SODAR Core Documentation

Release 0.9.0

Mikko Nieminen

Feb 03, 2021

OVERVIEW GETTING STARTED

SODAR Core is a framework for Django web application development.

It was conceived to facilitate the creation of scientific data management and analysis web applications (but can be useful in other contexts as well). In that it is similar to the CMS or ecommerce frameworks that you can find Awesome Django List but you will find the components/libraries provided in SODAR Core are more generic and in this reflecting the broader range of applications that we target.

CHAPTER

ONE

HOW TO READ THIS MANUAL?

There is two ways:

Front to Back If you have the time and patience, reading the whole manual will teach you everything.

Jump Around (recommended) Start with *For the Impatient* and/or *User Stories*, skim over the summary of each app, and explore what interests you most.

CHAPTER

TWO

WHAT SODAR CORE IS AND WHAT IT IS NOT

SODAR Core

- is Django-based and you will need some knowledge about Django programming in Python for it to be useful,
- provides you with libraries for developing your own applications.

SODAR Core

- is NOT a ready-made web application,
- is NOT for entry-level Python programmers (you will need intermediate Django knowledge; you probably do not want to base your first Django web application on SODAR Core).

CHAPTER

THREE

WHAT'S INSIDE SODAR CORE?

The full list of apps are shown in the table of contents (on the left if you are reading the HTML version of this documentation) and here are some highlights:

- 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

CHAPTER

FOUR

WHAT'S INSIDE THIS DOCUMENTATION?

Overview & Getting Started This part aims at getting you an birds-eye view of SODAR Core and its usage.

SODAR Core Apps This part documents each Django app that ships with SODAR. As a reminder, in Django development, *apps* are re-useable modules with code for supporting a certain use case.

Project Info This part of the documentation provides meta information about the project and the full changelog.

CHAPTER

FIVE

WHAT'S NOT INSIDE THIS DOCUMENTATION?

You should know the following before this documentation is useful to you:

Python Programming There's tons of documentation on the internet but the official Python documentation is a good starting point as any.

Django Development For learning about Django, head over to the excellent documentation of the Django Project.

HTML / Javascript / CSS / Bootstrap 4 Together with Django, SODAR Core provides a framework to plug in your own HTML and related front-end code. We assume that you have web development experience and in particular know your way around Bootstrap 4.

We're using the Bootstrap 4 CSS framework and you can learn about it in many places including the official documentation

Note: You can find the official version of this documentation at readthedocs.io. If you view these files on GitHub, beware that their renderer does not render the ReStructuredText files correctly and content may be missing.

5.1 SODAR Core Overview and Example Use Case

This document presents an overview of the SODAR Core package along with an example use case.

SODAR Core is a relatively complex system and we have created a *Glossary* to help you with keeping track of the terminology.

5.1.1 SODAR Core Overview

The SODAR Core package provides a suite of *Django apps* to be installed on a *Django-based web site*. The main app in the package, `projectroles`, provides core project access, content management framework and default UI templates for other apps on the site. Those apps must implement or use specific parts of the projectroles app to enable desired SODAR Core functionality.

Apps in a SODAR Core based site are separated into *project*, *site* and *backend* apps, depending on their scope and purpose. The SODAR Core package includes optional general purpose apps of each type, which the user may enable on their site if needed. These apps all depend on projectroles. More on the general purpose apps can be found in the *Getting Started* document.

To build their own web based system with SODAR Core, the user will develop required functionality and UIs as one or more Django apps, using and extending functionalities offered by the projectroles app and optional backend apps. This allows integration of the app into the project access management, standardized layout and other features such as advanced logging. Furthermore, the projectroles app will call certain functions implemented in the user's apps to

dynamically include app and project content in Django views. In addition to developing new Django apps, existing apps can be easily modified to gain access to SODAR Core features.

5.1.2 Example Use Case

In a typical scenario for SODAR Core use, a research organization wants to develop a user friendly system for accessing, browsing and/or manipulating research data. The data may belong to several different projects, with different research groups or scientists working on it. This data may be confidential in nature, so access control is required, preferably using the organization's existing LDAP/AD servers. The organization wants to provide a web-based GUI as well as programmatic API views.

Site Setup

First the developer will *set up a SODAR Core based Django site*. This can be created from a SODAR Core site template or by integrating the `django-sodar-core` package on an existing site. The Django site must be configured according to organization requirements, e.g. setting up user access via the organization LDAP/AD server.

The initial SODAR Core integration into a Django site adds e.g. a general layout, organization of apps and content into projects as well as project-based access control.

App Development

Next the developer needs to *develop the Django app(s)* which contain the actual program logic and user interfaces required for the use case. The content or number of these apps are not restricted by SODAR Core. Anything which would go to a typical Django app is OK, as long as certain building blocks for SODAR Core functionality are used.

Optional applications bundled with SODAR Core can also be enabled or disabled at this point. for example, if involved projects require sharing and storage of reports and presentations, the developer may select to enable the *filesfolders* app. Likewise, if detailed logging or audit trails are needed, the developer can enable the *timeline* app.

User and Project Setup

Once the site is deployed, the developer should *create initial project categories* and provide access to those for high level personnel such as project owners. The owners can then go on and create relevant projects, grant access to researchers and set up initial data in the applications.

Using the Site

The researchers will log in to the site on their web browser, in most cases using the standard LDAP credentials provided by their organization. They will see the projects they have been granted access to and can use whichever applications have been enabled or developed for the site, according to their assigned user rights. SODAR Core provides common navigation, overview and search views for all enabled apps, including the one(s) developed by the organization. The same user access management features are shared for all apps, along with possible REST APIs developed by the organization.

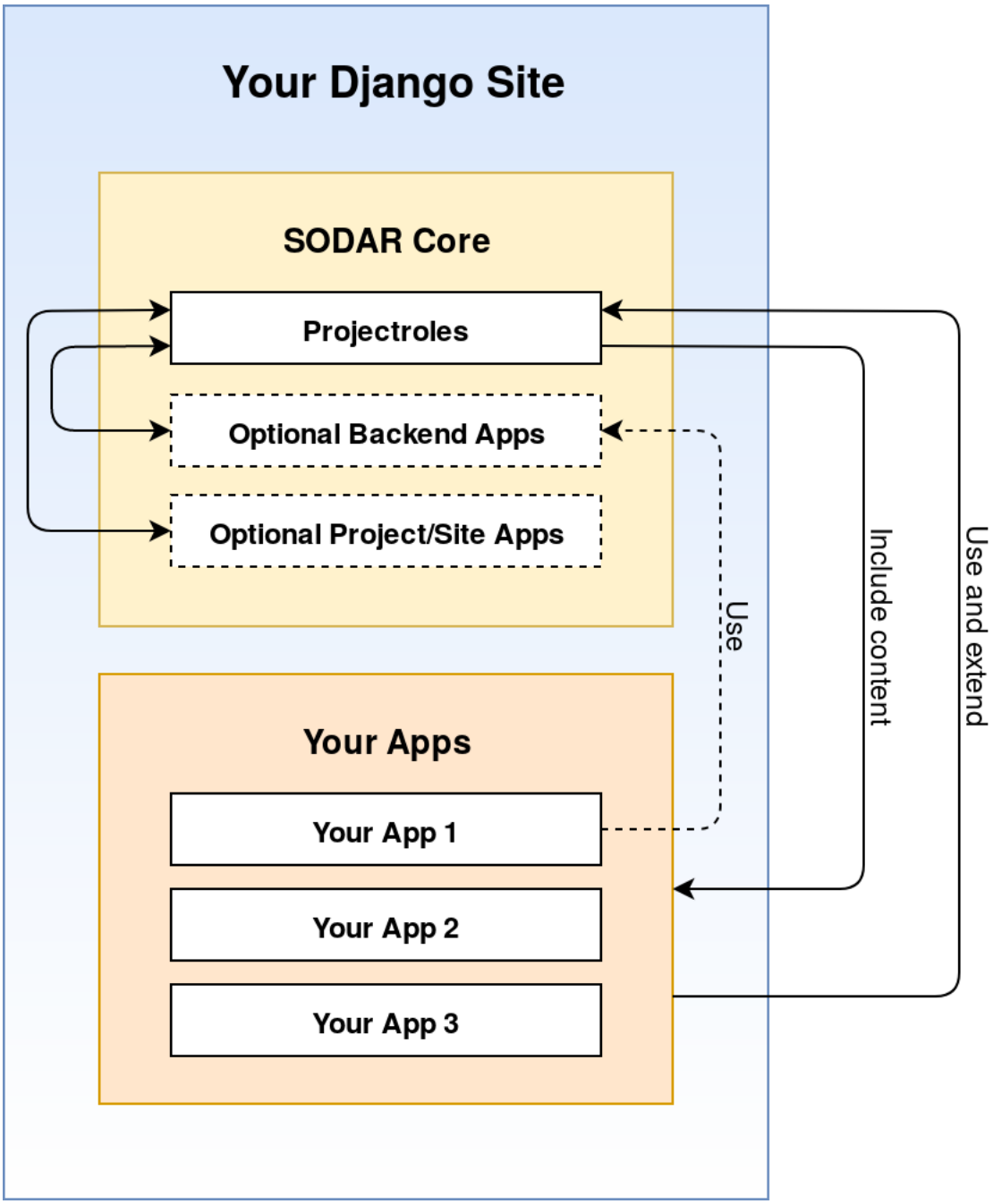


Fig. 1: Structure of a SODAR Core based web site

5.1.3 Next Steps

See the *Getting Started* document for instructions on installing SODAR Core and how to proceed with developing your own SODAR Core based site.

5.2 Getting Started

Installation and basic concepts of the SODAR Core framework and its apps are detailed in this document.

5.2.1 Installation

The `django-sodar-core` package can be installed from GitHub using pip as follows. It is strongly recommended to specify a version tag, as the package is under active development and breaking changes can be expected. PyPI install is forthcoming.

```
pip install -e git+https://github.com/bihealth/sodar_core.git@v0.9.0#egg=django-sodar-
↪core
```

Please note that the django-sodar-core package only installs *Django apps*, which you need to include in a *Django web site* project. For instructions for integrating SODAR Core into an existing Django site or setting up a new site, see the *projectroles app documentation*.

5.2.2 SODAR Core Apps

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.
- **tokens**: Token management for API access.
- **userprofile**: Site app for viewing user profiles.

5.2.3 Requirements

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

- Ubuntu (16.04 Xenial recommended and supported) / CentOS 7
- System 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 (**NOTE:** 2.x not currently supported)
- PostgreSQL >=9.6 and psycopg2-binary
- Bootstrap 4.x
- JQuery 3.3.x
- Shepherd and Tether
- Clipboard.js
- DataTables

For more details on installation and requirements for local development, see *SODAR Core Development*.

5.2.4 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.

5.3 For the Impatient

This section will give you the essential steps to setup a new SODAR Core based project. We will link to the parts of the manual where they were taken from such that you can read more in depth there.

5.3.1 See It In Action

We have developed the following data management and analysis web applications using SODAR Core. Although there only is a public demo available for VarFish at this time, the source code of the applications demonstrate how to use SODAR Core in complex web applications.

VarFish is a web-based tool for the analysis of variants. It showcases how to build a complex data warehousing and data analysis web appliction using SODAR Core. More details are described in the NAR Web Server Issue publication (doi:10.1093/nar/gkaa241). The source code can be found on github.com/bihealth/varfish-server. A demo is available at varfish-demo.bihealth.org.

DigestiFlow is a web-based data system for the management and demultiplexing of Illumina Flow Cells. It further implements various tools for sanity checking Illumina sample sheets and quality control (e.g., comparing barcode adapter sequence and actual sequence present in the sequencer output). You can find out more in our publication in Bioinformatics (doi:10.1093/bioinformatics/btz850). The source code can be found on github.com/bihealth/digestiflow-server. There currently is no public demo instance yet.

Kiosc is a web application that allows to build scheduler Docker containers for "data science" apps and dashboards. There currently is no public demo instance yet.

5.3.2 Prerequisites

Linux / Mac We use bash syntax on a Unix system and assume that you can adjust this to your system when needed.

PostgreSQL Please install version 9.6 or above. We assume that you have access to the `postgres` user or some other administrative user.

Development Essentials We assume that you have `git`, Python 3.6, and other essential tools installed. If you are using a mainstream Unix-like distribution (Mac qualifies) then you should be good to go.

5.3.3 Isolate Python Environment

If you use virtualenv, please create a new virtual environment for the project and activate it. Otherwise, follow the previous link and do this now or you can follow along us using conda.

The following creates a new Miniconda installation on 64 bit Linux or Mac. The Miniconda website has URLs to more.

```
# Linux
$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
$ bash Miniconda3-latest-Linux-x86_64.sh -b -p ~/miniconda3
$ source ~/miniconda3/bin/activate
$ conda install -y python=3.7

# Mac
$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
$ bash Miniconda3-latest-MacOSX-x86_64.sh -b -p ~/miniconda3
$ source ~/miniconda3/bin/activate
$ conda install -y python=3.7
```

For activating the conda installation, use `source ~/miniconda3/bin/activate`.

5.3.4 Install SODAR Core

We simply use `pip` for this:

```
# you must have your miniconda3 install sourced, skip if done already
$ source ~/miniconda3/bin/activate
$ pip install -e git+https://github.com/bihealth/sodar_core.git@v0.9.0#egg=django-
↪sodar-core
```

5.3.5 Download Example Site

We maintain a Git repository with a django project using the latest SODAR Core version here on GitHub: sodar_django_site. We will use this to get hit the ground running. See *Projectroles Integration* on other ways to get started with SODAR Core.

```
$ git clone https://github.com/bihealth/sodar_django_site.git  sodar_django_site
$ cd sodar_django_site
```

From here on, we assume that you are located (a) within the `sodar_django_site` directory and (b) have done `source ~/miniconda3/bin/activate` such that `which python` shows `~/miniconda3/bin/python`.

To complete this step install the development requirements.

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

5.3.6 Configure Environment

The next step is to perform some configuration. SODAR Core is built on the 12 factor app principles. Configuration is done using environment variables. For development, they are read from the `.env` file in your `sodar_django_site` checkout. We are shipping an example setting file that you should copy and then edit.

```
$ cp env.example .env
# now edit .env
```

To start out, it will be sufficient to make sure you can connect to the database. The default value for this is shown below.

```
DATABASE_URL="postgres://sodar_django_site:sodar_django_site@127.0.0.1/sodar_django_
↪site"
```

To keep it simple, you can use the following commands to create the correct database, user, and set the password.

```
$ sudo -u postgres createuser -ds sodar_django_site -W
[sudo] password for USER: <enter your password>
Password: <enter: sodar_django_site>
$ sudo -u postgres createdb --owner=sodar_django_site sodar_django_site
```

Now, we have to make sure that the environment file is read:

```
$ sed -ie "s/^READ_DOT_ENV_FILE.*/READ_DOT_ENV_FILE = env.bool('DJANGO_READ_DOT_ENV_
↪FILE', default=True)/" config/settings/base.py
```

5.3.7 Database Initialization

For the final steps, you will initialize the database. . .

```
$ python manage.py migrate
```

. . . and create a new admin user . . .

```
$ python manage.py createsuperuser
Username: root
Email address: root@example.com
Password:
Password (again):
Superuser created successfully.
```

