 'val'},      # The actual data that should be cached
)
```

Note: The item ID in the `name` argument is not unique, but it is expected to be unique together with the `project` and `app_name` arguments.

Retrieve items with `sodarcache.get_cache_item()` or just check the time the item was last updated with `sodarcache.get_update_time()` like this:

```
projectcache.get_cache_item(  
    app_name='yourapp',  
    name='some_item',  
    project=project,  
    data_type='json'  
) # Returns a JsonCacheItem  
  
projectcache.get_update_time(  
    app_name='yourapp',  
    name='some_item',  
    project=project  
)
```

It is also possible to retrieve a Queryset with all cached items for a specific project with `sodarcache.get_project_cache()`

```
projectcache.get_project_cache(  
    project=project,          # Project object  
    data_type='json'         # must be 'json' for JsonCacheItem  
)
```

Using the Management commands

To create or update the data cache for all apps and projects, you can use a management command.

```
$ ./manage.py synccache
```

To limit the sync to a specific project, you can provide the `-p` or `--project` argument with the project UUID.

```
$ ./manage.py synccache -p e9701604-4ccc-426c-a67c-864c15aff6e2
```

Similarly, there is a command to delete all cached data:

```
$ ./manage.py deletecache
```

1.8.3 Sodarcache Backend API Documentation

This document contains API documentation for the backend plugin in the `sodarcache` app. Included are functionalities and classes intended to be used by other applications.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Backend API

The `SodarCacheAPI` class contains the Sodar Cache backend API. It should be initialized with `Projectroles.plugins.get_backend_api('sodar_cache')`.

class `sodarcache.api.SodarCacheAPI`

SodarCache backend API to be used by Django apps.

classmethod `delete_cache` (*app_name=None, project=None*)

Delete cache items. Optionallly limit to project and/or user.

Parameters

- **app_name** – Name of the app which sets the item (string)
- **project** – Project object (optional)

Returns Integer (deleted item count)

Raise `ValueError` if `app_name` is given but invalid

classmethod `get_cache_item` (*app_name, name, project=None*)

Return cached data by `app_name`, `name` (identifier) and optional `project`. Returns `None` if not found.

Parameters

- **name** – Item name (string)
- **app_name** – Name of the app which sets the item (string)
- **project** – Project object (optional)

Returns `JSONCacheItem` object

Raise `ValueError` if `app_name` is invalid

classmethod `get_project_cache` (*project, data_type='json'*)

Return all cached data for a project.

Parameters

- **project** – Project object
- **data_type** – String stating the data type of the cache items

Returns `QuerySet`

Raise `ValueError` if `data_type` is invalid

classmethod `get_update_time` (*app_name, name, project=None*)

Return the time of the last update of a cache object as seconds since epoch.

Parameters

- **name** – Item name (string)
- **app_name** – Name of the app which sets the item (string)
- **project** – Project object (optional)

Returns Float

classmethod `set_cache_item` (*app_name, name, data, data_type='json', project=None, user=None*)

Create or update and save a cache item.

Parameters

- **app_name** – Name of the app which sets the item (string)

- **name** – Item name (string)
- **data** – Item data (dict)
- **data_type** – String stating the data type of the cache items
- **project** – Project object (optional)
- **user** – User object to denote user triggering the update (optional)

Returns JSONCacheItem object

Raise ValueError if app_name is invalid

Raise ValueError if data_type is invalid

classmethod **update_cache** (*name=None, project=None, user=None*)

Update items by certain name within a project by calling implemented functions in project app plugins.

Parameters

- **name** – Item name to limit update to (string, optional)
- **project** – Project object to limit update to (optional)
- **user** – User object to denote user triggering the update (optional)

Models

class sodarcache.models.**BaseCacheItem** (*args, **kwargs)

Abstract class representing a cached item

app_name

App name

date_modified

DateTime of the update

name

Identifier for the item given by the data setting app

project

Project in which the item belongs (optional)

sodar_uuid

UUID for the item

user

User who updated the item (optional)

class sodarcache.models.**JSONCacheItem** (*args, **kwargs)

Class representing a cached item in JSON format

exception **DoesNotExist**

exception **MultipleObjectsReturned**

data

Cached data as JSON

1.9 Taskflow Backend

The `taskflowbackend` backend app is an optional add-on used if your site setup contains the separate **SODAR Taskflow** data transaction service.

If you have not set up a SODAR Taskflow service for any purpose, this backend is not needed and can be ignored.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

1.9.1 Basics

The `taskflowbackend` backend app is used in the main SODAR site to communicate with an external SODAR Taskflow service to manage large-scale data transactions. It has no views or database models and only provides an API for other apps to use.

Note: At the time of writing, SODAR Taskflow is in development and has not been made public.

1.9.2 Installation

Warning: To install this app you **must** have the `django-sodar-core` package installed and the `projectroles` app integrated into your Django site. See the [projectroles integration document](#) for instructions.

Django Settings

The `taskflowbackend` app is available for your Django site after installing `django-sodar-core`. Add the app into `THIRD_PARTY_APPS` as follows:

```
THIRD_PARTY_APPS = [
    # ...
    'taskflowbackend.apps.TaskflowbackendConfig',
]
```

Next add the backend to the list of enabled backend plugins:

```
ENABLED_BACKEND_PLUGINS = env.list('ENABLED_BACKEND_PLUGINS', None, [
    # ...
    'taskflow',
])
```

The following app settings **must** be included in order to use the backend. Note that the values for `TASKFLOW_TARGETS` depend on your SODAR Taskflow configuration.

```
# Taskflow backend settings
TASKFLOW_BACKEND_HOST = env.str('TASKFLOW_BACKEND_HOST', 'http://0.0.0.0')
TASKFLOW_BACKEND_PORT = env.int('TASKFLOW_BACKEND_PORT', 5005)
TASKFLOW_SODAR_SECRET = env.str('TASKFLOW_SODAR_SECRET', 'CHANGE ME!')
TASKFLOW_TARGETS = [
    'sodar',
    # ..
]
```

Register Plugin

To register the taskflowbackend plugin, run the following management command:

```
$ ./manage.py syncplugins
```

You should see the following output:

```
Registering Plugin for taskflowbackend.plugins.BackendPlugin
```

1.9.3 Usage

Once enabled, Retrieve the backend API class with the following in your Django app python code:

```
from projectroles.plugins import get_backend_api
taskflow = get_backend_api('taskflow')
```

See the docstrings of the API for more details.

To initiate sync of existing data with your SODAR Taskflow service, you can use the following management command:

```
./manage.py synctaskflow
```

1.9.4 API Documentation

The `TaskflowAPI` class contains the SODAR Taskflow backend API. It should be initialized using the `Projectroles.plugins.get_backend_api()` function.

class `taskflowbackend.api.TaskflowAPI`
SODAR Taskflow API to be used by Django apps

exception `CleanupException`
SODAR Taskflow cleanup exception

exception `FlowSubmitException`
SODAR Taskflow submission exception

cleanup ()
Send a cleanup command to SODAR Taskflow. Only allowed in test mode.

Returns Boolean

Raise `ImproperlyConfigured` if `TASKFLOW_TEST_MODE` is not set True

Raise `CleanupException` if SODAR Taskflow raises an error

get_error_msg (*flow_name*, *submit_info*)
Return a printable version of a SODAR Taskflow error message.

Parameters

- **flow_name** – Name of submitted flow
- **submit_info** – Returned information from SODAR Taskflow

Returns String

submit (*project_uuid*, *flow_name*, *flow_data*, *request=None*, *targets=['sodar']*, *request_mode='sync'*, *timeline_uuid=None*, *force_fail=False*, *sodar_url=None*)
Submit taskflow for SODAR project data modification.

Parameters

- **project_uuid** – UUID of the project (UUID object or string)
- **flow_name** – Name of flow to be executed (string)
- **flow_data** – Input data for flow execution (dict)
- **request** – Request object (optional)
- **targets** – Names of backends to sync with (list)
- **request_mode** – “sync” or “async”
- **timeline_uuid** – UUID of corresponding timeline event (optional)
- **force_fail** – Make flow fail on purpose (boolean, default False)
- **sodar_url** – URL of SODAR server (optional, for testing)

Returns Boolean**Raise** FlowSubmitException if submission fails**use_taskflow** (*project*)

Check whether taskflow use is allowed with a project.

Parameters **project** – Project object**Returns** Boolean

1.10 Timeline App

The `timeline` app enables the developer of a SODAR Core based site to log project related user events and link objects (both existing and deleted) to those events.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Unlike the standard Django object history accessible in the admin site, these events are not restricted to creation/modification of objects in the Django database, but can concern any user-triggered activity.

The events can also have multiple temporal status states in case of e.g. events requiring async requests.

The app provides front-end views to list project timeline events, as well as a backend API for saving desired activity as timeline events. For details on how to use these, see the [timeline usage documentation](#).

1.10.1 Timeline Installation

This document provides instructions and guidelines for installing the `timeline` app to be used with your SODAR Core enabled Django site.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Warning: To install this app you **must** have the `django-sodar-core` package installed and the `projectroles` app integrated into your Django site. See the [projectroles integration document](#) for instructions.

Django Settings

The timeline app is available for your Django site after installing `django-sodar-core`. Add the app into `THIRD_PARTY_APPS` as follows:

```
THIRD_PARTY_APPS = [  
    # ...  
    'timeline.apps.TimelineConfig',  
]
```

You also need to add the timeline backend plugin in enabled backend plugins.