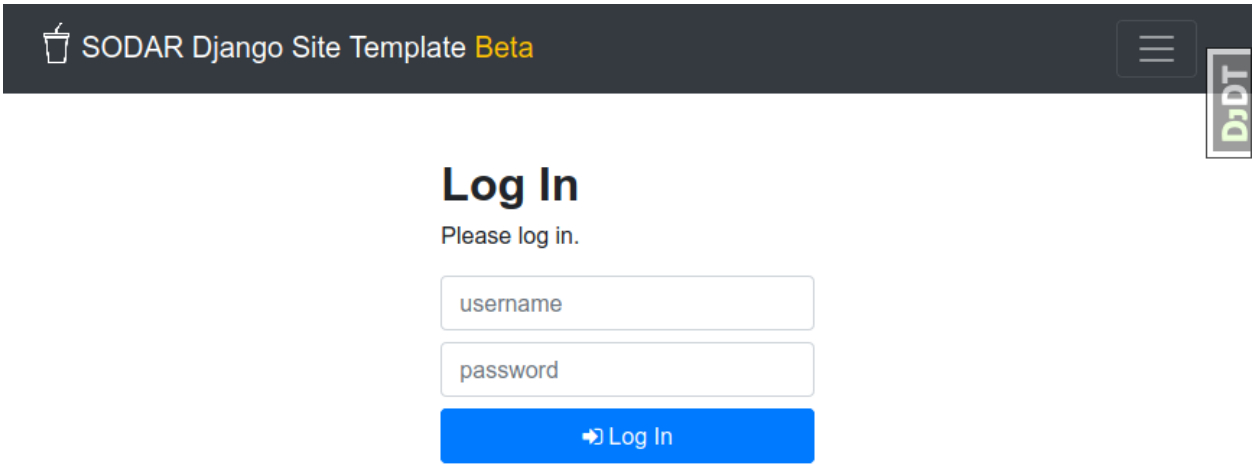
5.3.8 The First Login

Now, start the server, and you can then visit http://127.0.0.1:8000/login/?next=/

```
$ python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
February 03, 2020 - 10:00:53
Django version 1.11.25, using settings 'config.settings.local'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

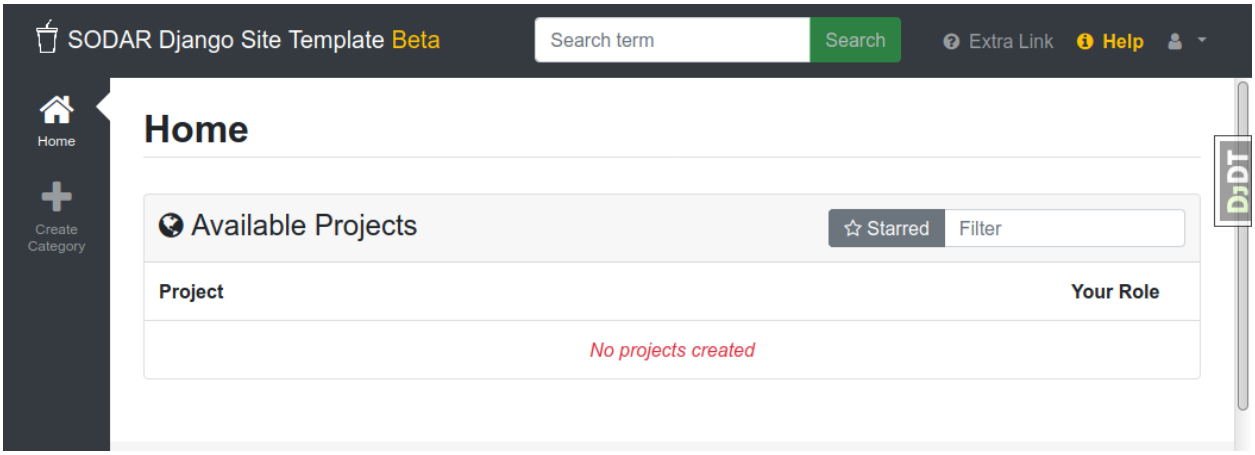
You should see the following:



Now, login with your superuser and you should see the following:

Click the little user icon on the top right to access the django admin (here is where you can create more users, for example) but also the preconfigured *Site Apps adminalerts*, *siteinfo*, *userprofile* and configuration for remote sites. The plus button on the left allows to create new categories and projects.

Now might also be a good time to read up more on the *projectroles* app as this is the fundamental app for most further development.



5.3.9 The First Project

You cannot create projects on the root level but you have to create a new category first (collections of projects). Use the "create category" button on the left to create a "example category" first, then create an "example project" within. The project details view should look as follows.

Note that the site search already works, so typing "example" into the text field on the top and clicking "search" will return your example project. The project overview shows the "project home card" for the installed project apps filefolders, timeline, and bgjobs. Usually, the latest five entries are shown here.

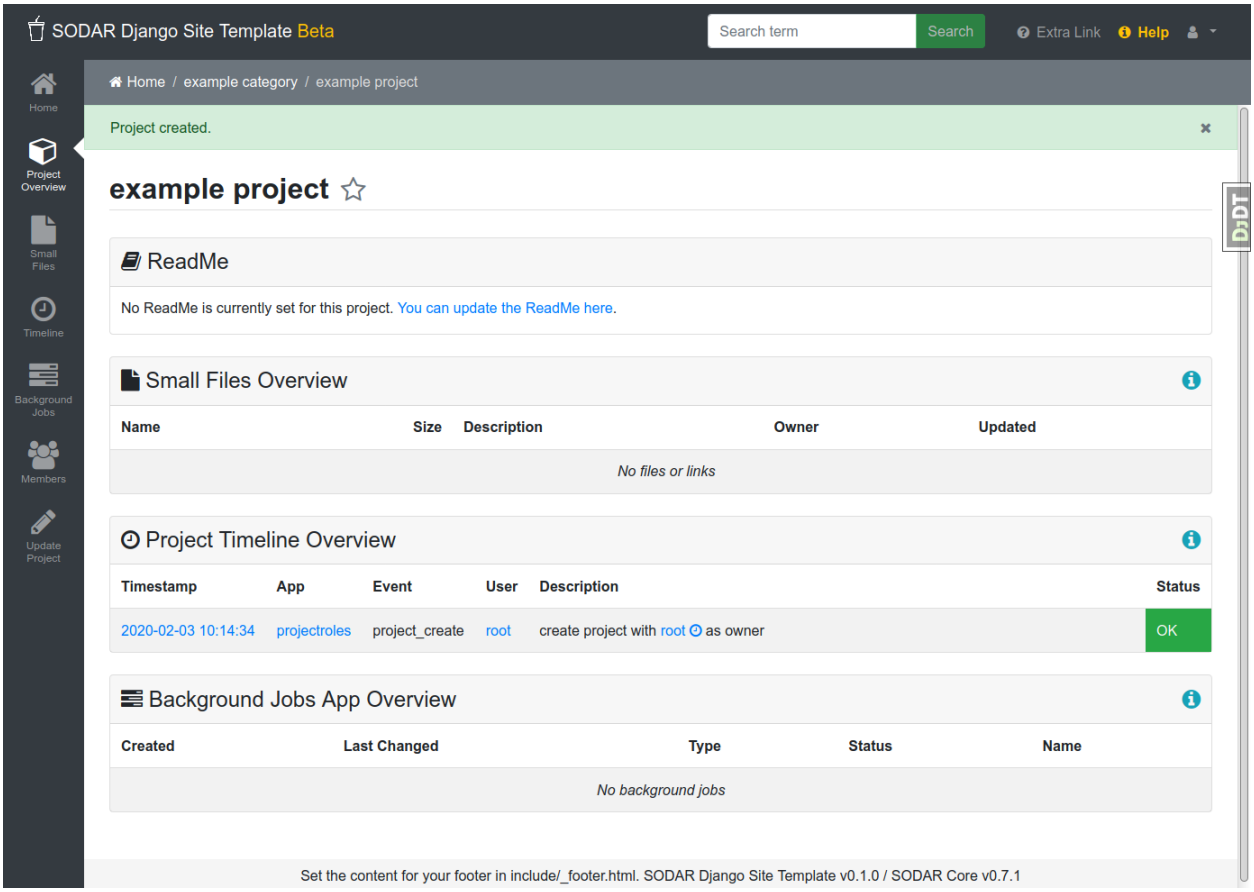
Note: The filesfolders app is an example of the **data management** application of SODAR Core based apps. You can easily imagine a more advanced module/app that not only allows tagging of files but more structuring data and meta data more strongly.

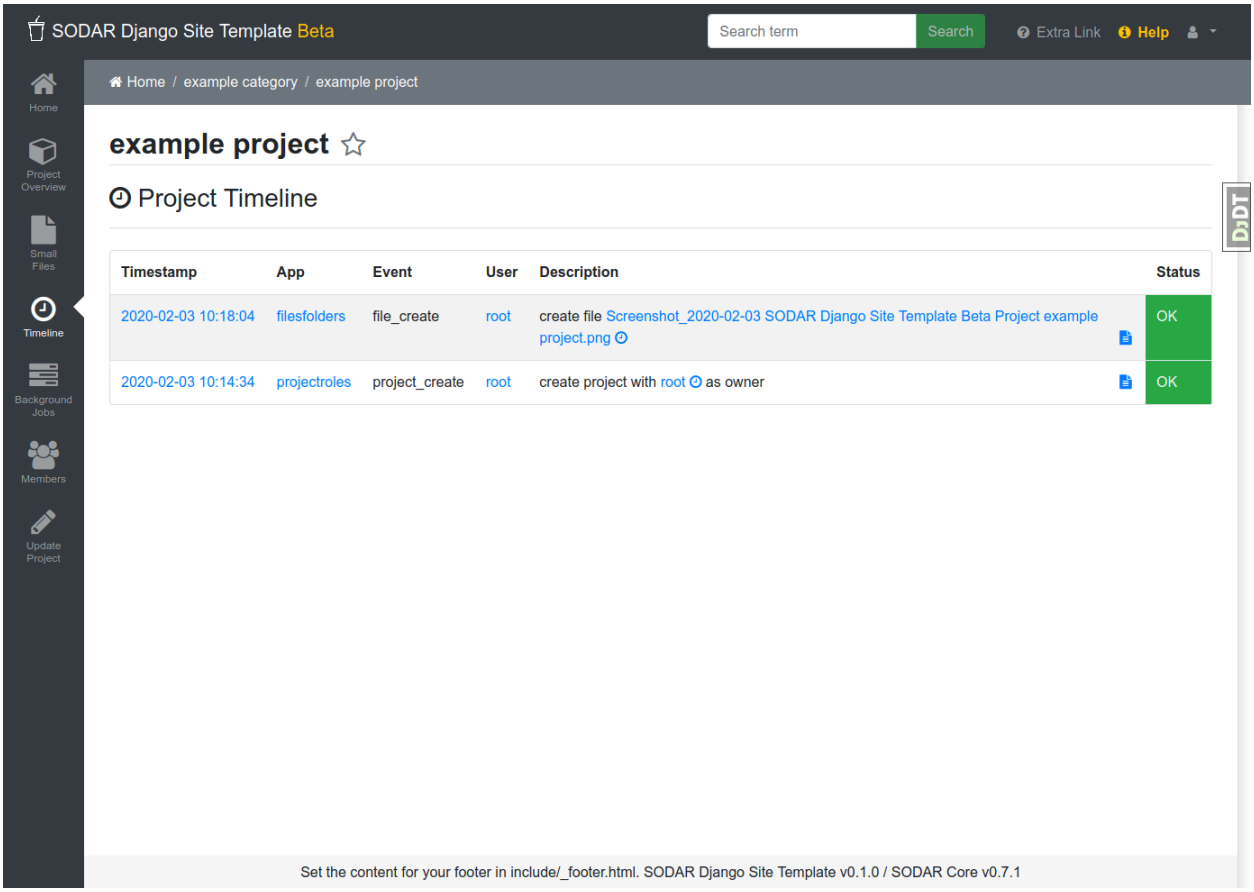
SODAR Core was extracted as a re-useable library of component from our (not yet released) SODAR project that allows the management of structured meta data of experiments and files with data from such experiments.

Go ahead and try out the filesfolders app by clicking the "small files" icon on the left. After creating folders and uploading a few files, you will see a trace of actions in the timeline app:

Note: By default, `sodar_django_site` will store the files in the PostgreSQL database but you can easily configure it to use other storage backends, e.g., the S3 protocol, with the django-storage package but this goes beyond this documentation.

Also, you will be able to find your uploaded file by name through the search box. Note that any app that you write can easily provide all the integrations with the SODAR Core framework (your apps are no different than the built-in ones). Just have a look how we did it in the apps shipping with SODAR Core.


SODAR Django Site Template Beta
Search term
Search
Extra Link
Help
Home
Project Overview
Small Files
Timeline
Background Jobs
Members
Update Project
Home / example category / example project
Project created.
example project
ReadMe
No ReadMe is currently set for this project. You can update the ReadMe here.
Small Files Overview
Name
Size
Description
Owner
Updated
No files or links
Project Timeline Overview
Timestamp
App
Event
User
Description
Status
2020-02-03 10:14:34
projectroles
project_create
root
create project with root as owner
OK
Background Jobs App Overview
Created
Last Changed
Type
Status
Name
No background jobs
Set the content for your footer in include/_footer.html. SODAR Django Site Template v0.1.0 / SODAR Core v0.7.1


SODAR Django Site Template Beta
Search term
Search
Extra Link
Help
Home
Project Overview
Small Files
Timeline
Background Jobs
Members
Update Project
Home / example category / example project
example project
Project Timeline
Timestamp
App
Event
User
Description
Status
2020-02-03 10:18:04
filesfolders
file_create
root
create file Screenshot_2020-02-03 SODAR Django Site Template Beta Project example project.png
OK
2020-02-03 10:14:34
projectroles
project_create
root
create project with root as owner
OK
Set the content for your footer in include/_footer.html. SODAR Django Site Template v0.1.0 / SODAR Core v0.7.1

5.3.10 Summary

Here is a quick summary on how SODAR Core interacts with the built-in and user apps:

- At the lower most level all content is managed in projects which themselves can be assigned into categories.
- Project apps can provide new content types that can be put into projects. For example, the filesfolders app allows you to store files, folders, and assign meta data to them. As another example, the timelines app stores events that occured in a project, and other apps such as the filesfolders app can register their events with it.
- SODAR Core defines several plugin extension points that your apps can implement and make their content findable, for example.
- Site apps allow to provide features independent of a project. For example, the userprofile app allows to access user settings and the adminalerts app allows to post global notifications.

5.3.11 Going on From Here

- You can now start exploring your `sodar_django_site` and play around with it.
- You can read the :ref`user_stories` section to learn how SODAR Core based applications are built.
- Continue reading *Getting Started* for a more comprehensive documentation and walk-through of SODAR Core and its apps.
- Have a look at the web apps developed by us that are using SODAR Core as shown in the *See It In Action* section.

5.4 User Stories

This section explains how SODAR Core based web applications are built on a very high level. We assume that you have read the *For the Impatient* section and are basing your web application on the SODAR Core example site as described there. Also, we assume that you have intermediate experience with Django and Python programming.

Please also note that the term *web app* refers to the overall *Django site*, that is what the user sees and what is commonly referred to as a "web application" or "dynamic website".

The term *Django app* refers to the technical term within Django development, that is a "Django app Python package".

5.4.1 Flow Cell Data Management

On a very high level here is how SODAR Core was used to built Digestiflow (cf. *See It In Action*). You can find the source code in the Github project.

The aim is to manage the meta data for sequencers and flow cells.

SODAR Core App Configuration

- Configure your Django site to use the `projectroles` SODAR Core Django app. Each SODAR Core "project" corresponds to one site and this would allow to have multiple groups manage their sequencing meta data in the same web app instance.
- Configure the `timeline` SODAR Core Django app to provide an audit trail of changes to data.
- Configure the `filesfolders` SODAR Core Django app to manage small file uploads in various places.

Custom Apps

The following description uses domain-specific language from the high-throughput sequencing domain. While this makes the section harder to understand to the layperson, explaining the different terms is out of scope in this manual.

- Write the `sequencers` app for management of sequencing machines.
- Write the `barcodes` app for management of the barcode adapter sets.
- Write the `flowcells` app for managing flow cells and libraries. This app depends on `sequencers` and `barcodes` for these apps' Django models.

5.4.2 Variant Analysis

On a very high level here is how SODAR Core was used to built VarFish (cf. *See It In Action*). You can find the source code in the Github project.

The aim is to provide an data analysis web application.

SODAR Core App Configuration

- Configure your Django site to use the `projectroles` SODAR Core Django app. Each SODAR Core "project" corresponds to one site and this would allow to have multiple groups manage their sequencing meta data in the same web app instance.
- Configure the `timeline` SODAR Core Django app to provide an audit trail of changes to data.
- Configure the `bgjobs` SODAR Core Django app to manage asynchronous jobs, such as long-running queries.

Custom Apps

The following description uses domain-specific language from the medical genetics domain. While this makes the section harder to understand to the layperson, explaining the different terms is out of scope in this manual.

- Write various Django apps for importing background data such as population frequencies from the gnomAD project, genes, etc.
- Write the `variants` app to implement the actual variant filtration. This is the core part of Varfish and provides the different Django models, views, and templates to actually perform the variant filtration.
- Write the `importer` app to implement efficient bulk import of variants from annotated TSV files.

5.5 Projectroles App

The `projectroles` app is the base app for building a *SODAR Core based Django site*. It provides a ramework for project access management, dynamic content retrieval, 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 Core project and app structure.

5.5.1 Projectroles Basics

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

Projects

The projectroles app groups data into **projects**. Here, a **project** is a data container object that other objects can be linked to (typically through a 1:n foreign key relationship). A **category** is a sub-type of a project which is allowed to contain other categories and projects but no other data type.

Using categories and projects, data can be organized in a tree structure of category and project "containers". Users can be granted access to projects using roles as described in the next section.

User Roles in Projects

A **role** is a data type that has a string identifier in it score (e.g., "project guest"). Roles are assigned to individual users in the context of individual projects (a n:m relation from user to project with the string identifier). For example, user "paul" might be assigned the "project guest" role in one project and another (or no) role in a second project. Users can only have one role in a given project at any given time. New types of roles can be defined by extending the default model's database table of the projectroles app.

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 (by 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
- 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 can have one of three modes: **source**, **target** or **peer** mode.

A SODAR site can be set by the user 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.
- **Peer** mode is used only if two or more Target sites link to the same Source site. If synchronizing a project which has multiple accessing Target sites, metadata about those other Target sites is included and stored in Peer mode site objects.

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)
- Information of other Target Sites linking a common project

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.

App Plugins

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

- **Project apps**: Apps related to specific projects, making use of project access control and providing data and content within the project's scope
- **Site apps**: Site-wide Django apps which are not project-specific
- **Backend apps**: Backend apps without a GUI entry point, imported and used dynamically by other SODAR-based apps for e.g. connectivity to external resources.

App plugins are not limited to one per Django app. A single Django app in SODAR Core may contain one or more of the aforementioned plugin types, depending on the required functionality.

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 and site 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 Timeline integration**: Included but disabled unless backend apps for Taskflow and Timeline are integrated in the Django site

Templates and Styles

Projectoles 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.

5.5.2 Projectroles Integration

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

Installation on a New Site

If you want to set up a new Django 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 the 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 following 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 sing 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+https://github.com/mikkonie/django-plugins.
↪git@1bc07181e6ab68b0f9ed3a00382eb1f6519e1009#egg=django-plugins
-e git+https://github.com/bihealth/sodar_core.git@v0.9.0#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 based 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 include 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 = [
    # ...
    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 SODAR Core based Django site with support for projectroles features and SODAR Core apps. To test the site locally execute the supplied make command:

```
$ make serve
```

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.

5.5.3 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, it is highly recommended to set `DJANGO_READ_DOT_ENV_FILE=1` in your system's environment variables

and place the env variables into a `.env` file in the root directory of your Django site repository. See `env.example` for an example of such a file.

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',
    '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 `djangorestframework` 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_EMAIL_SENDER_REPLY`: Whether replies are expected to the sender address (bool). If set `False` and nothing is set in the `reply-to` header, a "do not reply" note is added to the email body.
- `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)
PROJECTROLES_EMAIL_SENDER_REPLY = env.bool('PROJECTROLES_EMAIL_SENDER_REPLY', False)
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)
- PROJECTROLES_KIOSK_MODE: If true, allow accessing certain project views *without* user authentication in order to e.g. demonstrate features in a kiosk-style deployment. Also hides and/or disables views not intended to be used in this mode (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
PROJECTROLES_KIOSK_MODE = False
```

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.

Warning: The PROJECTROLES_KIOSK_MODE setting is under development and considered experimental. More implementation, testing and documentation is forthcoming.

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, [])
```

API View Settings (Optional)

If you want to build an API to your site using SODAR Core functionality, it is recommended to base your API views on `projectroles.views.SODARAPIBaseView`. Using this base class also allows you to define your API media type, version number and allowed versions via Django settings.

The recommended API setup uses accept header versioning. The `SODAR_API_MEDIA_TYPE` setting should be changed to your organization and API identification if API views are 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/your.application+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 = {
```

(continues on next page)

(continued from previous page)

```
    'first_name': 'givenName',
    '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,
        )
    )
```

SAML SSO Configuration (optional)

Optional Single Sign-On (SSO) authorization via SAML is also available. To enable this feature, set `ENABLE_SAML=1` in your environment. Configuring SAML for SSO requires proper configuration of the Keycloak SSO server and the SAML client library.

Keycloak

Create a new client in Keycloak and configure it as follows. Please note that **Client ID** can be chosen however you like, but it must match the setting in the client.

To generate the `metadata.xml` file required for the client, go to the **Realm Settings** page and in the **General** tab, click `SAML 2.0 Identity Provider Metadata` to download the xml data. Save it somewhere on the client, the preferred name is `metadata.xml`.

For the signing of the request send to the Keycloak server you will require a certificate and key provided by the Keycloak server and incorporated into the configuration of the client. Switch to the `SAML Keys`. Make sure to select `PKCS12` as **Archive Format**.

Convert the archive on the commandline with the follow command and store them in some place on your client.

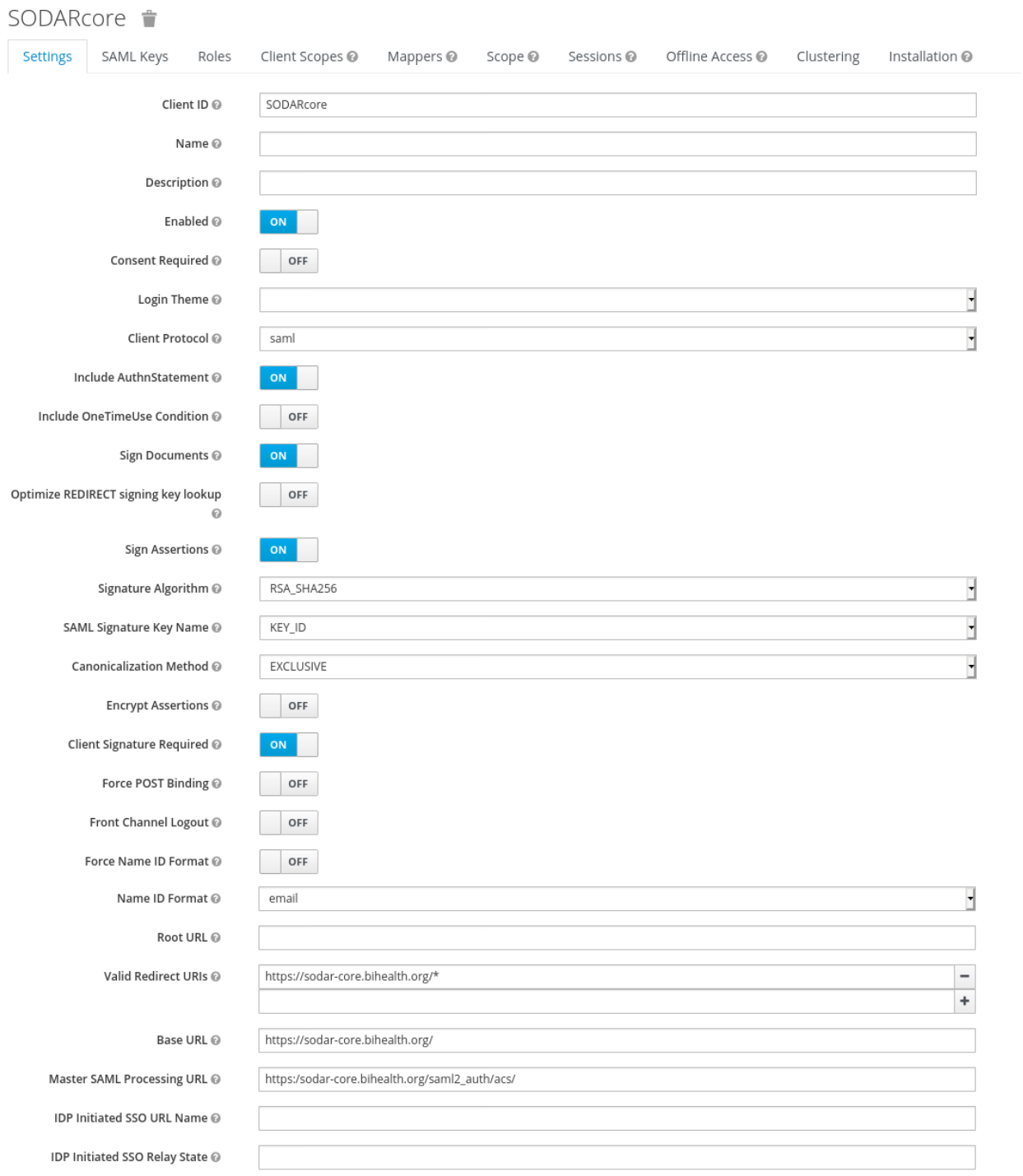
```
openssl pkcs12 -in keystore.p12 -password "pass:<PASSWORD>" -nodes | openssl x509 -
↪out cert.pem
openssl pkcs12 -in keystore.p12 -password "pass:<PASSWORD>" -nodes -nocerts | openssl␣
↪rsa -out key.pem
```

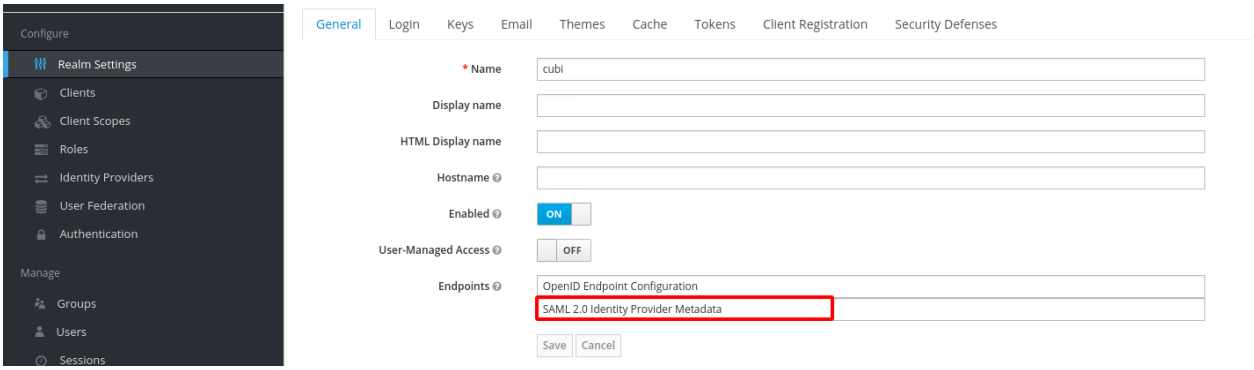
SODAR Core

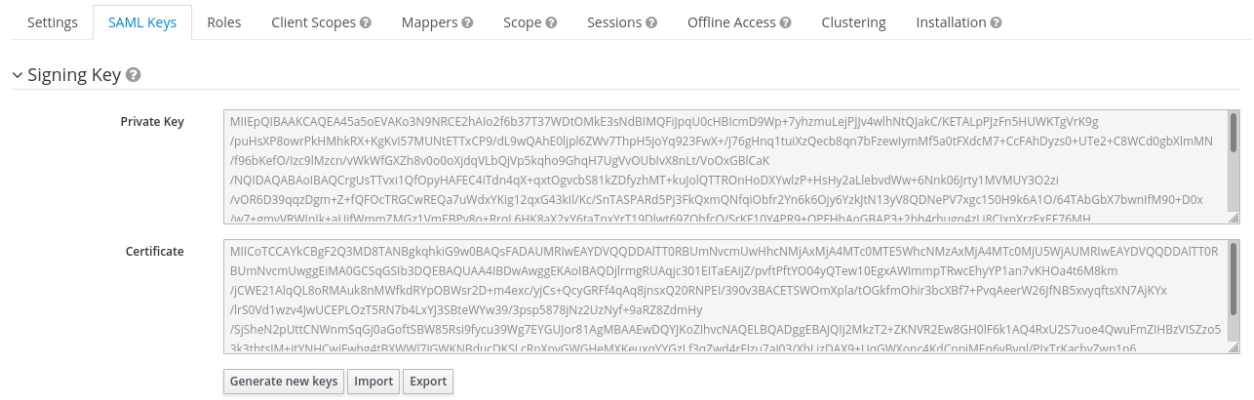
Make sure that your `config/settings/base.py` contains the following configuration:

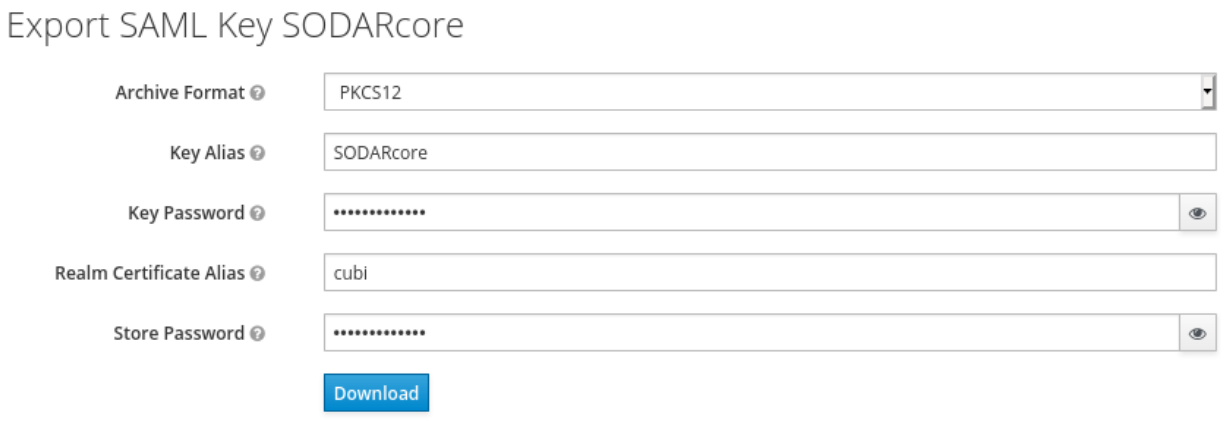
```
ENABLE_SAML = env.bool('ENABLE_SAML', False)
SAML2_AUTH = {
    # Required setting
    'SAML_CLIENT_SETTINGS': {  # Pysaml2 Saml client settings (https://pysaml2.
↪readthedocs.io/en/latest/howto/config.html)
        'entityid': env.str(
            'SAML_CLIENT_ENTITY_ID', 'SODARcore'
        ),  # The optional entity ID string to be passed in the 'Issuer' element of␣
↪authn request, if required by the IDP.
        'entitybaseurl': env.str(
            'SAML_CLIENT_ENTITY_URL', 'https://localhost:8000'
        ),
        'metadata': {
            'local': [
                env.str(
                    'SAML_CLIENT_METADATA_FILE', 'metadata.xml'
                ),  # The auto(dynamic) metadata configuration URL of SAML2
            ],
        },
        "service": {
            'sp': {
                'idp': env.str(
                    'SAML_CLIENT_IPD',
                    'https://sso.hpc.bihealth.org/auth/realms/cubi',
                ),
```

(continues on next page)


SODARcore
Settings
SAML Keys
Roles
Client Scopes
Mappers
Scope
Sessions
Offline Access
Clustering
Installation
Client ID
SODARcore
Name
Description
Enabled
ON
Consent Required
OFF
Login Theme
Client Protocol
saml
Include AuthnStatement
ON
Include OneTimeUse Condition
OFF
Sign Documents
ON
Optimize REDIRECT signing key lookup
OFF
Sign Assertions
ON
Signature Algorithm
RSA_SHA256
SAML Signature Key Name
KEY_ID
Canonicalization Method
EXCLUSIVE
Encrypt Assertions
OFF
Client Signature Required
ON
Force POST Binding
OFF
Front Channel Logout
OFF
Force Name ID Format
OFF
Name ID Format
email
Root URL
Valid Redirect URIs
https://sodar-core.bihealth.org/*
Base URL
https://sodar-core.bihealth.org/
Master SAML Processing URL
https:/sodar-core.bihealth.org/saml2_auth/acs/
IDP Initiated SSO URL Name
IDP Initiated SSO Relay State


Configure
Realm Settings
Clients
Client Scopes
Roles
Identity Providers
User Federation
Authentication
Manage
Groups
Users
Sessions
General
Login
Keys
Email
Themes
Cache
Tokens
Client Registration
Security Defenses
* Name
cubi
Display name
HTML Display name
Hostname
Enabled
ON
User-Managed Access
OFF
Endpoints
OpenID Endpoint Configuration
SAML 2.0 Identity Provider Metadata
Save
Cancel


Settings
SAML Keys
Roles
Client Scopes
Mappers
Scope
Sessions
Offline Access
Clustering
Installation
Signing Key
Private Key
MIIEpQIBAAKCAQEA45a5oEVAKo3N9NRCE2hAIo2f6b37T37WDtOMkE3sNdBIMQFiJpqU0cHBlcmD9Wp+7yhzmuLejPJJv4wlhNtQJakC/KETALpPJzFn5HUWKTgVrK9g
/puHsXP8owrPkHMhkRX+KgKvl57MUNtETTxCP9/dL9wQAhE0ljpl6ZWv7ThpH5joYq923FwX+/j76gHnq1tuiXzQecb8qn7bFzewIymMf5a0tFXdcM7+CcFAhDyzs0+UTe2+C8WCd0gbXlmMN
/f96bKefO/Izc9lMzcn/vWkWfGXZh8v0o0oXjdqVLbQjVp5kqho9GhqH7UgVvOUblvX8nLt/VoOxGBlCaK
/NQIDAQABAoIBAQCrgUsTTvxi1QfOpyHAFEC4iTdn4qX+qxtOgvcbS81kZDfyzhMT+kuJolQTTROnHoDXYwlzP+HsHy2aLlebvdWw+6Nnk06Jrty1MVMUY3O2zi
/vOR6D39qqzDgm+Z+fQFOcTRGCwREQa7uWdxYKig12qxG43kll/Kc/SnTASPARd5PJ3FkQxmQNfqiObfr2Yn6k6Ojy6YzkJtN13yV8QDNePV7xgc150H9k6A1O/64TAbGbX7bwnlfM90+D0x
/w7+gmvVRWIplk+alJjfWmmZMGz1VmFRPv8o+RrnL6HK8aX2xY6taTnxYrT19Dlwf697ObfcO/SrkF10Y4PR9+QPFHhAoGRAP3+2hb4rhugn4zLI8CIxnXrzFxFF6MH
Certificate
MIICoTCCAYkCBgF2Q3MD8TANBgkqhkiG9w0BAQsFADAUMRIwEAYDVQQDDAlTT0RBUmNvcmUwHhcNMjAxMjA4MTc0MTE5WhcNMzAxMjA4MTc0MjU5WjAUMRIwEAYDVQQDDAlTT0R
BUmNvcmUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDjlrmgRUAqjc301ElTaEAijZ/pvftPftYO04yQTew10EgxAWImmpTRwcEhyYP1an7vKHOa4t6M8km
/jCWE21AlqQL8oRMAuk8nMWfkdRYpOBWsr2D+m4exc/yjCs+QcyGRFf4qAq8jnsxQ20RNPEI/390v3BACETSWOmXpla/tOGkfmOhir3bcXBf7+PvqAeerW26jfNB5xvyqftsXN7AjKYx
/lrS0Vd1wzv4JwUCEPLOzT5RN7b4LxYJ3SBteWYw39/3psp5878jNz2UzNyf+9aRZ8ZdmHy
/SjSheN2pUttCNWnmSqGj0aGoftSBW85Rsi9fycu39Wg7EYGUJor81AgMBAAEwDQYJKoZIhvcNAQELBQADggEBAJQIj2MkzT2+ZKNVR2Ew8GH0lF6k1AQ4RxU2S7uoe4QwuFmZlHBzVISZzo5
3k3thtsIM+itVNHCwiEwhg4tRXWWl7IGWkNRducDKSLcRnXnvGWGHeMXKeuvnYYGzLf3q7wd4rFIzu7al03/XbLizDAX9+UoGWXonc4KdCnniMEn6vBvol/PIxTrKachv7wn1n6
Generate new keys
Import
Export


Export SAML Key SODARcore
Archive Format
PKCS12
Key Alias
SODARcore
Key Password
Realm Certificate Alias
cubi
Store Password
Download

(continued from previous page)

```
                # Keycloak expects client signature
                'authn_requests_signed': 'true',
                # Enforce POST binding which is required by keycloak
                'binding': 'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST',
            },
        },
        'key_file': env.str('SAML_CLIENT_KEY_FILE', 'key.pem'),
        'cert_file': env.str('SAML_CLIENT_CERT_FILE', 'cert.pem'),
        'xmlsec_binary': env.str('SAML_CLIENT_XMLSEC1', '/usr/bin/xmlsec1'),
        'encryption_keypairs': [
            {
                'key_file': env.str('SAML_CLIENT_KEY_FILE', 'key.pem'),
                'cert_file': env.str('SAML_CLIENT_CERT_FILE', 'cert.pem'),
            }
        ],
    },
    'DEFAULT_NEXT_URL': '/',  # Custom target redirect URL after the user get logged␣
↪in. Default to /admin if not set. This setting will be overwritten if you have␣
↪parameter ?next= specificed in the login URL.
    # # Optional settings below
    # 'NEW_USER_PROFILE': {
    #     'USER_GROUPS': [],  # The default group name when a new user logs in
    #     'ACTIVE_STATUS': True,  # The default active status for new users
    #     'STAFF_STATUS': True,  # The staff status for new users
    #     'SUPERUSER_STATUS': False,  # The superuser status for new users
    # },
    # 'ATTRIBUTES_MAP': {  # Change Email/UserName/FirstName/LastName to␣
↪corresponding SAML2 userprofile attributes.
    #     'email': 'Email',
    #     'username': 'UserName',
    #     'first_name': 'FirstName',
    #     'last_name': 'LastName',
    # },
    # 'TRIGGER': {
    #     'FIND_USER': 'path.to.your.find.user.hook.method',
    #     'NEW_USER': 'path.to.your.new.user.hook.method',
    #     'CREATE_USER': 'path.to.your.create.user.hook.method',
    #     'BEFORE_LOGIN': 'path.to.your.login.hook.method',
    # },
    # 'ASSERTION_URL': 'https://cubi5.bihealth.org:8000',  # Custom URL to validate␣
↪incoming SAML requests against
}
```

Add the following settings to your environment variables:

```
ENABLE_SAML=1
SAML_CLIENT_ENTITY_ID=<Entity ID configured in Keycloak>
SAML_CLIENT_ENTITY_URL=<Client URL, e.g. https://sodar-core.bihealth.org>
SAML_CLIENT_METADATA_FILE=<e.g. metadata.xml>
SAML_CLIENT_IPO=<SSO server URL, e.g. https://sso.hpc.bihealth.org/auth/realms/cubi>
SAML_CLIENT_KEY_FILE=<e.g. key.pem>
SAML_CLIENT_CERT_FILE=<e.g. cert.pem>
SAML_CLIENT_XMLSEC1=<e.g. /usr/bin/xmlsec1>
```

Global JS/CSS Include Modifications (Optional)

It is possible to supplement (or replace, see below) global Javascript and CSS includes of your SODAR Core site without altering the base template. You can place a list of custom includes into the list variables `PROJECTROLES_CUSTOM_JS_INCLUDES` and `PROJECTROLES_CUSTOM_CSS_INCLUDES`. These can either be local static file paths or web URLs to e.g. CDN served files.