```
ENABLED_BACKEND_PLUGINS = [  
    # ...  
    'timeline_backend',  
]
```

Optional Settings

To alter default timeline app settings, insert the following **optional** variables with values of your choosing:

```
# Timeline app settings  
TIMELINE_PAGINATION = 15      # Number of events to be shown on one page (int)
```

URL Configuration

In the Django URL configuration file, add the following line under `urlpatterns` to include timeline URLs in your site.

```
urlpatterns = [  
    # ...  
    url(r'^timeline/', include('timeline.urls')),  
]
```

Migrate Database and Register Plugin

To migrate the Django database and register the timeline app/backend plugins, run the following management command:

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

In addition to the database migration operation, you should see the following output:

```
Registering Plugin for timeline.plugins.ProjectAppPlugin  
Registering Plugin for timeline.plugins.BackendPlugin
```

1.10.2 Timeline Usage

Usage instructions for the `timeline` app are detailed in this document.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Timeline UI

You can browse events logged for each project by navigating to the project and selecting the “Timeline” app from the project sidebar.

By clicking on the time stamp for each event, you can see details of the event execution (in case of e.g. asynchronous events).

By clicking on the clock icon next to an object link in the event description, you can view the event history of that object. The link itself will take you to the relevant view for the object on your Django site.

Admin users are able to see certain “*classified*” level events hidden from normal users.

🕒 Project Timeline

Timestamp	App	Event	User	Description	Status
2019-01-09 16:58:27	filesfolders	folder_update	admin	update folder test_folder 🕒 (description)	OK
2019-01-09 16:58:12	filesfolders	file_update	admin	update file test_basic.zip 🕒 (flag)	OK
2019-01-09 16:58:01	filesfolders	file_create	admin	create file report1.pdf 🕒	OK
2019-01-09 16:57:35	projectroles	project_update	admin	update project (settings.filesfolders.allow_public_links)	OK
2019-01-09 16:57:08	filesfolders	file_update	admin	update file test_basic.zip 🕒 (description)	OK
2019-01-09 16:56:57	filesfolders	hyperlink_create	admin	create hyperlink Web Link 🕒	OK
2019-01-09 16:56:19	filesfolders	file_create	admin	create file ms_powerpoint_xml.pptx 🕒	OK
2019-01-09 16:56:04	filesfolders	file_create	admin	create file test_basic.zip 🕒	OK

Fig. 11: Timeline event list view

Backend API for Event Logging

The API for logging events is located in `timeline.api`. For the full API documentation, see [here](#).

Invoking the API

The API is accessed through a backend plugin. This means you can write calls to the API without any hard-coded imports and your code should work even if the timeline app has not been installed on the site.

The most common use case is to save events within the Class-Based Views of your Django site, but technically this can be done by any part of the code in your Django apps.

Initialize the API using `projectroles.plugins.get_backend_api()` as follows:

```
from projectroles.plugins import get_backend_api
timeline = get_backend_api('timeline_backend')

if timeline:      # Only proceed if the backend was successfully initialized
    pass          # Save your events here..
```

Adding an Event

Once you can access the timeline backend, add the event with `timeline.add_event()`. A minimal example is as follows:

```
tl_event = timeline.add_event(  
    project=project,          # Project object  
    app_name=APP_NAME,       # Name of the current app  
    user=request.user,        # The user triggering the activity being saved  
    event_name='some_event',  # You can define these yourself, not unique  
    description='Description') # Human readable description
```

Linking an Object

Say you want to link a Django model object to the event for tracking its history? In this example, let's say it's a SODAR Core compatible `User` model object `user_obj`.

Note: The given object **must** contain an `sodar_uuid` field with an auto-generated UUID. For more information, see the [project app development document](#).

Create the event as in the previous section, but add a label `target_user` in the description. The name of the label is arbitrary:

```
tl_event = timeline.add_event(  
    project=project,  
    app_name=APP_NAME,  
    user=request.user,  
    event_name='some_event',  
    description='Do something to {target_user}')
```

All you have to do is add an object reference to the created event:

```
obj_ref = tl_event.add_object(  
    obj=user_obj,  
    label='target_user',  
    name=user_obj.username)
```

The `name` field specifies which name the object will be referred to when displaying the event description to a user.

Defining Object References

The example before is all fine and good for a `User` object, but what about your own custom Django model?

When encountering an unknown object model from your app, timeline will call the `get_object_link()` function in the `ProjectAppPlugin` defined for your app. Make sure to implement it for all the relevant models in your app.

Displaying Object Links

In order to display object links with timeline history link included, you can use the `timeline.api.get_object_link()` function in your app's template tags.

Defining Status States

Note: If your Django apps only deal with normal synchronous requests, you don't need to pay attention to this functionality right now.

By default, `timeline.add_event()` treats events as synchronous and automatically saves them with the status of OK. However, in case of e.g. asynchronous requests, you can alter this by setting the `status_type` and (optionally) `status_desc` types upon creation.

```
tl_event = timeline.add_event(  
    project=project,  
    app_name=APP_NAME,  
    user=request.user,  
    event_name='some_event',  
    description='Description',  
    status_type='SUBMIT',  
    status_desc='Just submitted this')
```

After that, you can add new status states for the event using the object returned by `timeline.add_event()`:

```
tl_event.set_status('OK', 'Submission was successful!')
```

Currently supported status types are listed below, some only applicable to async events:

- OK: All OK, event successfully performed
- INFO: Used for events which do not change anything, e.g. viewing something within an app
- INIT: Initializing the event in progress
- SUBMIT: Event submitted asynchronously
- FAILED: Asynchronous event submission failed
- CANCEL: Event cancelled

Extra Data

Extra data can be added in the JSON format for both events and their status states with the `extra_data` and `status_extra_data` parameters.

Specifying a label `{extra-NAME}` in the event description will lead to a callback to `get_extra_data_link()` in the app plugin. To support this you need to make sure to implement the `get_extra_data_link()` function in your plugin.

Classified Events

To mark an event “classified”, that is, restricting its visibility to project owners and admins, set the `classified` argument to true when invoking `timeline.add_event()`.

Note: Multiple levels of classification may be introduced to the timeline event model in the future.

1.10.3 Timeline API Documentation

This document contains API documentation for the `timeline` app. Included are functionalities and classes intended to be used by other applications.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Backend API

The `TimelineAPI` class contains the Timeline backend API. It should be initialized using the `Projectroles.plugins.get_backend_api()` function.

class `timeline.api.TimelineAPI`

Timeline backend API to be used by Django apps.

static `add_event` (*project*, *app_name*, *user*, *event_name*, *description*, *classified=False*,
extra_data=None, *status_type=None*, *status_desc=None*, *status_extra_data=None*)

Create and save a timeline event.

Parameters

- **project** – Project object
- **app_name** – ID string of app from which event was invoked (NOTE: should correspond to member “name” in app plugin!)
- **user** – User invoking the event
- **event_name** – Event ID string (must match schema)
- **description** – Description of status change (may include {object label} references)
- **classified** – Whether event is classified (boolean, optional)
- **extra_data** – Additional event data (dict, optional)
- **status_type** – Initial status type (string, optional)
- **status_desc** – Initial status description (string, optional)
- **status_extra_data** – Extra data for initial status (dict, optional)

Returns `ProjectEvent` object

Raise `ValueError` if *app_name* or *status_type* is invalid

static `get_event_description` (*event*, *request=None*)

Return the description of a timeline event as HTML.

Parameters

- **event** – `ProjectEvent` object
- **request** – Request object (optional)

Returns String (contains HTML)

static `get_object_link` (*project_uuid*, *obj*)

Return an inline HTML icon link for a timeline event object history.

Parameters

- **project_uuid** – UUID of the related project

- **obj** – Django database object

Returns String (contains HTML)

static get_object_url (*project_uuid, obj*)

Return the URL for a timeline event object history.

Parameters

- **project_uuid** – UUID of the related project
- **obj** – Django database object

Returns String

static get_project_events (*project, classified=False*)

Return timeline events for a project.

Parameters

- **project** – Project object
- **classified** – Include classified (boolean)

Returns QuerySet

Models

class `timeline.models.ProjectEvent` (**args, **kwargs*)

Class representing a Project event

exception DoesNotExist

exception MultipleObjectsReturned

add_object (*obj, label, name, extra_data=None*)

Add object reference to an event.

Parameters

- **obj** – Django object to which we want to refer
- **label** – Label for the object in the event description (string)
- **name** – Name or title of the object (string)
- **extra_data** – Additional data related to object (dict, optional)

Returns ProjectEventObjectRef object

app

App from which the event was triggered

classified

Event is classified (only viewable by user levels specified in rules)

description

Description of status change (may include {object_name} references)

event_name

Event ID string

extra_data

Additional event data as JSON

get_current_status ()

Return the current event status

get_status_changes (*reverse=False*)

Return all status changes for the event

get_timestamp ()

Return the timestamp of current status

project

Project in which the event belongs

set_status (*status_type, status_desc=None, extra_data=None*)

Set event status.

Parameters

- **status_type** – Status type string (see EVENT_STATUS_TYPES)
- **status_desc** – Description string (optional)
- **extra_data** – Extra data for the status (dict, optional)

Returns ProjectEventStatus object

Raise TypeError if status_type is invalid

sodar_uuid

UUID for the event

user

User who initiated the event

class timeline.models.**ProjectEventManager**

Manager for custom table-level ProjectEvent queries

get_object_events (*project, object_model, object_uuid, order_by='-pk'*)

Return events which are linked to an object reference.

Parameters

- **project** – Project object
- **object_model** – Object model (string)
- **object_uuid** – sodar_uuid of the original object
- **order_by** – Ordering (default = pk descending)

Returns QuerySet

class timeline.models.**ProjectEventObjectRef** (**args, **kwargs*)

Class representing a reference to an object (existing or removed) related to a Timeline event status

exception DoesNotExist

exception MultipleObjectsReturned

event

Event to which the object belongs

extra_data

Additional data related to the object as JSON

label

Label for the object related to the event

name
Name or title of the object

object_model
Object model as string

object_uuid
Object SODAR UUID

class `timeline.models.ProjectEventStatus(*args, **kwargs)`
Class representing a Timeline event status

exception `DoesNotExist`

exception `MultipleObjectsReturned`

description
Description of status change (optional)

event
Event to which the status change belongs

extra_data
Additional status data as JSON

status_type
Type of the status change

timestamp
DateTime of the status change

1.11 Development

This document presents instructions and guidelines for developing apps compatible with the SODAR Core framework, as well as development of the SODAR Core package itself.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

1.11.1 General Development Guidelines

- Best practices from [Two Scoops](#) should be followed where applicable
- To maintain consistency, app packages should be named without delimiting characters, e.g. `projectroles` and `userprofile`
- It is recommended to add a “*Projectroles dependency*” comment when directly importing e.g. mixins or tags from the `projectroles` app
- **Hard-coded imports from apps *other than* `projectroles` should be avoided**
 - Use the plugin structure instead
 - See the `example_backend_app` for an example
- Using `Bootstrap 4` classes together with SODAR specific overrides and extensions provided in `projectroles.js` is recommended

1.11.2 Project App Development

This document details instructions and guidelines for developing **project apps** to be used with the SODAR Core framework. This also applies for modifying existing Django apps into project apps.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Hint: The package `example_project_app` in the `projectroles` repository provides a concrete minimal example of a working project app.

Project App Basics

Characteristics of a project app:

- Provides a functionality related to a project
- Is dynamically included in project views by `projectroles` using plugins
- Uses the project-based role and access control provided by `projectroles`
- Is included in `projectroles` search (optionally)
- Provides a dynamically included element (e.g. content overview) for the project details page
- Appears in the project menu sidebar in the default `projectroles` templates

Requirements for setting up a project app:

- Implement project relations and SODAR UUIDs in the app’s Django models
- Use provided mixins, keyword arguments and conventions in views
- Extend `projectroles` base templates in your templates
- Implement specific templates for dynamic inclusion by `Projectroles`
- Implement `plugins.py` with definitions and function implementations
- Implement `rules.py` with access rules

Fulfilling these requirements is detailed further in this document.

Prerequisites

This documentation assumes you have a Django project with the `projectroles` app set up (see the [projectroles integration document](#)). The instructions can be applied either to modify a previously existing app, or to set up a fresh app generated in the standard way with `./manage.py startapp`.

It is also assumed that apps are more or less created according to best practices defined by [Two Scoops](#), with the use of [Class-Based Views](#) being a requirement.

Models

In order to hook up your Django models into projects, there are two requirements: implementing a **project foreign key** and a **UUID field**.

Project Foreign Key

Add a `ForeignKey` field for the `projectroles.models.Project` model, either called `project` or accessible with a `get_project()` function implemented in your model.

If the project foreign key for your is **not** `project`, make sure to define a `get_project_filter_key()` function. It should return the name of the field to use as key for filtering your model by project.

Note: If your app contains a complex model structure with e.g. nested models using foreign keys, it's not necessary to add this to all your models, just the topmost one(s) used e.g. in URL kwargs.

Model UUID Field

To provide a unique identifier for objects in the SODAR context, add a `UUIDField` with the name of `sodar_uuid` into your model.

Note: Projectroles links to objects in URLs, links and forms using UUIDs instead of database private keys. This is strongly recommended for all Django models in apps using the projectroles framework.

Note: When updating an existing Django model with an existing database, the `sodar_uuid` field needs to be populated. See [instructions in Django documentation](#) on how to create the required migrations.

Model Example

Below is an example of a projectroles-compatible Django model:

```
import uuid
from django.db import models
from projectroles.models import Project

class SomeModel(models.Model):
    some_field = models.CharField(
        help_text='Your own field'
    )
    project = models.ForeignKey(
        Project,
        related_name='some_objects',
        help_text='Project in which this object belongs',
    )
    sodar_uuid = models.UUIDField(
        default=uuid.uuid4,
        unique=True,
        help_text='SomeModel SODAR UUID',
    )
```

Note: The `related_name` field is optional, but recommended as it provides an easy way to lookup objects of a certain type related to a project. For example the `project` foreign key in a model called `Document` could feature

e.g. `related_name='documents'`.

Rules File

Create a file `rules.py` in your app's directory. You should declare at least one basic rule for enabling a user to view the app data for the project. This can be named e.g. `{APP_NAME}.view_data`. Predicates for the rules can be found in `projectroles` and they can be extended within your app if needed.

```
import rules
from projectroles import rules as pr_rules

rules.add_perm(
    'example_project_app.view_data',
    pr_rules.is_project_owner
    | pr_rules.is_project_delegate
    | pr_rules.is_project_contributor
    | pr_rules.is_project_guest,
)
```

Hint: The `rules.is_superuser` predicate is often redundant, as permission checks are skipped for Django superusers. However, it can be handy if you e.g. want to define a rule allowing only superuser access for now, with the potential for adding other predicates later.

ProjectAppPlugin

Create a file `plugins.py` in your app's directory. In the file, declare a `ProjectAppPlugin` class implementing `projectroles.plugins.ProjectAppPluginPoint`. Within the class, implement member variables and functions as instructed in comments and docstrings.

```
from projectroles.plugins import ProjectAppPluginPoint
from .urls import urlpatterns

class ProjectAppPlugin(ProjectAppPluginPoint):
    """Plugin for registering app with Projectroles"""
    name = 'example_project_app'
    title = 'Example Project App'
    urls = urlpatterns
    # ...
```

The following variables and functions are **mandatory**:

- `name`: App name (**NOTE**: should correspond to the app package name or some functionality may not work as expected)
- `title`: Printable app title
- `urls`: Urlpatterns (usually imported from the app's `urls.py` file)
- `icon`: Font Awesome 4.7 icon name (without the `fa-*` prefix)
- `entry_point_url_id`: View ID for the app entry point (**NOTE**: The view **must** take the project `sodar_uuid` as a kwarg named `project`)
- `description`: Verbose description of app

- `app_permission`: Basic permission for viewing app data in project (see above)
- `search_enable`: Boolean for enabling/disabling app search
- `details_template`: Path to template to be included in the project details page, usually called `{APP_NAME}/_details_card.html`
- `details_title`: Title string to be displayed in the project details page for the app details template
- `plugin_ordering`: Number to define the ordering of the app on the project menu sidebar and the details page

Implementing the following is **optional**:

- `app_settings`: Implement if project or user specific settings for the app are needed. See the plugin point definition for an example.
- `search_types`: Implement if searching the data of the app is enabled
- `search_template`: Implement if searching the data of the app is enabled
- `project_list_columns`: Optional custom columns do be shown in the project list. See the plugin point definition for an example.
- `get_taskflow_sync_data()`: Applicable only if working with `sodar_taskflow` and `iRODS`
- `get_object_link()`: If Django models are associated with the app. Used e.g. by `django-sodar-timeline`.
- `search()`: Function called when searching for data related to the app if search is enabled
- `get_statistics()`: Return statistics for the siteinfo app. See details in [the siteinfo documentation](#).
- `get_project_list_value()`: A function which **must** be implemented if `project_list_columns` are defined, to retrieve a column cell value for a specific project.

Once you have implemented the `rules.py` and `plugins.py` files and added the app and its URL patterns to the Django site configuration, you can create the project app plugin in the Django database with the following command:

```
$ ./manage.py syncplugins
```

You should see the following output to ensure the plugin was successfully registered:

```
Registering Plugin for {APP_NAME}.plugins.ProjectAppPlugin
```

For info on how to implement the specific required views/templates, see the end of this document.

Views

Certain guidelines must be followed in developing Django web UI views for them to be successfully used with `projectroles`.

URL Keyword Arguments

In order to link a view to project and check user permissions using mixins, the URL keyword arguments **must** include an argument which matches *one of the following conditions*:

- Contains a kwarg `project` which corresponds to the `sodar_uuid` member value of a `projectroles.models.Project` object

- Contains a kwarg corresponding to the `sodar_uuid` of another Django model, which must contain a member field `project` which is a foreign key for a `Projectroles.models.Project` object. The kwarg **must** be named after the Django model of the referred object (in lowercase).
- Same as above, but the Django model provides a `get_project()` function which returns (you guessed it) a `Projectroles.models.Project` object.

Examples:

```
urlpatterns = [  
    # Direct reference to the Project model  
    url(  
        regex=r'^(?P<project>[0-9a-f-]+)$',  
        view=views.ProjectDetailView.as_view(),  
        name='detail',  
    ),  
    # RoleAssignment model has a "project" member which is also OK  
    url(  
        regex=r'^members/update/(?P<roleassignment>[0-9a-f-]+)$',  
        view=views.RoleAssignmentUpdateView.as_view(),  
        name='role_update',  
    ),  
]
```

Mixins

The `projectroles.views` module provides several useful mixins for augmenting your view classes to add projectroles functionality. These can be found in the `projectroles.views` module.

The most commonly used mixins:

- `LoggedInPermissionMixin`: Ensure correct redirection of users on no permissions
- `ProjectPermissionMixin`: Provides a `Project` object for permission checking based on URL kwargs
- `ProjectContextMixin`: Provides a `Project` object into the view context based on URL kwargs

See `example_project_app.views.ExampleView` for an example.

Templates

Template Structure

It is strongly recommended to extend `projectroles/project_base.html` in your project app templates. Just start your template with the following line:

```
{% extends 'projectroles/project_base.html' %}
```

The following **template blocks** are available for overriding or extending when applicable:

- `title`: Page title
- `css`: Custom CSS (extend with `{{ block.super }}`)
- `projectroles_extend`: Your app content goes here!
- `javascript`: Custom Javascript (extend with `{{ block.super }}`)
- `head_extend`: Optional block if you need to include additional content inside the HTML `<head>` element

Within the `projectroles_extend` block, it is recommended to use the following `div` classes, both extending the Bootstrap 4 `container-fluid` class:

- `sodar-subtitle-container`: Container for the page title
- `sodar-content-container`: Container for the actual content of your app

Rules

To control user access within a template, just do it as follows:

```
{% load rules %}
{% has_perm 'app.do_something' request.user project as can_do_something %}
```

This checks if the current user from the HTTP request has permission for `app.do_something` in the current project retrieved from the page context.

Template Tags

General purpose template tags are available in `projectroles/templatetags/projectroles_common_tags.py`. Include them to your template as follows:

```
{% load projectroles_common_tags %}
```

Example

Minimal example for a project app template:

```
{% extends 'projectroles/project_base.html' %}

{% load projectroles_common_tags %}
{% load rules %}

{% block title %}
    Page Title
{% endblock title %}

{% block head_extend %}
    {# OPTIONAL: extra content under <head> goes here #}
{% endblock head_extend %}

{% block css %}
    {{ block.super }}
    {# OPTIONAL: Extend or override CSS here #}
{% endblock css %}

{% block projectroles_extend %}

    {# Page subtitle #}
    <div class="container-fluid sodar-subtitle-container">
        <h3><i class="fa fa-rocket"></i> App and/or Page Title/h3>
    </div>

    {# App content #}
```

(continues on next page)

(continued from previous page)

```
<div class="container-fluid sodar-page-container">
  <p>Your app content goes here!</p>
</div>

{% endblock projectroles_extend %}

{% block javascript %}
  {{ block.super }}
  {# OPTIONAL: include additional Javascript here #}
{% endblock javascript %}
```

See `example_project_app/example.html` for a working and fully commented example of a minimal template.

Hint: If you include some controls on your `sodar-subtitle-container` class and want it to remain sticky on top of the page while scrolling, use `row` instead of `container-fluid` and add the `bg-white sticky-top` classes to the element.

General Guidelines for Views and Templates

General guidelines and hints for developing views and templates are discussed in this section.

Referring to Project Type

As of SODAR Core v0.4.3, it is possible to customize the display name for the project type from the default “project” or “category”. For more information, see [Projectroles Customization](#).

It is thus recommended that instead of hard coding “project” or “category” in your views or templates, use the `get_display_name()` function to refer to project type.

In templates, this can be achieved with a custom template tag. Example:

```
{% load projectroles_common_tags %}
{% get_display_name project.type title=True plural=False %}
```

In views and other Python code, the similar function can be accessed through `utils.py`:

```
from projectroles.utils import get_display_name
display_name = get_display_name(project.type, plural=False)
```

Hint: If not dealing with a `Project` object, you can provide the `PROJECT_TYPE_*` constant from `SODAR_CONSTANTS`. In templates, it’s most straightforward to use “CATEGORY” and “PROJECT”.

Forms

This section contains guidelines for implementing forms.

Custom User Selection Widget

A widget for autocomplete user selection in forms is available and can be build into any form.

First, the `UserAutocompleteWidget` needs to be imported from `projectroles/forms.py`.

```
from projectroles.forms import UserAutocompleteWidget
```

In your form's Meta class, assign the `UserAutocompleteWidget` as the widget of the user field:

```
class YourForm(forms.ModelForm):

    class Meta:
        model = YourModel
        fields ['user'] # ...
        widgets = {
            'user': UserAutocompleteWidget(
                url='projectroles:autocomplete_user',
                forward=['project'],
            )
        }
```

Some parameters have to be specified:

- `url`: The URL of the `UserAutocompleteAPIView` (or another custom API view)
- `forward`: Optional list with fields whose values will be forwarded to the view

If you wish to only display users who are members of a certain project, you need to include a field with the project's UUID in the form. This form can be hidden. This field's value needs to be forwarded to the autocomplete view (like in the code example above).

The alternative `UserAutocompleteExcludeMembersAPIView` view provides the opposite functionality: only users that are *not* project members are shown. That can be useful, for example, in a form to invite new members. To have the widget exclude project members, just change the URL parameter to the `UserAutocompleteExcludeMembersAPIView`'s URL (`projectroles:autocomplete_user_exclude`). In that same way, you can provide your custom view's URL to the widget to change its behaviour.

Also, as is required by SODAR, the user and the project fields need to point to SODAR UUIDs:

```
self.fields['project'].to_field_name = 'sodar_uuid'
self.fields['user'].to_field_name = 'sodar_uuid'
```

The following `django-autocomplete-light` and `select2` stylesheets and javascript files have to be added to the html template that includes the form.

```
{% block javascript %}
    {{ block.super }}
    <!-- DAL for autocomplete widgets -->
    <script type="text/javascript" src="{% static 'autocomplete_light/jquery.init.js' %}"
    <script type="text/javascript" src="{% static 'autocomplete_light/autocomplete.init.
    <script type="text/javascript" src="{% static 'autocomplete_light/vendor/select2/
    <script type="text/javascript" src="{% static 'autocomplete_light/select2.js' %}"></
    <script>
{% endblock javascript %}
```

(continues on next page)

(continued from previous page)

```
{% block css %}
  {{ block.super }}
  <!-- Select2 theme -->
  <link href="https://cdnjs.cloudflare.com/ajax/libs/select2/4.0.6-rc.0/css/select2.
  ↪min.css" rel="stylesheet" />
{% endblock css %}
```

When using the `RedirectWidget` or any other widget with custom javascript, include the corresponding js file instead of `autocomplete_light/select2.js`.

If you create your own custom user selection widget on the basis of the `UserAutocompleteWidget` take a look at the `RedirectWidget` as an example, or check out the `django-autocomplete-light` documentation for more information on how to customize your autocomplete-widget.

Specific Views and Templates

A few specific views/templates are expected to be implemented.

App Entry Point

As described in the Plugins chapter, an app entry point view is to be defined in the `ProjectAppPlugin`. This is **mandatory**.

The view **must** take a `project` URL kwarg which corresponds to a `Project.sodar_uuid`.

For an example, see `example_project_app.views.ExampleView` and the associated template.

Project Details Element

A sub-template to be included in the project details page (the project's "front page" provided by `projectroles`, where e.g. overview of app content is shown).

Traditionally these files are called `_details_card.html`, but you can name them as you wish and point to the related template in the `details_template` variable of your plugin.

It is expected to have the content in a `card-body` container:

```
<div class="card-body">
  {# Content goes here #}
</div>
```

Project Search Function and Template

If you want to implement search in your project app, you need to implement the `search()` function in your plugin as well as a template for displaying the results.

Hint: Implementing search *can* be complex. If you have access to the main SODAR repository, apps in that project might prove useful examples.

The search() Function

See the signature of `search()` in `projectroles.plugins.ProjectAppPluginPoint`. The arguments are as follows:

- **search_term**
 - Term to be searched for (string). Should be self-explanatory.
 - Multiple strings or separating multiple phrases with quotation marks not yet supported.
- **user**
 - User object for user initiating search
- **search_type**
 - The type of object to search for (string, optional)
 - Used to restrict search to specific types of objects
 - You can specify supported types in the plugin's `search_types` list.
 - Examples: `file`, `sample`..
- **keywords**
 - Special search keywords, e.g. “exact”
 - **NOTE:** Currently not implemented

Note: Within this function, you are expected to verify appropriate access of the searching user yourself!

The return data is a dictionary, which is split by groups in case your app can return multiple different lists for data. This is useful where e.g. the same type of HTML list isn't suitable for all returnable types. If only returning one type of data, you can just use e.g. `all` as your only category. Example of the result:

```
return {
    'all': {
        'title': 'List title',      # 1-N categories to be included
        'search_types': [],        # Title of the result list to be displayed
        'items': []               # Object types included in this category
    }                             # The actual objects returned
}
```

Search Template

Projectroles will provide your template context the `search_results` object, which corresponds to the result dict of the aforementioned function. There are also includes for formatting the results list, which you are encouraged to use.

Example of a simple results template, in case of a single `all` category:

```
{% if search_results.all.items|length > 0 %}

    {# Include standard search list header here #}
    {% include 'projectroles/_search_header.html' with search_title=search_results.all.
    title result_count=search_results.all.items|length %}
```

(continues on next page)

(continued from previous page)

```

    {# Set up a table with your results #}
    <table class="table table-striped sodar-card-table sodar-search-table" id="sodar-
    ff-search-table">
      <thead>
        <tr>
          <th>Name</th>
          <th>Some Other Field</th>
        </tr>
      </thead>
      <tbody>
        {% for item in search_results.all.items %}
          <tr>
            <td>
              <a href="#link_to_somewhere_in_your_app">{{ item.name }}</a>
            </td>
            <td>
              {{ item.some_other_field }}
            </td>
          </tr>
        {% endfor %}
      </tbody>
    </table>

    {# Include standard search list footer here #}
    {% include 'projectroles/_search_footer.html' %}