If using the default CDN imports for JQuery, Bootstrap4 etc. are not an optimal solution in your use case due to e.g. network issues, you can disable these includes by setting `PROJECTROLES_DISABLE_CDN_INCLUDES` to `True`.

Warning: If disabling the default CDN includes, you **must** provide replacements for **all** disabled files in your custom includes. Otherwise your SODAR Core based site will not function correctly!

Example:

```
PROJECTROLES_DISABLE_CDN_INCLUDES = True
PROJECTROLES_CUSTOM_JS_INCLUDES = [
    STATIC_ROOT + '/your/path/jquery-3.3.1.min.js',
    STATIC_ROOT + '/your/path/popper.min.js',
    'https://some-cdn.com/bootstrap.min.js',
    # ...
]
PROJECTROLES_CUSTOM_CSS_INCLUDES = [
    STATIC_ROOT + '/your/path/bootstrap.min.css',
    STATIC_ROOT + '/your/path/font-awesome.min.css',
    # ...
]
```

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.

5.5.4 Projectroles Usage

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

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`.

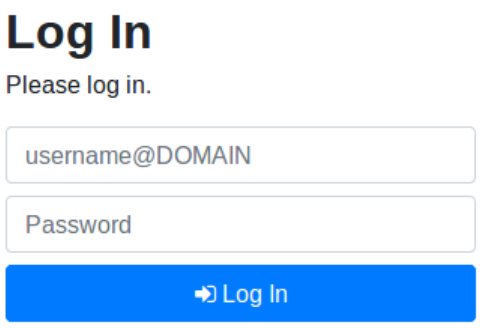


Fig. 2: SODAR login form

User Interface

Basics

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

- **Top navigation bar**: Contains the site logo and title, search element, link to advanced search, help link and the user dropdown menu.
- **User dropown 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

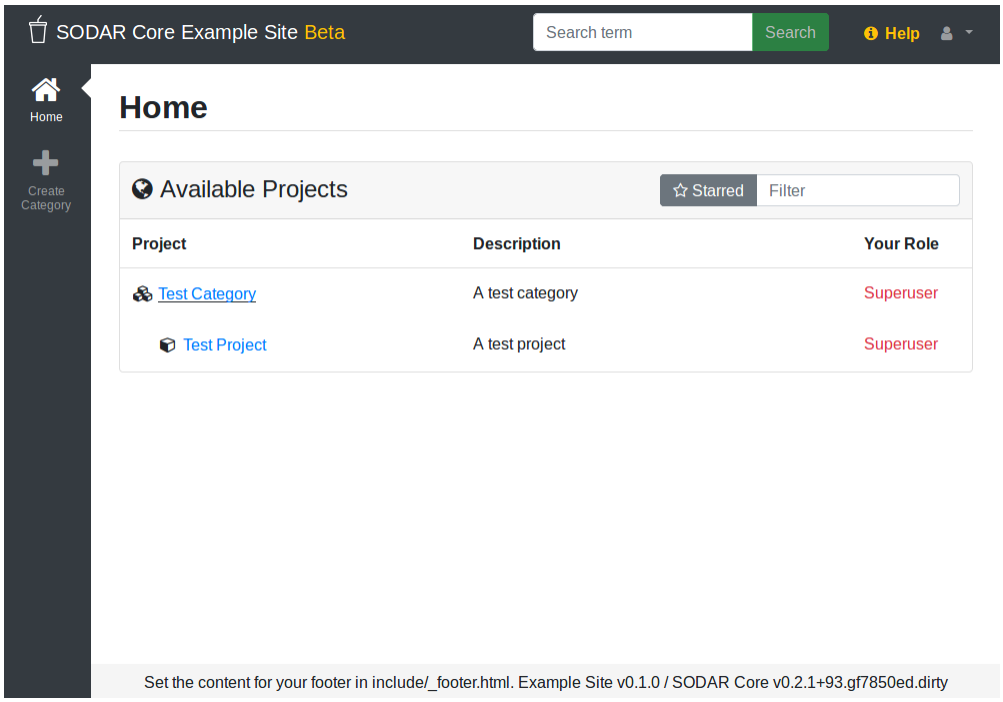


Fig. 3: Home view

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 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.

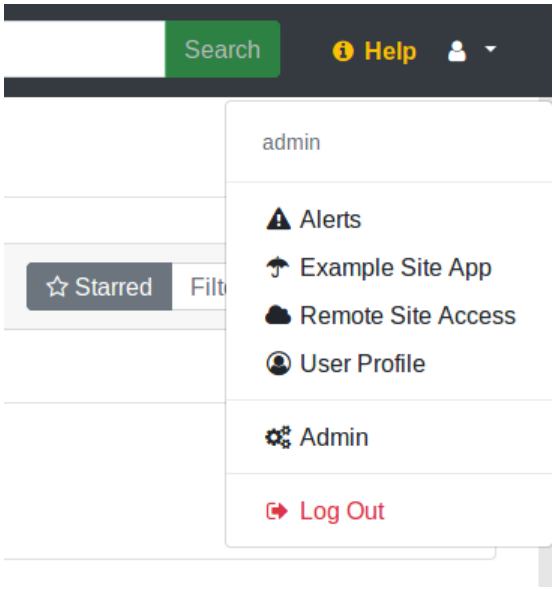


Fig. 4: User dropdown

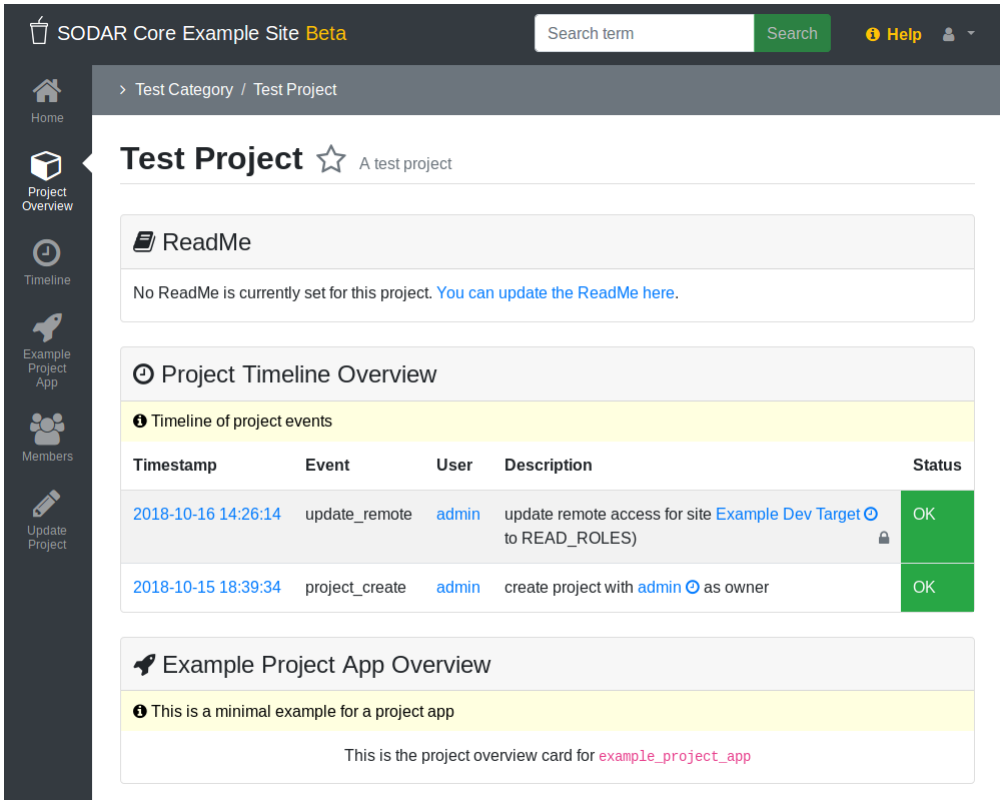


Fig. 5: Project detail view

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 are disabled for categories unless explicitly enabled. 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.

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.

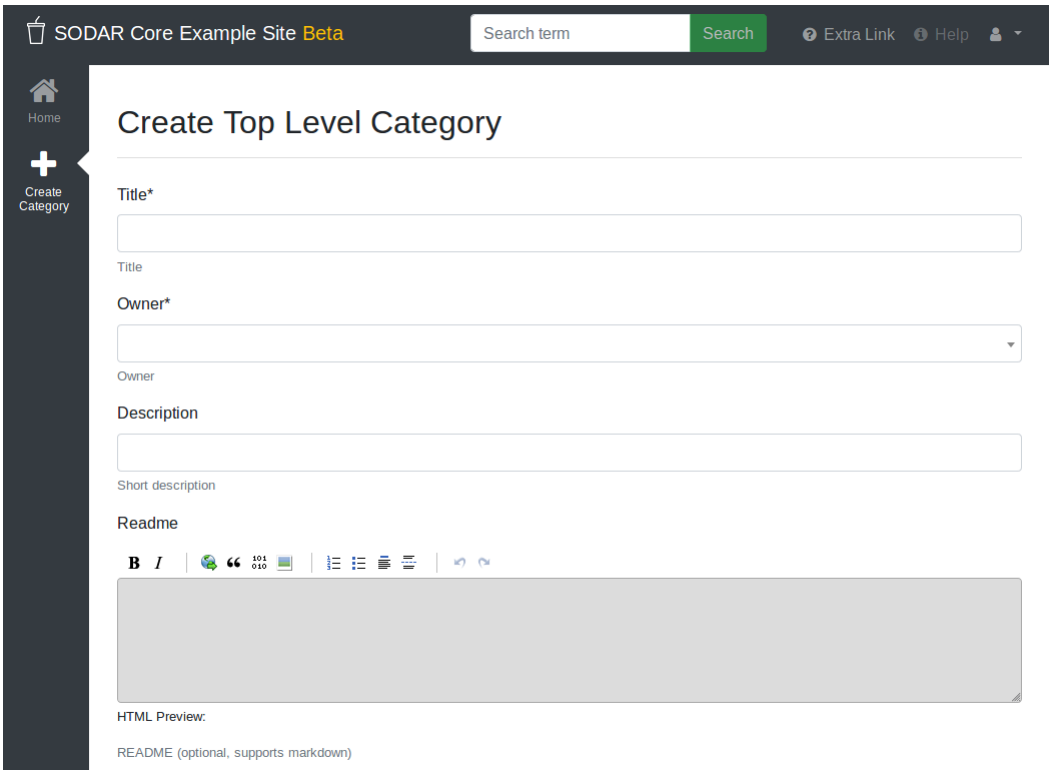


Fig. 6: Category/project creation form

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.

Users with the role of *project contributor* or higher in a category are allowed to create a project within that 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. The owner can not be changed here, but must be modified in the *Members* view instead. It is possible to move the current category or project under another category by altering the `parent` field. The user who does the updating must have a sufficient role in the target category or superuser status.

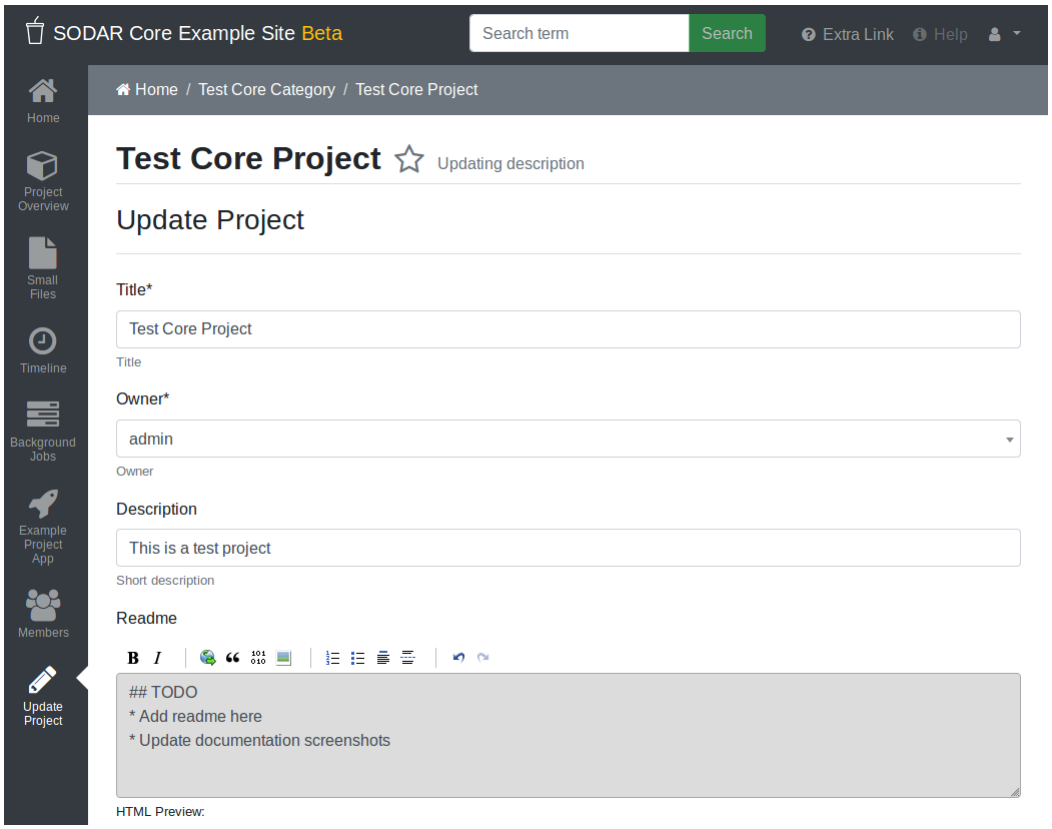


Fig. 7: Category/project updating form

Note: For remote project synchronized from another SODAR Core based site, you can only edit local application settings in this view.

App Settings

Project and site apps may define *app settings*, which can be either be set with the scope of *project*, *user* or *user within a project*.

Widgets for project specific settings will show up in the project creation and updating form and can only be modified by users with sufficient project access. User specific settings will be displayed in the *Userpforile app*.

By defining the attribute `user_modifiable=False`, project or user app settings will not be shown in the respective project/user update views. This is used e.g. in cases where a project app provides its own UI or updates some "hidden" setting due to user actions. Superusers will still see these hidden settings in the Update Project view.

Settings with the scope of user within a project do not currently have a separate UI of their own. Instead, project apps can produce their own user specific UIs for this functionality if manual user selection is needed.

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.

The projectroles app provides the following built-in app settings with the project scope:

- `ip_restrict`: Restict project access by an allowed IP list if enabled.
- `ip_allowlist`: List of allowed IP addresses for project access.

To clean up settings which have been stored in the database but have since been removed from the plugin app settings definitions, run the following management command:

```
$ ./manage.py cleanappsettings
```

Member Management

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

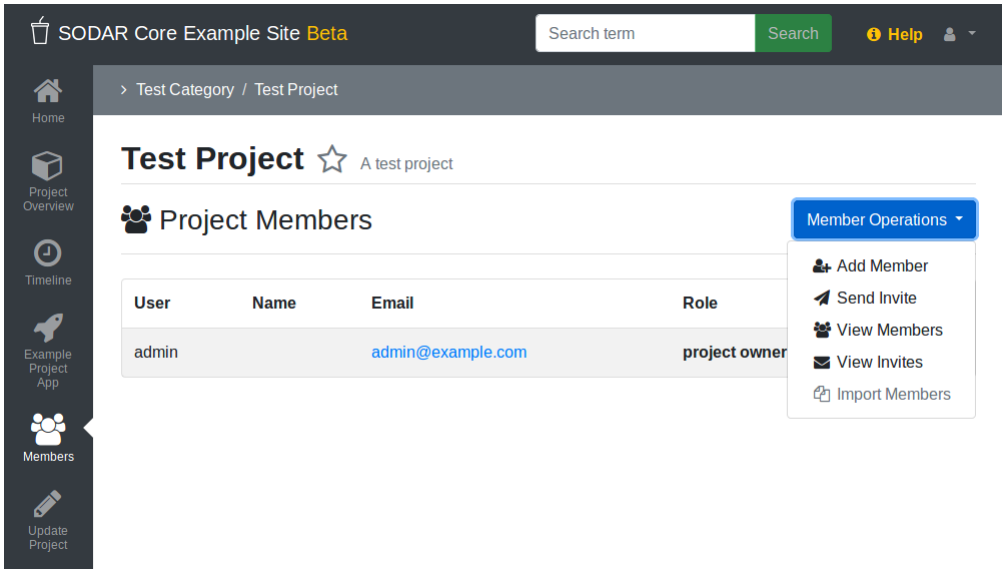


Fig. 8: Project member list view

Note: Owners of categories automatically inherit owner rights to projects placed under those categories, starting in SODAR Core v0.8.0. Adding separate roles for those users in the inherited projects is not allowed.

Note: At this time, category memberships are not automatically propagated to projects created under the category. An inheritance functionality may be implemented at a later date.

Adding Members

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

- **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.

Modifying Members

Changing or removing user roles can be done from links next to each role on the member list. Category or project ownership can be transferred to another user who currently has a role in the project by using the dropdown next to the owner role.

Invites

Invites are accepted by the responding user clicking on a link supplied in their invite email and either logging in to the site with their LDAP/AD credentials or creating a local user. The latter is only allowed if local users are enabled in the site's Django settings and the user email domain is not associated with configured LDAP domains. Invites expire after a certain time and can be reissued or revoked on the **Project Invites** page.

Batch Member Modifications

Batch member updates can be done either by using REST API views with appropriate project permissions, or by a site admin using the `batchupdateroles` management command. The latter supports multiple projects in one batch. It is also able to send invites to users who have not yet signed up on the site.

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.

Alternatively, remote sites can be created using the following management command:

```
$ ./manage.py addremotesite
```

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.

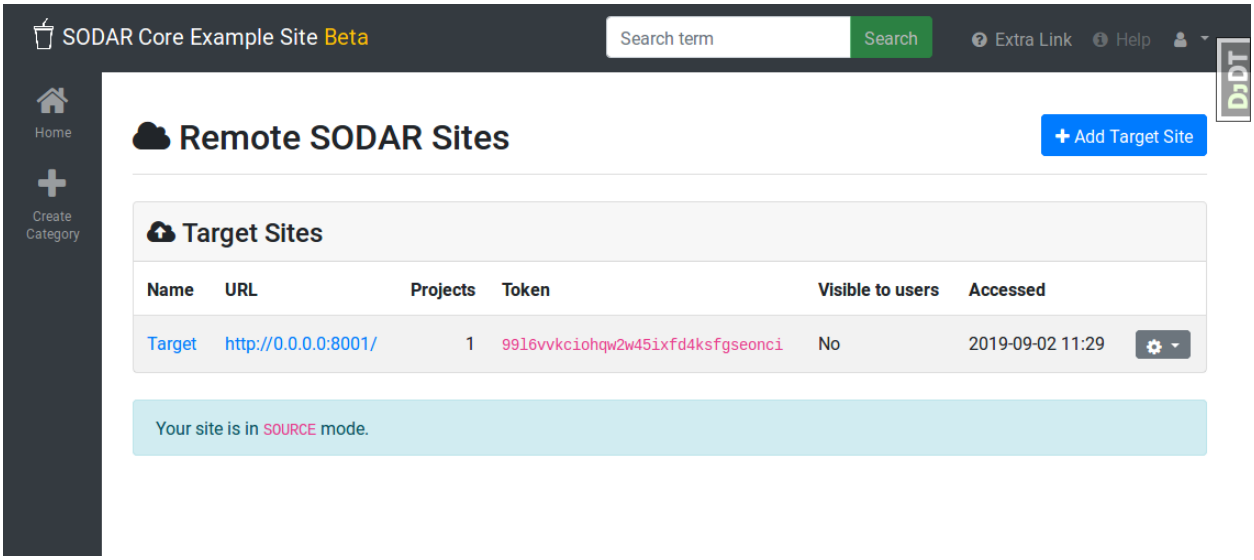


Fig. 9: Remote site list in source mode

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.

Here you also have the option to hide the remote project link from your users. Users viewing the project on the source site then won't see a link to the target site. Owners and Superusers will still see the link (greyed out). This is most commonly used for internal test sites which only needs to be used by admins.

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 three 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.
- **Revoked access**: Previously available access which has been revoked. The project will still remain in the target site, but only superusers, the project owner or the project delegate(s) can acesss it.

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.

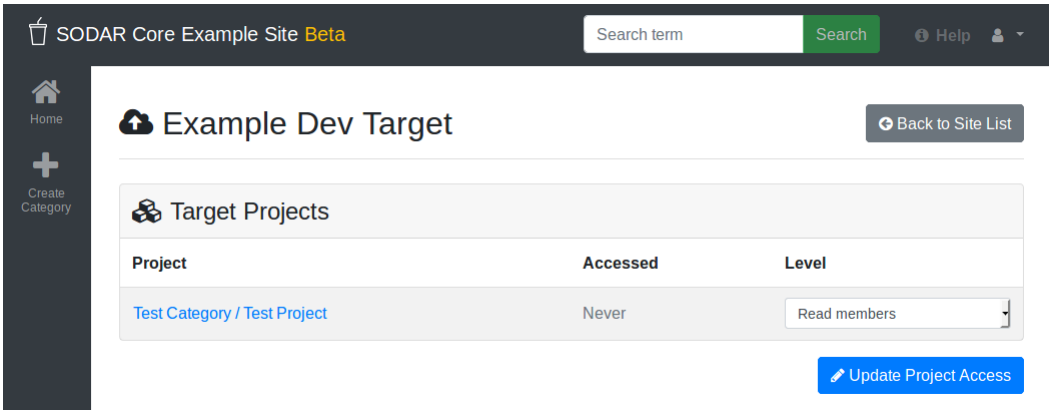


Fig. 10: Remote project list in source mode

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. Additionaly if the remote Source site is synchronized with multiple Target Sites, information about those other Target sites will be synchronized as well an displayed as *Peer Sites*.

Alternatively, the following management command can be used:

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

Note: Creating local projects under a category synchronized from a remote source site is **not** allowed from v0.8.3 onwards. For having local projects on a target site, you should create and use a local root category.

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 basic search form is displayed in the top navigation bar if enabled. It 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.

Left to the basic search form is a link to the *Advanced Search* page, where you can currently search for items using multiple search terms combined with the OR operator.

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: Additional features such as full-text search and more keywords/operators will be defined in the future.

REST API

Several SODAR Core functionalities are also available via a HTTP REST API starting in version 0.8. See *Projectroles REST API Documentation* for instructions on REST API usage.

5.5.5 Projectroles Customization

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

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 *Django API documentation* for details.

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

5.5.6 Projectroles REST API Documentation

This document contains the HTTP REST API documentation for the `projectroles` app. The provided API endpoints allow project and role operations through HTTP API calls in addition to the GUI.

API Usage

Usage of the REST API is detailed in this section. These instructions also apply to REST APIs in any other application within SODAR Core and are recommended as guidelines for API development in your SODAR Core based Django site.

Authentication

The API supports authentication through Knox authentication tokens as well as logging in using your SODAR username and password. Tokens are the recommended method for security purposes.

For token access, first retrieve your token using the *Tokens App*. Add the token in the `Authorization` header of your HTTP request as follows:

```
Authorization: token 90c2483172515bc8f6d52fd608e5031db3fcdc06d5a83b24bec1688f39b72bcd
```

Versioning

The SODAR Core REST API uses accept header versioning. While specifying the desired API version in your HTTP requests is optional, it is **strongly recommended**. This ensures you will get the appropriate return data and avoid running into unexpected incompatibility issues.

To enable versioning, add the `Accept` header to your request with the following media type and version syntax. Replace the version number with your expected version.

```
Accept: application/vnd.bihealth.sodar-core+json; version=0.9.0
```

Note: The media type and version for internal SODAR Core apps are by design intended to be different to applications implemented in your Django site. Only use the aforementioned values when calling REST API views in projectroles or other applications installed from the django-sodar-core package.

Model Access and Permissions

Objects in SODAR Core API views are accessed through their `sodar_uuid` field. This is strongly recommended for views implemented in your Django site as well, as using a field such as `pk` may reveal internal database details to users as well as be incompatible if e.g. mirroring roles between multiple SODAR Core sites.

In the remainder of this document and other REST API documentation, *"UUID"* refers to the `sodar_uuid` field of each model unless otherwise noted.

For permissions the API uses the same rules which are in effect in the SODAR Core GUI. That means you need to have appropriate project access for each operation.

Return Data

The return data for each request will be a JSON document unless otherwise specified.

If return data is not specified in the documentation of an API view, it will return the appropriate HTTP status code along with an optional `detail` JSON field upon a successfully processed request.

For creation views, the `sodar_uuid` of the created object is returned along with other object fields.

API Views

class projectroles.views_api.**ProjectListAPIView**(***kwargs*)
List all projects and categories for which the requesting user has access.

URL: `/project/api/list`

Methods: `GET`

Returns: List of project details (see `ProjectRetrieveAPIView`)

class projectroles.views_api.**ProjectRetrieveAPIView**(***kwargs*)
Retrieve a project or category by its UUID.

URL: `/project/api/retrieve/{Project.sodar_uuid}`

Methods: `GET`

Returns:

- `description`: Project description (string)
- `parent`: Parent category UUID (string or null)
- `readme`: Project readme (string, supports markdown)
- `roles`: Project role assignments (dict, assignment UUID as key)
- `sodar_uuid`: Project UUID (string)
- `submit_status`: Project creation status (string)
- `title`: Project title (string)
- `type`: Project type (string, options: `PROJECT` or `CATEGORY`)

class projectroles.views_api.**ProjectCreateAPIView**(***kwargs*)
Create a project or a category.

URL: `/project/api/create`

Methods: `POST`

Parameters:

- `title`: Project title (string)
- `type`: Project type (string, options: `PROJECT` or `CATEGORY`)
- `parent`: Parent category UUID (string)
- `description`: Projcet description (string, optional)
- `readme`: Project readme (string, optional, supports markdown)
- `owner`: User UUID of the project owner (string)

class projectroles.views_api.**ProjectUpdateAPIView**(***kwargs*)
Update the metadata of a project or a category.

Note that the project owner can not be updated here. Instead, use the dedicated API view `RoleAssignmentOwnerTransferAPIView`.

The project type can not be updated once a project has been created. The parameter is still required for non-partial updates via the `PUT` method.

URL: `/project/api/update/{Project.sodar_uuid}`

Methods: `PUT`, `PATCH`

Parameters:

- `title`: Project title (string)
- `type`: Project type (string, can not be modified)
- `parent`: Parent category UUID (string)
- `description`: Projcet description (string, optional)
- `readme`: Project readme (string, optional, supports markdown)
- `owner`: User UUID of the project owner (string)

class projectroles.views_api.**RoleAssignmentCreateAPIView**(***kwargs*)
Create a role assignment in a project.

URL: `/project/api/roles/create/{Project.sodar_uuid}`

Methods: `POST`

Parameters:

- `role`: Desired role for user (string, e.g. "project contributor")
- `user`: User UUID (string)

class projectroles.views_api.**RoleAssignmentUpdateAPIView**(***kwargs*)
Update the role assignment for a user in a project.

The user can not be changed in this API view.

URL: `/project/api/roles/update/{RoleAssignment.sodar_uuid}`

Methods: `PUT`, `PATCH`

Parameters:

- `role`: Desired role for user (string, e.g. "project contributor")
- `user`: User UUID (string)

class projectroles.views_api.**RoleAssignmentDestroyAPIView**(***kwargs*)
Destroy a role assignment.

The owner role can not be destroyed using this view.

URL: `/project/api/roles/destroy/{RoleAssignment.sodar_uuid}`

Methods: `DELETE`

class projectroles.views_api.**RoleAssignmentOwnerTransferAPIView**(***kwargs*)
Trensfer project ownership to another user with a role in the project. Reassign a different role to the previous owner.

The new owner must already have a role assigned in the project.

URL: `/project/api/roles/owner-transfer/{Project.sodar_uuid}`

Methods: `POST`

Parameters:

- `new_owner`: User name of new owner (string)
- `old_owner_role`: Role for old owner (string. e.g. "project delegate")

class projectroles.views_api.**ProjectInviteListAPIView**(***kwargs*)
List user invites for a project.

URL: `/project/api/invites/list/{Project.sodar_uuid}`

Methods: `GET`

Returns: List of project invite details

class projectroles.views_api.**ProjectInviteCreateAPIView**(***kwargs*)
Create a project invite.

URL: `/project/api/invites/create/{Project.sodar_uuid}`

Methods: `POST`

Parameters:

- `email`: User email (string)
- `role`: Desired role for user (string, e.g. "project contributor")

class projectroles.views_api.**ProjectInviteRevokeAPIView**(***kwargs*)
Revoke a project invite.

URL: `/project/api/invites/revoke/{ProjectInvite.sodar_uuid}`

Methods: `POST`

class projectroles.views_api.**ProjectInviteResendAPIView**(***kwargs*)
Resend email for a project invite.

URL: `/project/api/invites/resend/{ProjectInvite.sodar_uuid}`

Methods: `POST`

class projectroles.views_api.**UserListAPIView**(***kwargs*)
List users in the system.

URL: `/project/api/users/list`

Methods: `GET`

Returns:

For each user:

- `email`: Email address of the user (string)
- `name`: Full name of the user (string)
- `sodar_uuid`: User UUID (string)
- `username`: Username of the user (string)

class projectroles.views_api.**CurrentUserRetrieveAPIView**(***kwargs*)
Return information on the user making the request.

URL: `/project/api/users/current`

Methods: `GET`

Returns:

For current user:

- `email`: Email address of the user (string)
- `name`: Full name of the user (string)
- `sodar_uuid`: User UUID (string)
- `username`: Username of the user (string)

5.5.7 Projectroles Django API Documentation

This document contains the Django API documentation for the `projectroles` app. Included are functionalities and classes intended to be used by other applications when building a SODAR Core based Django site.

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*, *user*)
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
- **user** – User object (current user)

Returns String (may contain HTML), integer 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_terms*, *user*, *search_type=None*, *keywords=None*)
Return app items based on one or more search terms, user, optional type and optional keywords.

Parameters

- **search_terms** – Search terms to be joined with the OR operator (list of strings)
- **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'*, *custom_order=False*)
Return active plugins of a specific type.

Parameters

- **plugin_type** – "project_app", "site_app" or "backend" (string)
- **custom_order** – Order by plugin_ordering for project apps (boolean)

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*, ***kwargs*)
Return backend API object. NOTE: May raise an exception from plugin.get_api().

Parameters

- **plugin_name** – Plugin name (string)
- **force** – Return plugin regardless of status in ENABLED_BACKEND_PLUGINS
- **kwargs** – Optional kwargs for API

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.**AppSetting**(**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 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

user_modifiable
Setting visibility in forms

value
Value of the setting

value_json
Optional JSON value for the setting

class projectroles.models.**AppSettingManager**(**args*, ***kwargs*)
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 project and/or user must be set.

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

full_title

Full project title with parent path (auto-generated)

get_all_roles(*inherited=True*)

Return all RoleAssignments for the project, including inherited owner rights from parent categories.

Parameters **inherited** – Include inherited owners (bool, default=True)

Returns List

get_children(*flat=False*)

Return child objects for the Project sorted by title.

Parameters **flat** – Return all children recursively as a flat list (bool)

Returns Iterable of Project

get_delegates(*exclude_inherited=False*)

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 with path.

NOTE: Deprecated, will be removed in the next major release (#620) NOTE: Use Project.full_title instead!

get_members()

Return RoleAssignments for members of project excluding owner and delegates.

get_owner()

Return RoleAssignment for owner (without inherited owners) or None if not set.

get_owners(*inherited_only=False*)

Return RoleAssignments for project owner as well as possible inherited owners from parent projects.

Parameters **inherited_only** – Only show inherited owners if True (bool)

Returns List

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. Also return True if user inherits owner permissions from a parent category.

is_delegate(*user*)
Return True if user is delegate in this project.

is_owner(*user*)
Return True if user is owner in this project or inherits ownership from a parent category.

is_owner_or_delegate(*user*)
Return True if user is either an owner or a delegate in this project. Includes inherited owner relationships.

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

is_revoked()
Return True if remote access has been revoked for the project

parent
Parent category if nested, otherwise null

readme
Project README (optional, supports markdown)

save(**args*, ***kwargs*)
Custom validation and field populating 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**(**args*, ***kwargs*)
Manager for custom table-level Project queries

find(*search_terms*, *keywords=None*, *project_type=None*)
Return projects with a partial match in full title or, including titles of parent Project objects, or the description of the current object. Restrict to project type if project_type is set.

Parameters

- **search_terms** – Search terms (list)
- **keywords** – Optional search keywords as key/value pairs (dict)
- **project_type** – Project type or None

Returns QuerySet 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

user_display
RemoteSite's link visibilty for users

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 assignements 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**(**args*, ***kwargs*)
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. Use this on your SODAR Core based site.

get_full_name()
Return full name or username if not set

save(**args*, ***kwargs*)
Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

set_group()
Set user group based on user name.

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 delete_setting(*app_name*, *setting_name*, *project=None*, *user=None*)
Delete app setting.

Parameters

- **app_name** – App name (string, must equal "name" in app plugin)
- **setting_name** – Setting name (string)
- **project** – Project object to delete setting from (optional)

- **user** – User object to delete setting from (optional)

classmethod get_all_defaults(*scope*, *post_safe=False*)
Get all default settings for a scope.

Parameters

- **scope** – Setting scope (PROJECT, USER or PROJECT_USER)
- **post_safe** – Whether POST safe values should be returned (bool)

Returns Dict

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

Parameters

- **project** – Project object (optional)
- **user** – User object (optional)
- **post_safe** – Whether POST safe values should be returned (bool)

Returns Dict

Raise ValueError if neither project nor user are set

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

Parameters

- **app_name** – App name (string, must equal "name" in app plugin)
- **setting_name** – Setting name (string)
- **project** – Project object (optional)
- **user** – User object (optional)
- **post_safe** – Whether a POST safe value should be returned (bool)

Returns String or None

Raise KeyError if nothing is found with setting_name

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

Parameters

- **app_name** – App name (string, must equal "name" in app plugin)
- **setting_name** – Setting name (string)
- **post_safe** – Whether a POST safe value should be returned (bool)

Returns Setting value (string, integer or boolean)

Raise ValueError if app plugin is not found

Raise KeyError if nothing is found with setting_name

classmethod get_setting_def(*name*, *plugin=None*, *app_name=None*)
Return definition for a single app setting, either based on an app name or the plugin object.

Parameters

- **name** – Setting name
- **plugin** – Plugin object extending ProjectAppPluginPoint
- **app_name** – Name of the app plugin (string)

Returns Dict

Raise ValueError if neither app_name or plugin are set or if setting is not found in plugin

classmethod get_setting_defs(*scope*, *plugin=False*, *app_name=False*, *user_modifiable=False*)

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

Parameters

- **scope** – PROJECT, USER or PROJECT_USER
- **plugin** – project app plugin object extending ProjectAppPluginPoint
- **app_name** – Name of the app plugin (string)
- **user_modifiable** – Only return modifiable settings if True (boolean)

Returns Dict

Raise ValueError if scope is invalid or if if neither app_name or plugin are set

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 equal "name" in app plugin)
- **setting_name** – Setting name (string)
- **value** – Value to be set
- **project** – Project object (optional)
- **user** – User object (optional)
- **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*, *setting_options*)

Validate setting value according to its type.

Parameters

- **setting_type** – Setting type
- **setting_value** – Setting value
- **setting_options** – Setting options (can be None)

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.templatetags.projectroles_common_tags.**check_backend**(*name*)
Return True if backend app is available, else False

projectroles.templatetags.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_app_setting**(*app_name*, *setting_name*, *project=None*, *user=None*)
Get a project/user specific app setting from AppSettingAPI

projectroles.templatetags.projectroles_common_tags.**get_backend_include**(*backend_name*, *include_type='js'*)
Returns import string for backend app Javascript or CSS. Returns empty string if not found.

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_django_setting**(*name*, *default=None*, *js=False*)
Return value of Django setting by name or the default value if the setting is not found. Return a Javascript-safe value if js=True.