  {% endif %}

```

Tour Help

SODAR Core uses [Shepherd](#) to present an optional interactive tour for a rendered page. To enable the tour in your template, set it up inside the `javascript` template block. Within an inline javascript structure, set the `tourEnabled` variable to `true` and add steps according to the [Shepherd documentation](#).

Example:

```

{% block javascript %}
  {{ block.super }}

  {# Tour content #}
  <script type="text/javascript">
    tourEnabled = true;

    /* Normal step */
    tour.addStep('id_of_step', {
      title: 'Step Title',
      text: 'Description of the step',
      attachTo: '#some-element top',
      advanceOn: '.docs-link click',
      showCancelLink: true
    });

    /* Conditional step */
    if ($('#potentially-existing-element').length) {
      tour.addStep('id_of_another_step', {

```

(continues on next page)

(continued from previous page)

```

        title: 'Another Title',
        text: 'Another description here',
        attachTo: '.potentially-existing-element right',
        advanceOn: '.docs-link click',
        showCancelLink: true
    });
}

</script>
{% endblock javascript %}

```

Warning: Make sure you call `{{ block.super }}` at the start of the declared `javascript` block or you will overwrite the site's default Javascript setup!

API Views

API View usage will be explained in this chapter, currently under construction.

Warning: A unified SODAR API is currently under development and will be documented once stable. Current practices and base classes for API views are subject to change!

Ajax API Views

To set up Ajax API views for the UI, you can use the standard login and project permission mixins along with `APIPermissionMixin` together with any Django Rest Framework view class. Permissions can be managed as with normal Django views. Example with generic `APIView`:

```

from rest_framework.views import APIView
from projectroles.views import (
    LoginRequiredMixin,
    ProjectPermissionMixin,
    APIPermissionMixin,
)

class ExampleAjaxAPIView(
    LoginRequiredMixin,
    ProjectPermissionMixin,
    APIPermissionMixin,
    APIView,
):
    permission_required = 'projectroles.view_project'

    def get(self, request):
        # ...

```

TODO

- Naming conventions

- Examples of recurring template styles (e.g. forms)

1.11.3 Site App Development

This document details instructions and guidelines for developing **site apps** to be used with the SODAR Core framework.

It is recommended to read *Project App Development* before this document.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Site App Basics

Site apps are basically normal Django apps *not* hooked to SODAR projects. However, they provide a few nice features to be used in a SODAR-enabled Django site:

- Rules for accessing app data (similar to project apps but without the need for being associated with a project)
- Dynamic inclusion into the site and default templates via plugins
- The ability to show site-wide messages to users

Prerequisites

See *Project App Development*.

Models

No specific model implementation is required. However, it is strongly to refer to objects using `sodar_uuid` fields instead of the database private key.

Rules File

Generate a `rules.py` file similar to a project app. However, you should not use project predicates in this one. Example:

```
import rules
# Allow viewing data
rules.add_perm('{APP_NAME}.view_data', rules.is_authenticated)
```

SiteAppPlugin

Create a file `plugins.py` in your app’s directory. In the file, declare a `SiteAppPlugin` class implementing `projectroles.plugins.SiteAppPluginPoint`. Within the class, implement member variables and functions as instructed in comments and docstrings.

```
from projectroles.plugins import SiteAppPluginPoint
from .urls import urlpatterns

class SiteAppPlugin(SiteAppPluginPoint):
    """Plugin for registering a site-wide app"""
```

(continues on next page)

(continued from previous page)

```

name = 'example_site_app'
title = 'Example Site App'
urls = urlpatterns
# ...

```

The following variables and functions are **mandatory**:

- `name`: App name (ideally should correspond to the app package name)
- `title`: Printable app title
- `urls`: Urlpatterns (usually imported from the app's `urls.py` file)
- `icon`: Font Awesome 4.7 icon name (without the `fa-*` prefix)
- `entry_point_url_id`: View ID for the app entry point
- `description`: Verbose description of app
- `app_permission`: Basic permission for viewing app data in project (see above)

Implementing the following is **optional**:

- `get_messages()`: Implement if your site app needs to display site-wide messages for users.

Views

In your views, you can still use `projectroles` mixins which are *not* related to projects. Especially `LoggedInPermissionMixin` is useful to ensure users not allowed to access a view are properly redirected. Example:

```

from django.views.generic import TemplateView
from projectroles.views import LoggedInPermissionMixin

class ExampleView(LoginPermissionMixin, TemplateView):
    """Site app example view"""
    permission_required = 'example_site_app.view_data'
    template_name = 'example_site_app/example.html'

```

Note: The entry point URL is not expected to have any URL kwargs in the current implementation. If you intend to use a view which makes use of URL kwargs, you may need to modify it into also accepting a request without any parameters (e.g. displaying default content for the view).

Templates

It is recommended for you to extend `projectroles/base.html` and put your actual app content within the `projectroles` block. Example:

```

{# Projectroles dependency #}
{% extends 'projectroles/base.html' %}
{% load projectroles_common_tags %}

{% block title %}
    Example Site App Page Title
{% endblock title %}

```

(continues on next page)

(continued from previous page)

```
{% block projectroles %}

<div class="container sodar-subtitle-container">
  <h2><i class="fa fa-umbrella"></i> Example Site App</h2>
</div>

<div class="container-fluid sodar-page-container">
  <div class="alert alert-info">
    This is an example and the entry point for <code>example_site_app</code>.
  </div>
</div>

{% endblock projectroles %}
```

Site App Messages

The site app provides a way to display certain messages to users. For this, you need to implement `get_messages()` in the `SiteAppPlugin` class.

If you need to control e.g. which user should see the message or removal of a message after showing, you need to implement relevant logic in the function.

Example:

```
def get_messages(self, user=None):
    """
    Return a list of messages to be shown to users.
    :param user: User object (optional)
    :return: List of dicts or and empty list if no messages
    """
    return [{
        'content': 'Message content in here, can contain html',
        'color': 'info',          # Corresponds to bg-* in Bootstrap
        'dismissable': True       # False for non-dismissable
        'require_auth': True      # Only view for authorized users
    }]
```

1.11.4 Backend App Development

This document details instructions and guidelines for developing **backend apps** to be used with the SODAR Core framework.

It is recommended to read *Project App Development* before this document.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Backend App Basics

Backend apps are intended as apps used by other apps via their plugin, without requiring hard-coded imports. These may provide their own views for e.g. Ajax API functionality, but mostly they’re intended to be internal (hence the name).

Prerequisites

See *Project App Development*.

Models

No specific model implementation is required. However, it is strongly to refer to objects using `sodar_uuid` fields instead of the database private key.

BackendAppPlugin

The plugin is detected and retrieved using a `BackendAppPlugin`.

Declaring the Plugin

Create a file `plugins.py` in your app's directory. In the file, declare a `BackendAppPlugin` class implementing `projectroles.plugins.BackendPluginPoint`. Within the class, implement member variables and functions as instructed in comments and docstrings.

```
from projectroles.plugins import BackendPluginPoint
from .urls import urlpatterns

class BackendAppPlugin(BackendPluginPoint):
    """Plugin for registering a backend app"""
    name = 'example_backend_app'
    title = 'Example Backend App'
    urls = urlpatterns
    # ...
```

The following variables and functions are **mandatory**:

- `name`: App name (ideally should correspond to the app package name)
- `title`: Printable app title
- `icon`: Font Awesome 4.7 icon name (without the `fa-*` prefix)
- `description`: Verbose description of app
- `get_api()`: Function for retrieving the API class for the backend, to be implemented

Implementing the following is **optional**:

- `get_statistics()`: Return statistics for the siteinfo app. See details in *the siteinfo documentation*.

Hint: If you want to implement a backend API which is closely tied to a project app, there's no requirement to declare your backend as a separate Django app. You can just include the `BackendAppPlugin` in your app's `plugins.py` along with your `ProjectAppPlugin`. See the *timeline app* for an example of this.

Using the Plugin

To retrieve the API for the plugin, use the function `projectroles.plugins.get_backend_api()` as follows:

```
from projectroles.plugins import get_backend_api
example_api = get_backend_api('example_backend_app')

if example_api:      # Make sure the API is there, and only after that..
    pass             # ..do stuff with the API
```

1.11.5 SODAR Core Development

This document details instructions and guidelines for development of the SODAR Core package.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

Installation

Instructions on how to install a local development version of SODAR Core. Ubuntu 16.04 LTS (Xenial) is the supported OS at this time. System dependencies may vary for different OS versions or distributions.

System Installation

First you need to install OS dependencies, PostgreSQL 9.6 and Python3.6.

```
$ sudo utility/install_os_dependencies.sh
$ sudo utility/install_python.sh
$ sudo utility/install_postgres.sh
```

Database Setup

Create a PostgreSQL user and a database for your application. In the example, we use `sodar_core` for the database, user name and password. Make sure to give the user the permission to create further PostgreSQL databases (used for testing).

```
$ sudo su - postgres
$ psql
$ CREATE DATABASE sodar_core;
$ CREATE USER sodar_core WITH PASSWORD 'sodar_core';
$ GRANT ALL PRIVILEGES ON DATABASE sodar_core to sodar_core;
$ ALTER USER sodar_core CREATEDB;
$ \q
```

You have to add the credentials in the environment variable `DATABASE_URL`. For development it is recommended to place this variable in an `.env` file and set `DJANGO_READ_DOT_ENV_FILE=1` in your actual environment. See `config/settings/base.py` for more information.