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_plugin_name_by_id**(*plugin_id*)
Return Plugin by id

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_role_display_name**(*role_as*, *title=False*)
Return display name for role assignment

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.**get_visible_projects**(*projects*, *can_view_hidden_projects=*
Return all projects that are either visible by user display or by view hidden permission

projectroles.templatetags.projectroles_common_tags.**highlight_search_term**(*item*, *terms*)
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.templatetags.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

Base REST API View Classes

Base view classes and mixins for building REST APIs can be found in `projectroles.views_api`.

Permissions / Versioning / Rendering

class projectroles.views_api.**SODARAPIProjectPermission**
Bases: projectroles.views.ProjectAccessMixin, rest_framework.permissions.BasePermission

Mixin for providing a basic project permission checking for API views with a single permission_required attribute. Also works with Knox token based views.

This must be used in the permission_classes attribute in order for token authentication to work.

Requires implementing either permission_required or get_permission_required() in the view.

has_permission(*request*, *view*)
Override has_permission() for checking auth and project permission

class projectroles.views_api.**SODARAPIVersioning**
Bases: rest_framework.versioning.AcceptHeaderVersioning

Accept header versioning class for SODAR API views

class projectroles.views_api.**SODARAPIRenderer**
Bases: rest_framework.renderers.JSONRenderer

SODAR API JSON renderer with a site-specific media type retrieved from Django settings

Base API View Mixins

class projectroles.views_api.**SODARAPIBaseMixin**
Base SODAR API mixin to be used by external SODAR Core based sites

versioning_class
alias of *projectroles.views_api.SODARAPIVersioning*

class projectroles.views_api.**SODARAPIBaseProjectMixin**
Bases: projectroles.views.ProjectAccessMixin, *projectroles.views_api.SODARAPIBaseMixin*

API view mixin for the base DRF APIView class with project permission checking, but without serializers and other generic view functionality.

class projectroles.views_api.**APIProjectContextMixin**
Bases: projectroles.views.ProjectAccessMixin

Mixin to provide project context and queryset for generic API views. Can be used both in SODAR and SODAR Core API base views.

class projectroles.views_api.**SODARAPIGenericProjectMixin**
Bases: *projectroles.views_api.APIProjectContextMixin*, *projectroles.views_api.SODARAPIBaseProjectMixin*

API view mixin for generic DRF API views with serializers, SODAR project context and permission checkin.

Unless overriding permission_classes with their own implementation, the user MUST supply a permission_required attribute.

Replace lookup_url_kwarg with your view's url kwarg (SODAR project compatible model name in lowercase)

If the lookup is done via the project object, change lookup_field into "sodar_uuid"

class projectroles.views_api.**ProjectQuerysetMixin**
Mixin for overriding the default queryset with one which allows us to lookup a Project object directly.

Base Ajax API View Classes

Base view classes and mixins for building Ajax API views can be found in projectroles.views_ajax.

class projectroles.views_ajax.**SODARBaseAjaxView**(***kwargs*)
Bases: rest_framework.views.APIView

Base Ajax view with Django session authentication.

No permission classes or mixins used, you will have to supply your own if using this class directly.

class projectroles.views_ajax.**SODARBasePermissionAjaxView**(***kwargs*)
Bases: rules.contrib.views.PermissionRequiredMixin, *projectroles.views_ajax.SODARBaseAjaxView*

Base Ajax view with permission checks, to be used e.g. in site apps with no project context.

User-based perms such as is_superuser can be used with this class.

handle_no_permission()
Override handle_no_permission() to provide 403

class projectroles.views_ajax.**SODARBaseProjectAjaxView**(***kwargs*)
Bases: projectroles.views.ProjectAccessMixin, *projectroles.views_ajax.SODARBaseAjaxView*

Base Ajax view with SODAR project permission checks

Base Serializers

Base serializers for SODAR Core compatible models are available in projectroles.serializers.

class projectroles.serializers.**SODARModelSerializer**(**args*, ***kwargs*)
Bases: rest_framework.serializers.ModelSerializer

Base serializer for any SODAR model with a sodar_uuid field

post_save(*obj*)
Function to call at the end of a custom save() method. Ensures the returning of sodar_uuid in object creation POST responses.

Parameters **obj** – Object created in save()

Returns obj

save(***kwargs*)
Override save() to ensure sodar_uuid is included for object creation POST responses.

to_representation(*instance*)
Override to_representation() to ensure sodar_uuid is included for object creation POST responses.

class projectroles.serializers.**SODARProjectModelSerializer**(**args*, ***kwargs*)
Bases: *projectroles.serializers.SODARModelSerializer*

Base serializer for SODAR models with a project relation.

The project field is read only because it is retrieved through the object reference in the URL.

create(*validated_data*)
Override create() to add project into validated data

to_representation(*instance*)
Override to_representation() to ensure the project value is included in responses.

class projectroles.serializers.**SODARNestedListSerializer**(**args*, ***kwargs*)
Bases: *projectroles.serializers.SODARModelSerializer*

Serializer to display nested SODAR models as dicts with sodar_uuid as key.

to_representation(*instance*)
Override to_representation() to pop project from a nested list representation, where the project context is already known in the topmost model.

class projectroles.serializers.**SODARUserSerializer**(**args*, ***kwargs*)
Bases: *projectroles.serializers.SODARModelSerializer*

Serializer for the user model used in SODAR Core based sites

5.6 Adminalerts App

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

5.6.1 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.

5.6.2 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
```

5.6.3 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.

5.7 Bgjobs App

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

TODO: Docs to be filled out

5.7.1 Bgjobs Installation

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

> **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

5.7.2 Bgjobs Usage

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

TODO

5.8 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.

5.8.1 Filesfolders Installation

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

> **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
```

5.8.2 Filesfolders Usage

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

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.

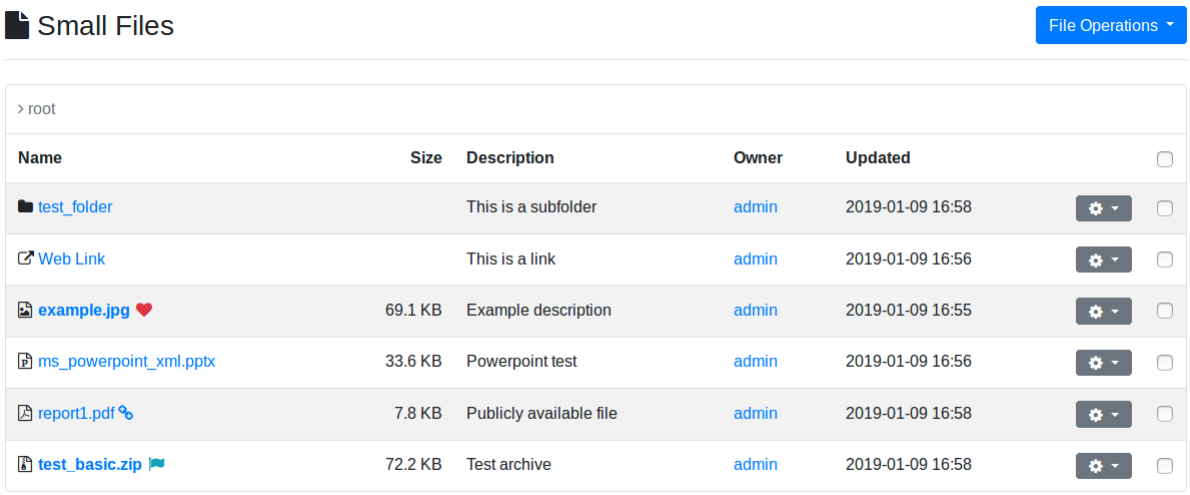


Fig. 11: Filesfolders main view

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.

5.8.3 Filesfolders REST API Documentation

This document contains the HTTP REST API documentation for the `filesfolders` app. The provided API endpoints allow file operations through HTTP API calls in addition to the GUI.

For general information on REST API usage in SODAR Core, see *Projectroles REST API Documentation*.

class filesfolders.views_api.**FolderListCreateAPIView**(***kwargs*)
List folders or create a folder.

URL: `/files/api/folder/list-create/{Project.sodar_uuid}`

Methods: `GET`, `POST`

Parameters for POST:

- `name`: Folder name (string)
- `folder`: Parent folder UUID (string)
- `owner`: User UUID of folder owner (string)
- `flag`: Folder flag (string, optional)
- `description`: Folder description (string, optional)

class filesfolders.views_api.**FolderRetrieveUpdateDestroyAPIView**(***kwargs*)
Retrieve, update or destroy a folder.

URL: `/files/api/folder/retrieve-update-destroy/{Folder.sodar_uuid}`

Methods: `GET`, `PUT`, `PATCH`, `DELETE`

Parameters for PUT and PATCH:

- `name`: Folder name (string)
- `folder`: Parent folder UUID (string)
- `owner`: User UUID of folder owner (string)
- `flag`: Folder flag (string, optional)
- `description`: Folder description (string, optional)

class filesfolders.views_api.**FileListCreateAPIView**(***kwargs*)
List files or upload a file. For uploads, the request must be made in the `multipart` format.

URL: `/files/api/file/list-create/{Project.sodar_uuid}`

Methods: `GET`, `POST`

Parameters for POST:

- `name`: Folder name (string)
- `folder`: Parent folder UUID (string)
- `owner`: User UUID of folder owner (string)

- `flag`: Folder flag (string, optional)
- `description`: Folder description (string, optional)
- `public_url`: Allow creation of a publicly viewable URL (bool)
- `file`: File to be uploaded

class filesfolders.views_api.**FileRetrieveUpdateDestroyAPIView**(***kwargs*)
Retrieve, update or destroy a file.

URL: `/files/api/file/retrieve-update-destroy/{File.sodar_uuid}`

Methods: `GET`, `PUT`, `PATCH`, `DELETE`

Parameters for PUT and PATCH:

- `name`: Folder name (string)
- `folder`: Parent folder UUID (string)
- `owner`: User UUID of folder owner (string)
- `flag`: Folder flag (string, optional)
- `description`: Folder description (string, optional)
- `public_url`: Allow creation of a publicly viewable URL (bool)
- `file`: File to be uploaded

class filesfolders.views_api.**FileServeAPIView**(***kwargs*)
Serve the file content.

URL: `/files/api/file/serve/{File.sodar_uuid}`

Methods: `GET`

class filesfolders.views_api.**HyperLinkListCreateAPIView**(***kwargs*)
List hyperlinks or create a hyperlink.

URL: `/files/api/hyperlink/list-create/{Project.sodar_uuid}`

Methods: `GET`, `POST`

Parameters for POST:

- `name`: Folder name (string)
- `folder`: Parent folder UUID (string)
- `owner`: User UUID of folder owner (string)
- `flag`: Folder flag (string, optional)
- `description`: Folder description (string, optional)
- `url`: URL for the link (string)

class filesfolders.views_api.**HyperLinkRetrieveUpdateDestroyAPIView**(***kwargs*)
Retrieve, update or destroy a hyperlink.

URL: `/files/api/hyperlink/retrieve-update-destroy/{HyperLink.sodar_uuid}`

Methods: `GET`, `PUT`, `PATCH`, `DELETE`

Parameters for PUT and PATCH:

- `name`: Folder name (string)

- `folder`: Parent folder UUID (string)
- `owner`: User UUID of folder owner (string)
- `flag`: Folder flag (string, optional)
- `description`: Folder description (string, optional)
- `url`: URL for the link (string)

5.9 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.

5.9.1 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
```

5.9.2 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.

5.9.3 User Settings

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

5.10 Siteinfo App

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

5.10.1 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.

5.10.2 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
```

5.10.3 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'
        }
    }
```

5.11 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.

5.11.1 Sodarcache Installation

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

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
```

5.11.2 Sodar Cache Usage

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

Backend API for Data Caching

The Django backend API for caching data is located in `sodarcache.api`. For the full 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
```

5.11.3 Sodarcache Backend API Documentation

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

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

5.12 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.

5.12.1 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.

5.12.2 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',
```

(continues on next page)

(continued from previous page)

```
    # ..
]
```

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
```

5.12.3 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
```

5.12.4 Django 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

classmethod get_inherited_roles(*project*, *user*, *roles=None*)
Return list of inherited owner roles to be used in taskflow sync.

Parameters

- **project** – Project object
- **user** – User object

Pram roles Previously collected roles (optional, list or None)

Returns List of dicts

classmethod get_inherited_users(*project*, *roles=None*)
Return list of all inherited users within a project and its children, to be used in taskflow sync.

Parameters **project** – Project object

Pram roles Previously collected roles (optional, list or None)

Returns List of dicts

submit(*project_uuid*, *flow_name*, *flow_data*, *request=None*, *targets=None*, *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

5.13 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.

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 timeline events for projects, categories and objects. Also included is a backend API for saving desired activity as timeline events. For details on how to use these, see the *timeline usage documentation*.

5.13.1 Timeline Installation

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

> **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
```

5.13.2 Timeline Usage

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

Timeline UI

You can browse events by navigating to a project or a category 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.

Certain events have a file icon in their description. If clicked, a popup showing a collection of extra data for the given event will appear. It will show the extra data of the event itself and of the different states the event went through, if there are any.

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

Backend API for Event Logging

The Django backend API for logging events is located in `timeline.api`. For the full documentation, see here.

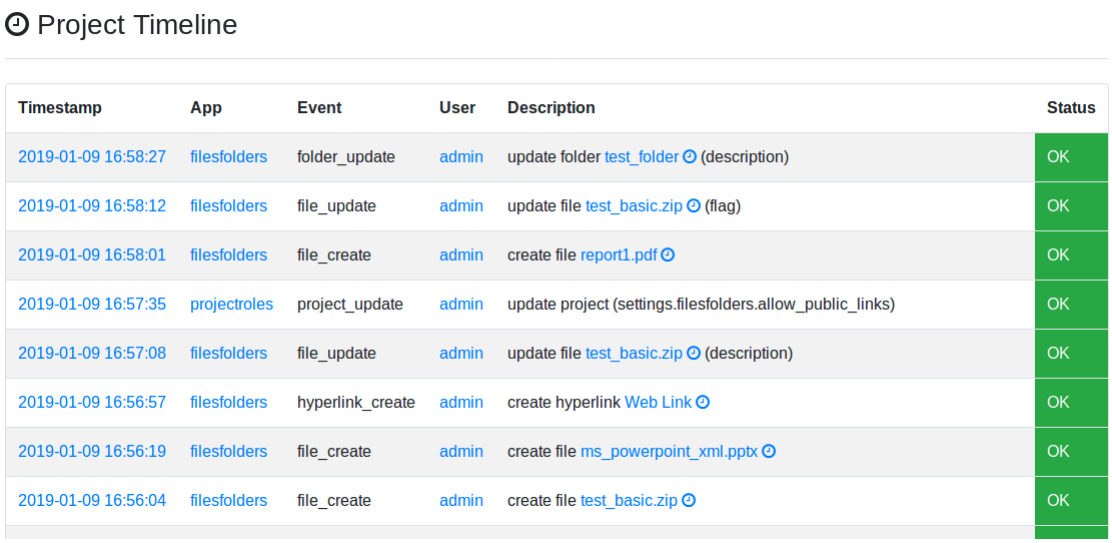


Fig. 12: Timeline event list view

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.

Speciying 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.

5.13.3 Timeline Django API Documentation

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

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_models()

Return project event model classes for custom/advanced queries.

Returns ProjectEvent, ProjectEventObjectRef

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**(**args*, ***kwargs*)
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

5.14 Tokens App

The `tokens` site app enables users to issue and manage access tokens for REST API views used on your SODAR Core based Django site.

5.14.1 Basics

Users can use this app to create and delete access tokens. These can be set to expire or work until deleted.

5.14.2 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 = [
    # ...
    'tokens.apps.TokensConfig',
]
```

URL Configuration

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

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

Register Plugin

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

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

You should see the following output:

```
Registering Plugin for tokens.plugins.SiteAppPlugin
```

5.14.3 Usage

When logged in to SODAR, you can find the "API Tokens" link in your user dropdown menu in the top right corner of the site.

Select "Create Token" from the "Token Operations" dropdown to create a new token. You will only see the token once, so make sure to copy it to clipboard at this point.

Deleting existing tokens can be done from the token list.

5.15 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.15.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/bihealth/sodar_core/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

SODAR Core could always use more documentation, whether as part of the official SODAR Core docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/bihealth/sodar_core/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.15.2 Get Started!

Ready to contribute? Here's how to set up `sodar_core` for local development.

1. Fork the `sodar_core` repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/sodar_core.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv sodar_core
$ cd sodar_core/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature dev
```

   Make sure you base your changes on the `dev` branch, which is the current active development branch. The `master` branch is intended for merging stable releases only. Now you can make your changes locally.

5. When you're done making changes, make sure to apply proper formatting using Black and the settings specified in the accompanying `black.sh` script. Next, check that your changes pass flake8 and the tests. It is recommended to use the accompanying `test.sh` script to ensure the correct Django configuration is used. For testing other Python versions use tox:

```
$ ./black.sh
$ flake8 .
$ ./test.sh
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.15.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. Make sure your pull request is up to date with the `dev` branch.
2. The pull request should include tests.
3. Black and flake8 should have been executed without errors using settings provided in the repo.
4. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in `CHANGELOG.rst`.
5. The pull request should work for Python 3.6 and preferably for 3.7. Check https://github.com/bihealth/sodar_core/actions and make sure that the tests pass for supported Python versions. The 1.11 branch of Django does not currently support Python 3.8.

5.15.4 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in `CHANGELOG.rst`). Then run:

```
$ git tag vX.Y.Z
$ git push
$ git push --tags
$ python setup.py sdist
$ twine upload --repository-url https://test.pypi.org/legacy/ dist/*.tar.gz
$ twine upload dist/*.tar.gz
```

5.16 Code of Conduct

Everyone interacting in the SODAR Core project's codebases, issue trackers, chat rooms, and mailing lists is expected to follow the PyPA Code of Conduct.

5.17 Glossary

App Plugin Mechanism for defining common properties and operations for dynamically including content and functionality from apps in SODAR Core views.

App Settings Project or user specific settings defined in SODAR Core app plugins. Different from e.g. Django settings used to configure the web site.

Backend App SODAR Core application which is used to provide additional functionality to other SODAR Core apps. Does not have its own GUI entry point. Common use cases include APIs to external services or other apps.

Backend API Django API provided by a backend app, to be dynamically imported and used by other SODAR Core based Django apps.

Django API Application programming interface offered by an app, to be used by other apps within the Django site.