```
$ export DATABASE_URL='postgres://sodar_core:sodar_core@127.0.0.1/sodar_core'
```

Project Setup

Clone the repository, setup and activate the virtual environment. Once in the environment, install Python requirements for the project:

```
$ git clone git+https://github.com/bihealth/sodar_core.git
$ cd sodar_core
$ pip install virtualenv
$ virtualenv -p python3.6 .venv
$ source .venv/bin/activate
$ utility/install_python_dependencies.sh
```

LDAP Setup (Optional)

If you will be using LDAP/AD auth on your site, make sure to also run:

```
$ sudo utility/install_ldap_dependencies.sh
$ pip install -r requirements/ldap.txt
```

Final Setup

Initialize the database (this will also synchronize django-plugins):

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

Create a Django superuser for the example_site:

```
$ ./manage.py createsuperuser
```

Now you should be able to run the server:

```
$ ./run.sh
```

Testing

To run unit tests, you have to install the headless Chrome driver (if not yet present on your system), followed by the Python test requirements:

```
$ sudo utility/install_chrome.sh
$ pip install -r requirements/test.txt
```

Now you can run all tests with the following script:

```
$ ./test.sh
```

If you want to only run a certain subset of tests, use e.g.:

```
$ ./test.sh projectroles.tests.test_views
```

For running tests with SODAR Taskflow (not currently publicly available), you can use the supplied shortcut script:

```
$ ./test_taskflow.sh
```

Contributing

SODAR Core is currently in active development in a private BIH repository. The public GitHub repository is primarily intended for publishing stable releases. Furthermore, the issue IDs within the code and documentation point to our private issue tracker unless otherwise mentioned.

1.12 Breaking Changes

This document details breaking changes from previous SODAR Core releases. It is recommended to review these notes whenever upgrading from an older SODAR Core version. For a complete list of changes in the current release, see the `CHANGELOG.rst` file.

NOTE: When viewing this document in GitLab critical content will by default be missing. Please click “display source” if you want to read this in GitLab.

1.12.1 v0.6.1 (2019-06-05)

App Settings Deprecation Protection Removed

The deprecation protection set up in the previous release has been removed. Project app plugins are now expected to declare `app_settings` in the format introduced in v0.6.0.

1.12.2 v0.6.0 (2019-05-10)

App Settings (Formerly Project Settings)

The former Project Settings module has been completely overhauled in this version and requires changes to your app plugins.

The `projectroles.project_settings` module has been renamed into `projectroles.app_settings`. Please update your dependencies accordingly.

Settings must now be defined in `app_settings`. The format is identical to the previous `project_settings` dictionary, except that a `scope` field is expected for each settings. Currently valid values are “PROJECT” and “USER”. It is recommended to use the related constants from `SODAR_CONSTANTS` instead of hard coded strings.

Example of settings:

```
#: Project and user settings
app_settings = {
    'project_bool_setting': {
        'scope': 'PROJECT',
        'type': 'BOOLEAN',
        'default': False,
        'description': 'Example project setting',
    },
    'user_str_setting': {
        'scope': 'USER',
        'type': 'STRING',
        'label': 'String example',
        'default': '',
        'description': 'Example user setting',
    },
}
```

(continues on next page)

(continued from previous page)

```

    },
}

```

Warning: Deprecation protection is place in this version for retrieving settings from `project_settings` if it has not been changed into `app_settings` in your project apps. This protection **will be removed** in the next SODAR Core release.

1.12.3 v0.5.1 (2019-04-16)

Site App Templates

Templates for **site apps** should extend `projectroles/base.html`. In earlier versions the documentation erroneously stated `projectroles/project_base.html` as the base template to use. Extending that document does work in this version as long as you override the given template blocks. However, it is not recommended and may break in the future.

Sodarcache App Changes

The following potentially breaking changes have been made to the sodarcache app.

App configuration naming has been changed to `sodarcache.apps.SodarcacheConfig`. Please update `config/settings/base.py` accordingly.

The field `user` has been made optional in models and the API.

An optional `user` argument has been added to `ProjectAppPlugin.update_cache()`. Correspondingly, the similar argument in `ProjectCacheAPI.set_cache_item()` has been made optional. Please update your plugin implementations and function calls accordingly.

The `updatecache` management command has been renamed to `synccache`.

Helper `get_app_names()` Fixed

The `projectroles.utils.get_app_names()` function will now return nested app names properly instead of omitting everything beyond the topmost module.

Default Admin Setting Deprecation Removed

The `PROJECTROLES_ADMIN_OWNER` setting no longer works. Use `PROJECTROLES_DEFAULT_ADMIN` instead.

1.12.4 v0.5.0 (2019-04-03)

Default Admin Setting Renamed

The setting `PROJECTROLES_ADMIN_OWNER` has been renamed into `PROJECTROLES_DEFAULT_ADMIN` to better reflect its uses. Please rename this settings variable on your site configuration to prevent issues.

Note: In this release, the old settings value is still accepted in remote project management to avoid sudden crashes. This deprecation will be removed in the next release.

Bootstrap 4.3.1 Upgrade

The Bootstrap and Popper dependencies have been updated to the latest versions. Please test your site to make sure this does not result in compatibility issues. The known issue of HTML content not showing in popovers has already been fixed in `projectroles.js`.

Default Templates Modified

The default templates `base_site.html` and `login.html` have been modified in this version. If you override them with your own altered versions, please review the difference and update your templates as appropriate.

1.12.5 v0.4.5 (2019-03-06)

System Prerequisites

The minimum version requirement for Django has been bumped to 1.11.20.

User Autocomplete Widget Support

Due to the use of autocomplete widgets for users, the following apps must be added into `THIRD_PARTY_APPS` in `config/settings/base.py`, regardless of whether you intend to use them in your own apps:

```
THIRD_PARTY_APPS = [
    # ...
    'dal',
    'dal_select2',
]
```

Project.get_delegate() Helper Renamed

As the limit for delegates per project is now arbitrary, the `Project.get_delegate()` helper function has been replaced by `Project.get_delegates()`. The new function returns a `QuerySet`.

Bootstrap 4 Crispy Forms Overrides Removed

Deprecated site-wide Bootstrap 4 theme overrides for `django-crispy-forms` were removed from the example site and are no longer supported. These workarounds were located in `{SITE_NAME}/templates/bootstrap4/`. Unless specifically required forms on your site, it is recommended to remove the files from your project.

Note: If you choose to keep the files or similar workarounds in your site, you are responsible of maintaining them and ensuring SODAR compatibility. Such site-wide template overrides are outside of the scope for SODAR Core components. Leaving the existing files in without maintenance may cause undesirable effects in the future.

Database File Upload Widget

Within SODAR Core apps, the only known issue caused by removal of the aforementioned Bootstrap 4 form overrides in the file upload widget of the `django-db-file-upload` package. If you are using the file upload package in your own SODAR apps and have removed the site-wide Crispy overrides, you can fix this particular widget by adding the following snippet into your form template. Make sure to replace `{FIELD_NAME}` with the name of your form field.

```
{% block css %}
{{ block.super }}
{# Workaround for django-db-file-storage Bootstrap4 issue (#164) #}
<style type="text/css">
    div#div_id_{FIELD_NAME} div p.invalid-feedback {
        display: block;
    }
</style>
{% endblock css %}
```

Alternatively, you can create a common override in your project-wide CSS file.

1.12.6 v0.4.4 (2019-02-19)

Textarea Height in Forms

Due to this feature breaking the layout of certain third party components, textarea height in forms is no longer adjusted automatically. An exception to this are Pagedown-specific markdown fields.

To adjust the height of a textarea field in your forms, the easiest way is to modify the widget of the related field in the `__init__()` function of your form as follows:

```
self.fields['field_name'].widget.attrs['rows'] = 4
```

1.12.7 v0.4.3 (2019-01-31)

SODAR Constants

`PROJECT_TYPE_CHOICES` has been removed from `SODAR_CONSTANTS`, as it can vary depending on implemented `DISPLAY_NAMES`. If needed, the currently applicable form structure can be imported from `projectroles.forms`.

1.12.8 v0.4.2 (2019-01-25)

System Prerequisites

The following minimum version requirements have been upgraded in this release:

- Django 1.11.18+
- Bootstrap 4.2.1
- JQuery 3.3.1
- Numerous required Python packages (see `requirements/*.txt`)

Please go through your site requirements and update dependencies accordingly. For project stability, it is still recommended to use exact version numbers for Python requirements in your SODAR Core based site.

If you are overriding the `projectroles/base_site.html` in your site, make sure to update Javascript and CSS includes accordingly.

Note: Even though the recommended Python version from Django 1.11.17+ is 3.7, we only support Python 3.6 for this release. The reason is that some dependencies still exhibit problems with the most recent Python release at the time of writing.

ProjectAccessMixin

The `_get_project()` function in `ProjectAccessMixin` has been renamed into `get_project()`. Arguments for the function are now optional and may be removed in a subsequent release: `self.request` and `self.kwargs` of the view class will be used if the arguments are not present.

Base API View

The base SODAR API view has been renamed from `BaseAPIView` into `SODARAPIBaseView`.

Taskflow Backend API

The `cleanup()` function in `TaskflowAPI` now correctly raises a `CleanupException` if SODAR Taskflow encounters an error upon calling its cleanup operation. This change should not affect normally running your site, as the function in question should only be called during Taskflow testing.

1.12.9 v0.4.1 (2019-01-11)

System Prerequisites

Changes in system requirements:

- **Ubuntu 16.04 Xenial** is the target OS version.
- **Python 3.6 or newer required:** 3.5 and older releases no longer supported.
- **PostgreSQL 9.6** is the recommended minimum version for the database.

Site Messages in Login Template

If your site overrides the default login template in `projectroles/login.html`, make sure your overridden version contains an include for `projectroles/_messages.html`. Following the SODAR Core template conventions, it should be placed as the first element under the `container-fluid` div in the content block. Otherwise, site app messages not requiring user authorization will not be visible on the login page. Example:

```
{% block content %}
<div class="container-fluid">
  {# Django messages / site app messages #}
  {% include 'projectroles/_messages.html' %}
  {# ... #}
```

(continues on next page)

(continued from previous page)

```
</div>
{% endblock content %}
```

1.12.10 v0.4.0 (2018-12-19)

List Button Classes in Templates

Custom small button and dropdown classes for including buttons within tables and lists have been modified. The naming has also been unified. The following classes should now be used:

- Button group: `sodar-list-btn-group` (formerly `sodar-edit-button-group`)
- Button: `sodar-list-btn`
- Dropdown: `sodar-list-dropdown` (formerly `sodar-edit-dropdown`)

See projectroles templates for examples.

Warning: The standard bootstrap class `btn-sm` should **not** be used with these custom classes!

SODAR Taskflow v0.3.1 Required

If using SODAR Taskflow, this release requires release v0.3.1 or higher due to mandatory support of the `TASKFLOW_SODAR_SECRET` setting.

Taskflow Secret String

If you are using the `taskflow` backend app, you **must** set the value of `TASKFLOW_SODAR_SECRET` in your Django settings. Note that this must match the similarly named setting in your SODAR Taskflow instance!

1.12.11 v0.3.0 (2018-10-26)

Remote Site Setup

For specifying the role of your site in remote project metadata synchronization, you will need to add two new settings to your Django site configuration:

The `PROJECTROLES_SITE_MODE` setting sets the role of your site in remote project sync and it is **mandatory**. Accepted values are `SOURCE` and `TARGET`. For deployment, it is recommended to fetch this setting from environment variables.

If your site is set in `TARGET` mode, the boolean setting `PROJECTROLES_TARGET_CREATE` must also be included to control whether creation of local projects is allowed. If your site is in `SOURCE` mode, this setting can be included but will have no effect.

Furthermore, if your site is in `TARGET` mode you must include the `PROJECTROLES_ADMIN_OWNER` setting, which must point to an existing local superuser account on your site.

Example for a `SOURCE` site:

```
# Projectroles app settings
PROJECTROLES_SITE_MODE = env.str('PROJECTROLES_SITE_MODE', 'SOURCE')
```

Example for a TARGET site:

```
# Projectroles app settings
PROJECTROLES_SITE_MODE = env.str('PROJECTROLES_SITE_MODE', 'TARGET')
PROJECTROLES_TARGET_CREATE = env.bool('PROJECTROLES_TARGET_CREATE', True)
PROJECTROLES_ADMIN_OWNER = env.str('PROJECTROLES_ADMIN_OWNER', 'admin')
```

General API Settings

Add the following lines to your configuration to enable the general API settings:

```
SODAR_API_DEFAULT_VERSION = '0.1'
SODAR_API_MEDIA_TYPE = 'application/vnd.bihealth.sodar+json'
```

DataTables Includes

Includes for the DataTables Javascript library are no longer included in templates by default. If you want to use DataTables, include the required CSS and Javascript in relevant templates. See the `projectroles/search.html` template for an example.

CHAPTER 2

Indices and tables

- `genindex`
- `modindex`
- `search`

p

`projectroles.models`, [31](#)
`projectroles.plugins`, [28](#)
`projectroles.templatetags.projectroles_common_tags`,
 [37](#)
`projectroles.utils`, [39](#)

s

`sodarcache.models`, [52](#)

t

`timeline.models`, [61](#)

A

active (*projectroles.models.ProjectInvite* attribute), 33
 add_event() (*timeline.api.TimelineAPI* static method), 60
 add_object() (*timeline.models.ProjectEvent* method), 61
 app (*timeline.models.ProjectEvent* attribute), 61
 app_name (*sodarcache.models.BaseCacheItem* attribute), 52
 app_permission (*projectroles.plugins.RemoteSiteAppPlugin* attribute), 30
 app_plugin (*projectroles.models.AppSetting* attribute), 31
 AppSetting (class in *projectroles.models*), 31
 AppSetting.DoesNotExist, 31
 AppSetting.MultipleObjectsReturned, 31
 AppSettingAPI (class in *projectroles.app_settings*), 36
 AppSettingManager (class in *projectroles.models*), 32
 assign_user_group() (in module *projectroles.models*), 36

B

BackendPluginPoint (class in *projectroles.plugins*), 28
 BaseCacheItem (class in *sodarcache.models*), 52
 build_invite_url() (in module *projectroles.utils*), 39
 build_secret() (in module *projectroles.utils*), 39

C

change_plugin_status() (in module *projectroles.plugins*), 30
 check_backend() (in module *projectroles.templatetags.projectroles_common_tags*), 37

classified (*timeline.models.ProjectEvent* attribute), 61
 cleanup() (*taskflowbackend.api.TaskflowAPI* method), 54
 core_version() (in module *projectroles.templatetags.projectroles_common_tags*), 37

D

data (*sodarcache.models.JSONCacheItem* attribute), 52
 date_access (*projectroles.models.RemoteProject* attribute), 34
 date_created (*projectroles.models.ProjectInvite* attribute), 33
 date_expire (*projectroles.models.ProjectInvite* attribute), 33
 date_modified (*sodarcache.models.BaseCacheItem* attribute), 52
 delete_cache() (*sodarcache.api.SodarCacheAPI* class method), 51
 description (*projectroles.models.Project* attribute), 32
 description (*projectroles.models.RemoteSite* attribute), 34
 description (*projectroles.models.Role* attribute), 35
 description (*projectroles.plugins.RemoteSiteAppPlugin* attribute), 30
 description (*timeline.models.ProjectEvent* attribute), 61
 description (*timeline.models.ProjectEventStatus* attribute), 63

E

email (*projectroles.models.ProjectInvite* attribute), 33
 entry_point_url_id (*projectroles.plugins.RemoteSiteAppPlugin* attribute), 30
 event (*timeline.models.ProjectEventObjectRef* attribute), 62

`event` (*timeline.models.ProjectEventStatus* attribute), 63

`event_name` (*timeline.models.ProjectEvent* attribute), 61

`extra_data` (*timeline.models.ProjectEvent* attribute), 61

`extra_data` (*timeline.models.ProjectEventObjectRef* attribute), 62

`extra_data` (*timeline.models.ProjectEventStatus* attribute), 63

F

`find()` (*projectroles.models.ProjectManager* method), 34

`force_wrap()` (in module *projectroles.templatetags.projectroles_common_tags*), 37

G

`get_access_date()` (*projectroles.models.RemoteSite* method), 34

`get_active_plugins()` (in module *projectroles.plugins*), 30

`get_all_defaults()` (*projectroles.app_settings.AppSettingAPI* class method), 36

`get_all_settings()` (*projectroles.app_settings.AppSettingAPI* class method), 36

`get_api()` (*projectroles.plugins.BackendPluginPoint* method), 28

`get_app_names()` (in module *projectroles.utils*), 39

`get_app_plugin()` (in module *projectroles.plugins*), 31

`get_app_setting()` (*projectroles.app_settings.AppSettingAPI* class method), 36

`get_assignment()` (*projectroles.models.RoleAssignmentManager* method), 35

`get_backend_api()` (in module *projectroles.plugins*), 31

`get_cache_item()` (*sodarcache.api.SodarCacheAPI* class method), 51

`get_children()` (*projectroles.models.Project* method), 32

`get_class()` (in module *projectroles.templatetags.projectroles_common_tags*), 38

`get_current_status()` (*timeline.models.ProjectEvent* method), 61

`get_default_setting()` (*projectroles.app_settings.AppSettingAPI* class method), 36

`get_delegates()` (*projectroles.models.Project* method), 32

`get_depth()` (*projectroles.models.Project* method), 32

`get_display_name()` (in module *projectroles.templatetags.projectroles_common_tags*), 38

`get_display_name()` (in module *projectroles.utils*), 39

`get_error_msg()` (*taskflowbackend.api.TaskflowAPI* method), 54

`get_event_description()` (*timeline.api.TimelineAPI* static method), 60

`get_expiry_date()` (in module *projectroles.utils*), 39

`get_extra_data_link()` (*projectroles.plugins.ProjectAppPluginPoint* method), 29

`get_full_name()` (*projectroles.models.SODARUser* method), 36

`get_full_title()` (*projectroles.models.Project* method), 32

`get_full_url()` (in module *projectroles.templatetags.projectroles_common_tags*), 38

`get_history_dropdown()` (in module *projectroles.templatetags.projectroles_common_tags*), 38

`get_info_link()` (in module *projectroles.templatetags.projectroles_common_tags*), 38

`get_members()` (*projectroles.models.Project* method), 32

`get_messages()` (*projectroles.plugins.SiteAppPluginPoint* method), 30

`get_object()` (*projectroles.plugins.ProjectAppPluginPoint* method), 29

`get_object_events()` (*timeline.models.ProjectEventManager* method), 62

`get_object_link()` (*projectroles.plugins.ProjectAppPluginPoint* method), 29

`get_object_link()` (*timeline.api.TimelineAPI* static method), 60

`get_object_url()` (*timeline.api.TimelineAPI* static method), 61

`get_owner()` (*projectroles.models.Project* method), 32

`get_parents()` (*projectroles.models.Project* method), 32

`get_project()` (*projectroles.models.RemoteProject*

method), 34

get_project_by_uuid() (in module *projectroles.templatetags.projectroles_common_tags*), 38

get_project_cache() (*sodarcache.api.SodarCacheAPI* class method), 51

get_project_events() (*timeline.api.TimelineAPI* static method), 61

get_project_link() (in module *projectroles.templatetags.projectroles_common_tags*), 38

get_project_list_value() (*projectroles.plugins.ProjectAppPluginPoint* method), 29

get_project_title_html() (in module *projectroles.templatetags.projectroles_common_tags*), 38

get_remote_icon() (in module *projectroles.templatetags.projectroles_common_tags*), 38

get_setting() (in module *projectroles.templatetags.projectroles_common_tags*), 38

get_setting_defs() (*projectroles.app_settings.AppSettingAPI* class method), 37

get_setting_value() (*projectroles.models.AppSettingManager* method), 32

get_source_site() (*projectroles.models.Project* method), 32

get_statistics() (*projectroles.plugins.BackendPluginPoint* method), 28

get_statistics() (*projectroles.plugins.ProjectAppPluginPoint* method), 29

get_status_changes() (*timeline.models.ProjectEvent* method), 62

get_taskflow_sync_data() (*projectroles.plugins.ProjectAppPluginPoint* method), 29

get_timestamp() (*timeline.models.ProjectEvent* method), 62

get_update_time() (*sodarcache.api.SodarCacheAPI* class method), 51

get_url() (*projectroles.models.RemoteSite* method), 35

get_user_by_username() (in module *projectroles.templatetags.projectroles_common_tags*), 38

get_user_display_name() (in module *projectroles.utils*), 39

get_user_html() (in module *projectroles.templatetags.projectroles_common_tags*), 38

get_value() (*projectroles.models.AppSetting* method), 31

H

handle_ldap_login() (in module *projectroles.models*), 36

has_role() (*projectroles.models.Project* method), 32

highlight_search_term() (in module *projectroles.templatetags.projectroles_common_tags*), 38

I

icon (*projectroles.plugins.RemoteSiteAppPlugin* attribute), 30

is_remote() (*projectroles.models.Project* method), 33

issuer (*projectroles.models.ProjectInvite* attribute), 33

J

JSONCacheItem (class in *sodarcache.models*), 52

JSONCacheItem.DoesNotExist, 52

JSONCacheItem.MultipleObjectsReturned, 52

L

label (*timeline.models.ProjectEventObjectRef* attribute), 62

level (*projectroles.models.RemoteProject* attribute), 34

M

message (*projectroles.models.ProjectInvite* attribute), 33

mode (*projectroles.models.RemoteSite* attribute), 35

N

name (*projectroles.models.AppSetting* attribute), 31

name (*projectroles.models.ProjectUserTag* attribute), 34

name (*projectroles.models.RemoteSite* attribute), 35

name (*projectroles.models.Role* attribute), 35

name (*projectroles.plugins.RemoteSiteAppPlugin* attribute), 30

name (*sodarcache.models.BaseCacheItem* attribute), 52

name (*timeline.models.ProjectEventObjectRef* attribute), 62

O

object_model (*timeline.models.ProjectEventObjectRef* attribute), 63

`object_uuid` (*timeline.models.ProjectEventObjectRef* attribute), 63

P

`parent` (*projectroles.models.Project* attribute), 33

`Project` (class in *projectroles.models*), 32

`project` (*projectroles.models.AppSetting* attribute), 31

`project` (*projectroles.models.ProjectInvite* attribute), 33

`project` (*projectroles.models.ProjectUserTag* attribute), 34

`project` (*projectroles.models.RemoteProject* attribute), 34

`project` (*projectroles.models.RoleAssignment* attribute), 35

`project` (*sodarcache.models.BaseCacheItem* attribute), 52

`project` (*timeline.models.ProjectEvent* attribute), 62

`Project.DoesNotExist`, 32

`Project.MultipleObjectsReturned`, 32

`project_uuid` (*projectroles.models.RemoteProject* attribute), 34

`ProjectAppPluginPoint` (class in *projectroles.plugins*), 29

`ProjectEvent` (class in *timeline.models*), 61

`ProjectEvent.DoesNotExist`, 61

`ProjectEvent.MultipleObjectsReturned`, 61

`ProjectEventManager` (class in *timeline.models*), 62

`ProjectEventObjectRef` (class in *timeline.models*), 62

`ProjectEventObjectRef.DoesNotExist`, 62

`ProjectEventObjectRef.MultipleObjectsReturned`, 62

`ProjectEventStatus` (class in *timeline.models*), 63

`ProjectEventStatus.DoesNotExist`, 63

`ProjectEventStatus.MultipleObjectsReturned`, 63

`ProjectInvite` (class in *projectroles.models*), 33

`ProjectInvite.DoesNotExist`, 33

`ProjectInvite.MultipleObjectsReturned`, 33

`ProjectManager` (class in *projectroles.models*), 33

projectroles.models (module), 31

projectroles.plugins (module), 28

projectroles.templatetags.projectroles_common_tags (module), 37

projectroles.utils (module), 39

`ProjectUserTag` (class in *projectroles.models*), 34

`ProjectUserTag.DoesNotExist`, 34

`ProjectUserTag.MultipleObjectsReturned`, 34

R

`readme` (*projectroles.models.Project* attribute), 33

`RemoteProject` (class in *projectroles.models*), 34

`RemoteProject.DoesNotExist`, 34

`RemoteProject.MultipleObjectsReturned`, 34

`RemoteSite` (class in *projectroles.models*), 34

`RemoteSite.DoesNotExist`, 34

`RemoteSite.MultipleObjectsReturned`, 34

`RemoteSiteAppPlugin` (class in *projectroles.plugins*), 30

`render_markdown()` (in module *projectroles.templatetags.projectroles_common_tags*), 38

`Role` (class in *projectroles.models*), 35

`role` (*projectroles.models.ProjectInvite* attribute), 33

`role` (*projectroles.models.RoleAssignment* attribute), 35

`Role.DoesNotExist`, 35

`Role.MultipleObjectsReturned`, 35

`RoleAssignment` (class in *projectroles.models*), 35

`RoleAssignment.DoesNotExist`, 35

`RoleAssignment.MultipleObjectsReturned`, 35

`RoleAssignmentManager` (class in *projectroles.models*), 35

S

`save()` (*projectroles.models.AppSetting* method), 31

`save()` (*projectroles.models.Project* method), 33

`save()` (*projectroles.models.RemoteSite* method), 35

`save()` (*projectroles.models.RoleAssignment* method), 35

`search()` (*projectroles.plugins.ProjectAppPluginPoint* method), 29

`secret` (*projectroles.models.ProjectInvite* attribute), 33

`secret` (*projectroles.models.RemoteSite* attribute), 35

`set_app_setting()` (*projectroles.app_settings.AppSettingAPI* class method), 37

`set_cache_item()` (*sodarcache.api.SodarCacheAPI* class method), 51

`set_status()` (*timeline.models.ProjectEvent* method), 62

`set_user_group()` (in module *projectroles.utils*), 39

`site` (*projectroles.models.RemoteProject* attribute), 34

`site_version()` (in module *projectroles.templatetags.projectroles_common_tags*), 38

`SiteAppPluginPoint` (class in *projectroles.plugins*), 30

`sodar_uuid` (*projectroles.models.AppSetting* attribute), 31

`sodar_uuid` (*projectroles.models.Project* attribute), 33

[sodar_uuid \(projectroles.models.ProjectInvite attribute\), 33](#)
[sodar_uuid \(projectroles.models.ProjectUserTag attribute\), 34](#)
[sodar_uuid \(projectroles.models.RemoteProject attribute\), 34](#)
[sodar_uuid \(projectroles.models.RemoteSite attribute\), 35](#)
[sodar_uuid \(projectroles.models.RoleAssignment attribute\), 35](#)
[sodar_uuid \(projectroles.models.SODARUser attribute\), 36](#)
[sodar_uuid \(sodarcache.models.BaseCacheItem attribute\), 52](#)
[sodar_uuid \(timeline.models.ProjectEvent attribute\), 62](#)
[sodarcache.models \(module\), 52](#)
[SodarCacheAPI \(class in sodarcache.api\), 51](#)
[SODARUser \(class in projectroles.models\), 35](#)
[static_file_exists\(\) \(in module projectroles.templatetags.projectroles_common_tags\), 38](#)
[status_type \(timeline.models.ProjectEventStatus attribute\), 63](#)
[submit\(\) \(taskflowbackend.api.TaskflowAPI method\), 54](#)
[submit_status \(projectroles.models.Project attribute\), 33](#)

T

[TaskflowAPI \(class in taskflowbackend.api\), 54](#)
[TaskflowAPI.CleanupException, 54](#)
[TaskflowAPI.FlowSubmitException, 54](#)
[template_exists\(\) \(in module projectroles.templatetags.projectroles_common_tags\), 38](#)
[timeline.models \(module\), 61](#)
[TimelineAPI \(class in timeline.api\), 60](#)
[timestamp \(timeline.models.ProjectEventStatus attribute\), 63](#)
[title \(projectroles.models.Project attribute\), 33](#)
[title \(projectroles.plugins.RemoteSiteAppPlugin attribute\), 30](#)
[type \(projectroles.models.AppSetting attribute\), 31](#)
[type \(projectroles.models.Project attribute\), 33](#)

U

[update_cache\(\) \(projectroles.plugins.ProjectAppPluginPoint method\), 30](#)
[update_cache\(\) \(sodarcache.api.SodarCacheAPI class method\), 52](#)
[url \(projectroles.models.RemoteSite attribute\), 35](#)

V

[validate_setting\(\) \(projectroles.app_settings.AppSettingAPI class method\), 37](#)
[value \(projectroles.models.AppSetting attribute\), 32](#)