Django App Application built for the Django web framework, including (but not limited to) SODAR Core based apps.

Django Settings Django settings used to configure the website. SODAR Core apps also use Django settings for configuring framework and app behaviour.

Django Site Web site built on the Django framework, including (but not limited to) any website based on SODAR Core.

Peer Site A SODAR Core based web site which mirrors one or more projects also mirrored on the currrently active site. This allows linking to remote projects on other sites where the user would have access.

Project App SODAR Core application with the scope of providing data and functionality related to a specific project. Uses project-based access control.

SODAR System for Omics Data Access and Retrieval. An omics research data management system which is the origin of the reusable SODAR Core framework.

SODAR Core Core framework and reusable apps originally built for the SODAR project.

SODAR Core App Django application with additional SODAR Core features. This includes one or more app plugin definitions to enable dynamic inclusion of the app into the SODAR Core framework, as well as project access control for project apps.

SODAR Core Based Site Django-based web site using SODAR Core apps as its framework.

Site App SODAR Core application with does not limit its scope to a single project. Common use cases include user account management and administrative tools.

Source Site SODAR Core based web site which mirrors project metadata and access control to "target" sites.

Target Site SODAR Core based web site which mirrors project metadata and access control from a "source" site.

5.18 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.

5.18.1 General Development Topics

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

Common Helpers

Via the projectroles app, SODAR Core provides optional templates for aiding in maintaining common functionality and layout. Those are defined here.

App Setting API

For accessing and modifying app settings for project or site apps, you should use the `AppSettingAPI`. Below is an example of invoking the API. For the full API docs, see *Projectroles Django API Documentation*.

```
from projectroles.app_settings import AppSettingAPI
app_settings = AppSettingAPI()
app_settings.get_app_setting('app_name', 'setting_name', project_object)  # Etc..
```

Form Base Classes

Although not required, it is recommended to use common SODAR Core base classes with built-in helpers for your Django forms. `SODARForm` and `SODARModelForm` extend Django's `Form` and `ModelForm` respectively. These base classes can be imported from `projectroles.forms`. Currently they add logging to `add_error()` calls, which helps administrators track form issues encountered by users. Further improvements are to be added in the future.

Pagination Template

A common template for adding navigation for list pagination can be found in `projectroles/_pagination.html`. This can be included to any Django `ListView` template which provides the `paginate_by` definition, enabling pagination. If a smaller layout is desired, the `pg_small` argument can be used. An example can be seen below:

```
{% include 'projectroles/_pagination.html' with pg_small=True %}
```

Testing

SODAR Core provides a range of ready made testing classes and mixins for different aspects of SODAR app testing, from user permissions to UI testing. See `projectroles.tests` for different base classes.

Test Settings

SODAR Core provides settings for configuring your UI tests, if using the base UI test classes found in `projectroles.tests.test_ui`. Default values for these settings can be found in `config/settings/test.py`. The settins are as follows:

- `PROJECTROLES_TEST_UI_CHROME_OPTIONS`: Options for Chrome through Selenium. Can be used to e.g. enable/disable headless testing mode.
- `PROJECTROLES_TEST_UI_WINDOW_SIZE`: Custom browser window size.
- `PROJECTROLES_TEST_UI_WAIT_TIME`: Maximum wait time for UI test operations
- `PROJECTROLES_TEST_UI_LEGACY_LOGIN`: If set `True`, use the legacy UI login and redirect function for testing with different users. This can be used if e.g. issues with cookie-based logins are encountered.

5.18.2 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.

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, user or project_user (Settings specific to a project and 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.
- `category_enable`: Whether the app should also be made available for categories. Defaults to `False` and should only be overridden when required. For an example of a project app enabled in categories, see *Timeline*.
- `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 databse 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 #}
  <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.

SODAR User Selection Field

Projectroles offers a custom field, widget and accompanying Ajax API views for autocomplete-enabled selection of SODAR users in Django forms. The field will handle providing appropriate choices according to the view context and user permissions, also allowing for customization.

The recommended way to use the built-in user form field is by using the `SODARUserChoiceField` class found in `projectroles.forms`. The field extends Django's `ModelChoiceField` and takes most of the same keyword arguments in its init function, with the exception of `queryset`, `to_field_name`, `limit_choices_to` and `widget` which will be overridden.

The init function also takes new arguments which are specified below:

- **`scope`: Scope of users to include (string)**
  - `all`: All users on the site
  - `project`: Limit search to users in given project
  - `project_exclude` Exclude existing users of given project
- `project`: Project object or project UUID string (optional)
- `exclude`: List of User objects or User UUIDs to exclude (optional)
- `forward`: Parameters to forward to autocomplete view (optional)

- `url`: Autocomplete ajax class override (optional)
- `widget_class`: Widget class override (optional)

Below is an example of the classes usage. Note that you can also define the field as a form class member, but the `project` or `exclude` values are not definable at that point. The following example assumes you are setting up your project app form with an extra `project` argument.

```
from projectroles.forms import SODARUserChoiceField

class YourForm(forms.ModelForm):
    class Meta:
        # ...
    def __init__(self, project, *args, **kwargs):
        # ...
        self.fields['user'] = SODARUserChoiceField(
            label='User',
            help_text='Select user for your thing here',
            required=True,
            scope='project',
            project=project,
            exclude=[unwanted_user]
        )
```

For more examples of usage of this field and its widget, see `projectroles.forms`. If the field class does not suit your needs, you can also retrieve the related widget to your own field with `projectroles.forms.get_user_widget()`.

The following `django-autocomplete-light` and `select2` CSS and Javascript links have to be added to the HTML template that includes the form with your user selection field:

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

{% 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 %}
```

If using a customized widget with its own Javascript, include the corresponding JS file instead of `autocomplete_light/select2.js`. See 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_terms`**
  - One or more terms to be searched for (list of strings). Expected to be combined with OR operators in your search logic.
  - Multiple search terms or phrases containing whitespaces can be provided via the Advanced Search view.
- **`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 seaching user yourself!

Warning: The old expected signature of providing a single `search_term` argument has been deprecated in v0.9 and will be removed in the next major release!

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': {                        # 1-N categories to be included
        'title': 'List title',      # Title of the result list to be displayed
        'search_types': [],         # Object types included in this category
        'items': []                 # 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 %}

   {# 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>
```

(continues on next page)

(continued from previous page)

```
      {% 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 strucure, 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', {
            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 in project apps is detailed in this section.

Rest API Views

To set up REST API views for project apps, it is recommended to use the base SODAR API view classes and mixins found in `projectroles.views_api`. These set up the recommended authentication methods, versioning through accept headers and project-based permission checks.

By default, the REST API views built on SODAR Core base classes support two methods of authentication: Knox tokens and Django session auth. These can of course be modified by overriding/extending the base classes.

For versioning we strongly recommend using accept header versioning, which is what is supported by the SODAR Core base classes. For this, supply your custom media type and version data using the corresponding `SODAR_API_*` settings. For details on these, see *Projectroles Django Settings*.

The base classes provide permission checks via SODAR Core project objects similar to UI view mixins.

Base REST API classes without a project context can also be used in site apps.

API documentation for each available base class and mixin for REST API views can be found in *Projectroles Django API Documentation*.

An example "hello world" REST API view for SODAR apps is available in `example_project_app.views.HelloExampleProjectAPIView`.

Note: Internal SODAR Core REST API views, specifically ones used in apps provided by the django-sodar-core package, use different media type and versioning from views to be implemented on your site. This is to prevent version number clashes and not require changes from your API when SODAR Core is updated.

Ajax API Views

To set up Ajax API views for the UI, it is recommended to use the base Ajax view classes found in `projectroles.views_ajax`. These views only support Django session authentication by default, so Knox token authentication will not work. Versioning is omitted. Base views without project permission checks can also be used in site apps.

API documentation for the base classes Ajax API views can be found in *Projectroles Django API Documentation*.

Example:

```
from projectroles.views_api import SODARBaseProjectAjaxView

class ExampleAjaxAPIView(SODARBaseProjectAjaxView):

permission_required = 'projectroles.view_project'

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

Serializers

Base serializers for SODAR Core based API views are available in `projectroles.serializers`. They provide `Project` context where needed, as well as setting default fields such as `sodar_uuid` which should be always used in place of `pk`.

API documentation for the base serializers can be found in *Projectroles Django API Documentation*.

Removing a Project App

Removing a project app from your Django site can be slightly more complicated than removing a normal non-SODAR-supporting Django application. Following the procedure detailed here you are able to cleanly remove a project app which has been in use on your site.

The instructions apply to project apps you have created yourself as well as project apps included in the django-sodar-core package, with the exception of `projectroles` which can not be removed from a SODAR based site.

> **Warning:** Make sure to perform these steps **in the order they are presented here**. Otherwise you may risk serious problems with your site functionality or your database!

Note: Just in case, it is recommended to make a backup of your Django database before proceeding.

First you should delete all Timeline references to objects in your app. This is not done automatically as, by design, the references are kept even after the original objects are deleted. Go to the Django shell via management command using `shell` or `shell_plus` and enter the following. Replace `app_name` with the name of your application as specified in its `ProjectAppPlugin`.

```
from timeline.models import ProjectEvent
ProjectEvent.objects.filter(app='app_name').delete()
```

Next you should delete existing database objects defined by the models in your app. This is also most easily done via the Django shell. Example:

```
from yourapp.models import YourModel
YourModel.objects.all().delete()
```

After the objects have been deleted, reset the database migrations of your application.

```
$ ./manage.py migrate yourapp zero
```

Once this has been executed successfully, you should delete the plugin object for your application. Returning to the Django shell, type the following:

```
from djangoplugins.models import Plugin
Plugin.objects.get(name='app_name').delete()
```

Finally, you should remove the references to the removed app in the Django configuration.

App dependency in `config/settings/base.py`:

```
LOCAL_APPS = [
# The app you are removing
'yourapp.apps.YourAppConfig',
```

(continues on next page)

(continued from previous page)

```
# ...
]
```

App URL patterns in `config/urls.py`:

```
urlpatterns = [
    # Your app's URLs
    url(r'^yourapp/', include('yourapp.urls')),
    # ...
]
```

Once you have performed the aforementioned database operations and deployed a version of your Django site with the application dependency and URL patterns removed, the project app should be cleanly removed from your site.

TODO

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

5.18.3 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.

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"""
    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**:

- `app_settings`: Implement if project or user specific settings for the app are needed. See the plugin point definition for an example.
- `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(LoggedInPermissionMixin, TemplateView):
```

(continues on next page)

(continued from previous page)

```
    """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 %}

{% 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 [{
```

(continues on next page)

(continued from previous page)

```
        '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
    }]
```

5.18.4 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.

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**:

- `javascript_url`: Path to on demand includeable Javascript file
- `css_url`: Path to on demand includeable CSS file
- `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
```

Including Backend Javascript/CSS

If you want Javascript or CSS files to be associated with your plugin you can set the `javascript_url` or `css_url` variables to specify the path to your file. Note that these should correspond to `STATIC` paths under your app directory.

```
class BackendPlugin(BackendPluginPoint):

    name = 'example_backend_app'
    title = 'Example Backend App'
    javascript_url = 'example_backend_app/js/example.js'
    css_url = 'example_backend_app/css/example.css'
```

The `get_backend_include` template-tag will return a `<script>` or `<link>` html tag with your specific file path, to be used in a template of your app making use of the backend plugin:

```
{% load projectroles_common_tags %}
{% get_backend_include 'example_backend_app' 'js' as javascript_include_tag %}
{{ javascript_include_tag|safe }}

{% get_backend_include 'example_backend_app' 'css' as css_include_tag %}
{{ css_include_tag|safe }}
```

This will result in the following HTML:

```
<script type="text/javascript" src="/static/example.js"></script>
<link rel="stylesheet" type="text/css" href="/static/example.css"/>
```

Be sure to use the backend plugin's name (not the title) as the key and filter the result by `safe`, so the tag won't be auto-escaped.

5.18.5 SODAR Core Development

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

Repository Contents

In addition to the apps which will be installed by the package, the following directories are included in the repository for development use 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 SODAR Core based Django site for development
- **example_site_app**: Example SODAR Core compatible site-wide app
- **requirements**: Requirements for SODAR Core and development
- **utility**: Setup scripts for development

Installation

Instructions on how to install a local development version of SODAR Core are detailed here. Ubuntu 16.04 LTS (Xenial) is the supported OS at this time. Later Ubuntu versions and Centos 7 have also been proven to to work, but some system dependencies may vary for different OS versions or distributions.

Installation and development should be possible on most recent versions of Linux, Mac and Windows, but this may require extra work and your mileage may vary.

If you need to set up the accompanying example site in Docker, please see online for up-to-date Docker setup tutorials for Django related to your operating system of choice.

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

Next you need to setup the database and postgres user. You'll be prompted to enter a database name, a username and a password.

```
$ sudo utility/setup_database.sh
```

You have to set the database URL and credentials for Django in the environment variable `DATABASE_URL`. For development it is recommended to place environment variables in file `.env` located in your project root. To enable loading the file in Django, set `DJANGO_READ_DOT_ENV_FILE=1` in your environment variables when running SODAR or any of its management commands. See `config/settings/base.py` for more information and the `env.example` file for an example environment file.

Example of the database URL variable as set within an `.env` file:

```
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 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:

```
$ make serve
```

App Development

Guidelines for developing **internal** SODAR Core apps (ones included when installing the django-sodar-core package) are detailed in this section.

REST API Views

For internal SODAR Core apps, you need to use core counterparts to the mixins than provided for SODAR Core using sites. The counterparts use different media type and versioning from views to be implemented on external sites. This is to prevent version number clashes requiring changes in external APIs. The classes can be found in `projectroles.views_api` and are as follows:

- `CoreAPIVersioning`
- `CoreAPIRenderer`
- `CoreAPIBaseMixin`
- `CoreAPIBaseProjectMixin`
- `CoreAPIGenericProjectMixin`

For detailed API descriptions, see docstrings in the `view_api` module. The media type and versioning for these views are **hardcoded** and should not be changed, except version information upon a new release of SODAR Core.

Projectroles App Development

This section details issues regarding updates to the `projectroles` app.

> **Warning:** As all other apps in SODAR Core as well as sites implementing SODAR Core are based on projectroles, changes to this app need to be implemented and tested with extra care. Also make sure to provide detailed documentation for all breaking changes.

Projectroles App Settings

Projectroles defines its own app settings in `projectroles/app_settings.py`. These are not expected to be altered by SODAR Core based sites. These settings add the `local` attribute, which allows/disallows editing the value on a `TARGET` site.

To alter projectroles app settings when developing the app, update the `PROJECTROLES_APP_SETTINGS` dictionary as follows:

```
'example_setting': {
    'scope': 'PROJECT',  # PROJECT/USER
    'type': 'STRING',  # STRING/INTEGER/BOOLEAN
    'default': 'example',
    'options': ['example', 'example2'],  # Optional, only for settings of type STRING␣
↪or INTEGER
    'label': 'Project setting',  # Optional, defaults to name/key
    'placeholder': 'Enter example setting here',  # Optional
    'description': 'Example project setting',  # Optional
    'user_modifiable': True,  # Optional, show/hide in forms
    'local': False,  # Allow editing in target site forms if True
}
```

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 make command:

```
$ make test
```

If you want to only run a certain subset of tests, use e.g.:

```
$ make test arg=projectroles.tests.test_views
```

For running tests with SODAR Taskflow (not currently publicly available), you can use the supplied make command:

```
$ make test_taskflow
```

Remote Site Development

For developing remote site features, you will want to run two or more SODAR Core example sites concurrently: one `SOURCE` site and one or more `TARGET` sites.

For running a single `TARGET` site in addition to the main site, the fastest way to get started is the following:

First, set up a second database called `sodar_core_target` using `utility/setup_database.sh`.

Next, migrate the new database and create a superuser using `make manage_target`. It is recommended to use a different admin user name than on your `SOURCE` site, to help debugging.

```
$ make manage_target arg=migrate
$ make manage_target arg=createsuperuser
```

Launch your site with `make serve_target`. By default, you can access the site at Port `8001` on localhost. The port can be altered by providing the `target_port` parameter, e.g. `make serve_target target_port=8002`. Management commands to the target site can be issued with the `make manage_target` make command.

Due to how cookies are set by Django, you currently may have to relogin when switching to a different site on your browser. As a workaround you can launch one of the sites in a private/incognito window or use different browsers.

If you need to create multiple target sites for testing `PEER` synchronization features, make sure that you have a separate SODAR Core database for each site and launch each site on a different port on localhost. The configuration `local_target2.py` is included for developing with multiple `TARGET` sites.

5.19 Major Changes

This document details highlighted updates and breaking changes in 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 current and previous releases, see the *full changelog*.

5.19.1 v0.9.0 (2021-02-03)

Release Highlights

- Last major update based on Django v1.11
- Enable modifying local app settings in project update form on target sites
- Add projectroles app settings
- Add remote sync for global projectroles app settings
- Add IP address based access restriction for projects
- Add SSO support via SAML
- Add support for local user invites and local user account creation
- Add batch invites and role updates via management command
- Add REST API views for project invite management
- Add advanced search with multiple terms
- Add REST API view for current user info retrieval

Breaking Changes

Development Helper Scripts

Development helper scripts (`.sh`) have been replaced by a `Makefile`. Get an overview of the available commands via `make usage`.

System Prerequisites

Third party Python package requirements have been upgraded. See the `requirements` directory for up-to-date package versions.

The following third party JS/CSS requirements have been updated:

- JQuery v3.5.1
- Bootstrap v4.5.3

Note: This is the last major update of SODAR Core based on and supporting Django v1.11, which is now out of long term support. From v0.10 onwards, SODAR Core based sites must be implemented on Django v3.x+.

ProjectAppPlugin Search Updates

The expected signature for `ProjectAppPluginPoint.search()` has changed. Instead of the `search_term` string argument, `search_terms` is expected. This argument is a list of strings expected to be combined with `OR` operators.

See the `filesfolders` app for an example of the new implementation.

In SODAR Core v0.9, the old deprecated implementation still works, but searching for multiple terms in the "Advanced Search" view will only return results for the first search term given. This deprecation protection will be removed in the next major version. Please update the `search()` methods in your project app plugins if you have implemented them.

Project Full Title Field

The full title of a project, including the entire category path, can now be accessed via the `Project.full_title`. This enables you to use the field directly in your Django queries and ordering. The value of the field is auto-populated on `Project.save()` and in a database migration accompanied in this release.

As a result, the `Project.get_full_title()` has been deprecated and will be removed in the next major SODAR Core release. Please refactor your usage of that helper into referring to `Project.full_title` directly.

5.19.2 v0.8.4 (2020-11-12)

Release Highlights

This release updates documentation for JOSS submission.

Breaking Changes

N/A

5.19.3 v0.8.3 (2020-09-28)

Release Highlights

- Fix issues in remote project synchronization
- Fix crashes in `siteinfo` app from exceptions raised by plugins

Breaking Changes

Remote Project Sync and Local Categories

When working on a `TARGET` site, creating local projects under categories synchronized from a `SOURCE` site is no longer allowed. This is done to avoid synchronization clashes. If you want to enable local projects on your site in addition to remote ones, you will need to create a local root category for them.

API Changes

`ProjectCreateAPIView` now returns status `403` if called on a target site with disabled local projects, instead of `400` as before.

5.19.4 v0.8.2 (2020-07-22)

Release Highlights

- Enable site-wide background jobs
- Critical bug fixes for project member management
- Minor fixes and updates

Breaking Changes

N/A

5.19.5 v0.8.1 (2020-04-24)

Release Highlights

- Fix checking for remote project status in projectroles REST API views
- Miscellaneous bug fixes

Breaking Changes

SODARAPIObjectInProjectPermissions Removed

The deprecated `SODARAPIObjectInProjectPermissions` base class has been removed from `projectroles.views_api`. Please base your REST API views to one of the remaining base classes instead.

5.19.6 v0.8.0 (2020-04-08)

Release Highlights

- Add API views for the `projectroles` and `filesfolders` apps
- Add new base view classes and mixins for API/Ajax views
- Import the `tokens` API token management app from VarFish
- Allow assigning roles other than owner for categories
- Allow category delegates and owners to create sub-categories and projects
- Allow moving categories and projects under different categories
- Inherit owner permissions from parent categories
- Allow displaying project apps in categories with `category_enable`

- Reorganization of views in apps

Breaking Changes

Owner Permissions Inherited from Categories

Starting in this version of SODAR Core, category owner permissions are automatically inherited by projects below those categories, as well as possible subcategories. If this does not fit your use case, it is recommend to reorganize your project structure and/or give category access to admin users who have access to all projects anyway.

Projectroles Views Reorganized

Views, base views related mixins for the `projectroles` app have been reorganized in this version. Please review your projectroles imports.

The revised structure is as follows:

- UI views and related mixins **remain** in `projectroles.views`
- Ajax API view classes were **moved** into `projectroles.views_ajax`
- REST API view classes **moved** into `projectroles.views_api`
- Taskflow API view classes **moved** into `projectroles.views_taskflow`

The same applies to classes and mxins in view tests. See `projectroles.tests.test_views*` to update imports in your tests.

Renamed Projectroles View Classes

In addition to reorganizing classes into different views, certain view classes intended to be usable by other apps have been renamed. They are listed below.

- `UserAutocompleteAPIView` -> `UserAutocompleteAjaxView`
- `UserAutocompleteRedirectAPIView` -> `UserAutocompleteRedirectAjaxView`

API View Class Changes

`SODARAPIBaseView` and `APIPermissionMixin` have been removed. Please use appropriate classes and mixins found in `projectroles.views_api` and `projectroles.views_ajax` instead.

Base Test Class and Mixin Changes

Base test classes and helper mixins in `projectroles` have been changed as detailed below.

- `SODARAPIViewMixin` has been moved into `projectroles.test_views_api` and renamed into `SODARAPIViewTestMixin`.
- `KnoxAuthMixin` has been combined into `SODARAPIViewTestMixin`.
- `get_accept_header()` returns the header as dict instead of a string.
- `assert_render200_ok()` and `assert_redirect()` have been removed from `TestPermissionBase`. Please use `assert_response()` instead.

In addition to the aforementioned changes, certain minor setup details such as default user rights and may have changed. If you experience unexpected failures in your tests, please review the SODAR Core base test classes and helper methods, refactoring your tests where required.

User Group Updating

The `set_user_group()` helper has been moved from `projectroles.utils` into the `SODARUser` model. It is called automatically on `SODARUser.save()`, so manual calling of the method is not required for most cases.

System Prerequisites

The following third party JS/CSS requirements have been updated:

- JQuery v3.4.1
- Bootstrap v4.4.1
- Popper.js v1.16.0

The minimum supported versions have been upgraded for a number of Python packages in this release. It is highly recommended to also upgrade these for your SODAR Core based site. See the `requirements` directory for up-to date dependencies.

The minimum version requirement for Django has been bumped to 1.11.29.

Default Templates Modified

The default template `base_site.html` has been modified in this version. If you override it with your own altered version, please review the difference and update your templates as appropriate.

SODAR Taskflow v0.4.0 Required

If using SODAR Taskflow, this release requires release v0.4.0 or higher due to required support for the `role_update_irods_batch` flow.

Known Issues

- Category roles beyond owner are not synchronized to target sites in remote project sync. This was omitted to maintain compatibility in existing APIs in this release. The feature is intended to be implemented in SODAR Core v0.9.
- Project/user app settings cannot be set or updated in the project REST API. A separate API for this will be developed. Currently the only way to modify app settings is via the GUI.

5.19.7 v0.7.2 (2020-01-31)

Release Highlights

- Enforce API versions in remote project sync
- Separate base API views for SODAR Core API and external SODAR site APIs
- Redesign user autocomplete field
- Set issuing user email to `reply-to` header for role and invite emails
- Display hidden project app settings to superusers in project update form
- Allow providing custom keyword arguments for backend plugin `get_api()` through `get_backend_api()`
- Enable sorting custom project list columns in plugin definition
- Bug fixes for project list columns

Breaking Changes

User Autocomplete Field Redesigned

User autocomplete field for forms with its related widget(s) have been redesigned with breaking API changes. Please review the *Project App Development* documentation and modify your implementation accordingly.

Remote Project Sync API Version Enforcing

The remote project sync view initiated from a `TARGET` site now sends the version number, making the `SOURCE` site enforce allowed API versions in its request. Hence, when a major breaking change is made on the source site and version requirements updated, requests from the target site will no longer work without upgrading to the latest SODAR Core version.

Exceptions Raised by get_backend_api()

The `get_backend_api()` method for retrieving backend plugin API objects no longer suppresses potential exceptions raised by API object initialization. If it is possible for your API object to raise an exception on initialization, you will need to handle it when calling this method.

System Prerequisites

The minimum version requirement for Django has been bumped to 1.11.27.

KnoxAuthMixin in Tests

Default API configuration for methods in `KnoxAuthMixin` are now set to internal SODAR Core API values. If you use the mixin in the tests of your site, please update the arguments in your method calls accordingly. You can also now supply the *media_type* argument for relevant functions. The `get_accept_header()` method has been moved to a separate `SODARAPIViewMixin` helper mixin.

5.19.8 v0.7.1 (2019-12-18)

Release Highlights

- Project list layout and extra column handling improved
- Allow customizing widgets in app settings
- Enable managing global JS/CSS includes in Django settings
- Initial support for deploying site in kiosk mode
- Critical bug fixes for category and project owner management

Breaking Changes

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.

User Added to get_project_list_value()

The signature of the `get_project_list_value()` method implemented by project app plugins to return data for extra project list columns has changed. The `user` argument which provides the current user has been added. If using this feature, please make sure to update your implementation(s) of the method.

See *Projectroles Django API Documentation* to review the API changes.

5.19.9 v0.7.0 (2019-10-09)

Release Highlights

- Sync peer project information for remote target sites
- Enable revoking access to remote projects
- Allow defining app settings in site apps
- "User in project" scope added into app settings
- Support JSON in app settings
- Project owner management moved to project member views

Breaking Changes

System Prerequisites

The minimum supported versions have been upgraded for a number of Python packages in this release. It is highly recommended to also upgrade these for your SODAR Core based site. See the `requirements` directory for up-to date dependencies.

Backend Javascript Include

The code in `base.html` which was including javascript from backend apps to all templates in projectsroles was removed. Instead, Javascript and CSS associated to a backend plugin should now be included in app templates as needed. This is done using the newly introduced `get_backend_include()` template tag in `projectroles_common_tags`.

Deprecated get_setting() Tag Removed

The deprecated `get_setting()` template tag has been removed from `projectroles_common_tags`. Please use `get_django_setting()` in your templates instead.

ProjectSettingMixin Removed

In `projectroles.tests.test_views`, the deprecated `ProjectSettingMixin` was removed. If you need to populate app settings in your tests, use the `AppSettingAPI` instead.

AppSettingAPI get_setting_defs() Signature Changed

The `get_settings_defs()` function in the app settings API now accepts either a project app plugin or simply the name of the plugin as string. Due to this change, the signature of the API function including argument order has changed. Please see the *API documentation* for more details and update your function calls accordingly.

Default Footer Styling Changed

The styling of the page footer and the default `_footer.html` have changed. You no longer need an extra `<div>` element for the footer content, unless you need to do styling overrides yourself.

5.19.10 v0.6.2 (2019-06-21)

Release Highlights

- Allow hiding app settings from UI forms
- Add template tag for retrieving app settings

Breaking Changes

System Prerequisites

The minimum version requirement for Django has been bumped to 1.11.21.

Template Tag for Django Settings Access Renamed

The `get_setting()` template tag in `projectroles_common_tags` has been renamed into `get_django_setting()`. In this version the old tag still works, but this deprecation protection will be removed in the next release. Please update any references to this tag in your templates.

5.19.11 v0.6.1 (2019-06-05)

Release Highlights

- Add custom project list columns definable in ProjectAppPlugin
- Add example project list column implementation in the filesfolders app

Breaking Changes

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.

5.19.12 v0.6.0 (2019-05-10)

Release Highlights

- Add user specific settings
- Refactor project settings into project/user specific app settings
- Add siteinfo app

Breaking Changes

App Settings (Formerly Project Settings)

The former Project Settings module has been completely overhauled in this version and requries 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',
    },
}
```

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.

5.19.13 v0.5.1 (2019-04-16)

Release Highlights

- Sodarcache refactoring and improvements for API, models, management and app config
- New default error templates

Breaking Changes

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.

5.19.14 v0.5.0 (2019-04-03)

Release Highlights

- New sodarcache app for caching and aggregating data from external services
- Local user mode for site UI and remote sync
- Improved display and logging of remote project sync
- Upgrade to Bootstrap 4.3.1

Breaking Changes

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.

5.19.15 v0.4.5 (2019-03-06)

Release Highlights

- Add user autocomplete in forms
- Allow multiple delegates per project

Breaking Changes

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 undesireable 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.

5.19.16 v0.4.4 (2019-02-19)

Release Highlights

N/A (maintenance/bugfix release)

Breaking Changes

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
```

5.19.17 v0.4.3 (2019-01-31)

Release Highlights

- Add display name configuration for projects and categories
- Hide immutable fields in projectroles forms

Breaking Changes

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`.

5.19.18 v0.4.2 (2019-01-25)

Release Highlights

N/A (maintenance/bugfix release)

Breaking Changes

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.

5.19.19 v0.4.1 (2019-01-11)

Release Highlights

- Configuration updates for API and Projectroles
- Travis-CI setup

Breaking Changes

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' %}
    {# ... #}
  </div>
{% endblock content %}
```

5.19.20 v0.4.0 (2018-12-19)

Release Highlights

- Add filesfolders app from SODAR v0.4.0
- Add bgjobs app from Varfish-Web
- Secure SODAR Taskflow API views
- Separate test server configuration for SODAR Taskflow
- Extra data variable rendering for timeline
- Additional site settings

Breaking Changes

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!

5.19.21 v0.3.0 (2018-10-26)

Release Highlights

- Add remote project metadata and member synchronization between multiple SODAR sites
- Add adminalerts app
- Add taskflowbackend app

Breaking Changes

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.

5.20 SODAR Core Changelog

Changelog for the **SODAR Core** Django app package. Loosely follows the Keep a Changelog guidelines.

Note that the issue IDs here refer to ones in the private CUBI GitLab.

5.20.1 v0.9.0 (2021-02-03)

Added

- **General**
  - SAML SSO authentication support (#588)
  - REST API example `HelloExampleProjectAPIView` in `example_project_app` (#518)
- **Projectroles**
  - Projectroles app settings (#532)
  - Remote sync for projectroles app setting (#533, #586)
  - IP address based access restriction for projects (#531)
  - `is_delegate()` and `is_owner_or_delegate()` helpers for `Project` model
  - Remote sync for non-owner category members (#502)
  - `setting_delete()` function to `AppSettingAPI` (#538)
  - `cleanappsettings` management command (#374)
  - `exclude_inherited` argument in `Project.get_delegates()` (#595)
  - Value options for app settings of type `STRING` and `INTEGER` (#592)
  - Display placeholders for app setting form fields (#584)
  - Support for local user invites (#548, #613, #615, #621)
  - Local user account creation and updating (#547)
  - `batchupdateroles` management command (#15, #602)
  - Project invite REST API views (#15, #598)
  - Advanced search with multiple terms (#609)
  - Search result pagination control (#610)
  - REST API endpoint for retrieving current user info (#626)

Changed

- **General**
  - Replace development helper scripts with `Makefile` (#135)
  - Upgrade to Bootstrap v4.5.3 and jQuery v3.5.1 (#563)
  - Upgrade to Chromedriver v87
  - Upgrade general Python requirements (#576)
  - Migrate GitHub CI from Travis to GitHub actions (#577)
  - Refactor example `PROJECT_USER` scope app settings (#599)
  - Set logging level in test configurations to `CRITICAL` (#604)
- **Filesfolders**
  - Update `search()` and `find()` for multiple search terms (#609)

- **Projectroles**
  - Allow updating local app settings on a `TARGET` site (#545)
  - Refactor project list filtering (#566)
  - Move project list javascript to `project_list.js` (#566)
  - Rename owner role transfer URL pattern and timeline event (#590)
  - Add `sodar_url` override to `modify_assignment()`
  - Rename `ProjectSearchResultsView` and its template (#609)
  - Implement `get_full_title()` as `Project.full_title` field (#93)
  - Clarify invite accepting procedure in invite email (#627)
  - Redirect to home view when reusing accepted invite link (#628)
- **Userprofile**
  - Cosmetic updates for user detail template (#600)

Fixed

- **Projectroles**
  - Invite redirect not working in Add Member view (#589)
  - Wrong role label displayed for category owner/delegate in member list (#593)
  - Django settings access in `forms` and `serializers`
  - Delegate limit check broken by existing delegate roles of inherited owners (#595)
  - Crash in project invite if multiple users exist with the same email (#614)
  - Project delegate able to revoke invite for another delegate (#617)
  - Column alignment in invite list (#606)
  - `get_not_found_alert()` fails if called with app plugin search type (#624)
- **Taskflowbackend**
  - Django settings access in `api` (#605)
  - `sodar_url` override not working if `request` object is present (#605)

Removed

- **General**
  - Travis CI setup in `.travis.yml` (#577)
- **Projectroles**
  - Template `_project_filter_item.html` (#566)
  - Template tag `get_project_list()` (#566)
  - Deprecate old implementation of `ProjectAppPluginPoint.search()` (#609, #618)
  - Deprecate `Project.get_full_title()` (#93)

5.20.2 v0.8.4 (2020-11-12)

Changed

- **General**
  - Documentation updates for JOSS submission

5.20.3 v0.8.3 (2020-09-28)

Added

- **General**
  - Missing migration for the `SODARUser` model (#581)

Changed

- **General**
  - Upgrade to Chromedriver v85 (#569)
- **Projectroles**
  - Improve project list header legend (#571)
  - Make `sync_source_data()` atomic
  - Prevent creation of local projects under remote categories (#583)
- **Siteinfo**
  - Refactor app plugin statistics retrieval (#573)

Fixed

- **General**
  - Invalid statement in `setup_database.sh` (#580)
- **Projectroles**
  - Missing exception handling for `sync_source_data()` calls (#582)
  - Crash from conflicting local category structure (#582)
- **Siteinfo**
  - Crash from exceptions raised by app plugin `get_statistics()` (#572)
- **Timeline**
  - CSS for `sodar-tl-link-detail` links (#578)

Removed

- **General**
  - Unused `Pillow` dependency (#575)

5.20.4 v0.8.2 (2020-07-22)

Added

- **Bgjobs**
  - Enable site-wide background jobs (#544)
  - Site app plugin for site-wide background jobs (#544)
- **Projectroles**
  - `sodar-header-button` CSS class (#550)
  - Logging for `AppSettingAPI` (#559)

Changed

- **Projectroles**
  - Upgrade to Chromedriver v83 (#543)
  - Rename `is_app_link_visible()` template tag into `is_app_visible()` (#546)
  - Refactor project list to reduce queries and template tag use (#551, #567)

Fixed

- **Projectroles**
  - Transferring project ownership to inherited owner not allowed (#534)
  - Uniqueness constraint in `AppSetting` incompatible with `PROJECT_USER` scope settings (#542)
  - Inherited owner email address not displayed in project member list (#541)
  - App visibility check broken in `project_detail.html` (#546)
  - Invite accept for a category invoking Taskflow and causing a crash (#552)
  - Project form `parent` forced to wrong value if user lacks role in parent category (#558)
  - Invalid `app_name` not handled in `AppSettingAPI.get_default_setting()` (#560)
  - Empty JSON and false boolean app settings not set in project form (#557)
  - Minor Javascript errors thrown by `projectroles.js` (#536)
  - Long lines breaking email preview layout (#564)

5.20.5 v0.8.1 (2020-04-24)

Added

- **Projectroles**
  - CSS class `sodar-pr-project-list-custom` for custom project list items (#525)

Fixed

- **Projectroles**
  - CSS padding issue with `sodar-list-btn` and Chrome (#529, sodar#844)
  - Crash from missing optional setting `PROJECTROLES_DISABLE_CATEGORIES` (#524)
  - Remote project editing not prevented in REST API views (#523)

Removed

- **Projectroles**
  - Deprecated `SODARAPIObjectInProjectPermissions` base class (#527)

5.20.6 v0.8.0 (2020-04-08)

Added

- **General**
  - "For the Impatient" section in docs
- **Filesfolders**
  - API views for file, folder and hyperlink management (#443)
- **Projectroles**
  - Import new REST API view base classes from SODAR (#48, #461)
  - Import base serializers from SODAR (#462)
  - API views for project and role management (#48, #450)
  - `projectroles.tests.test_views_api.TestAPIViewsBase` for API view testing (#48)
  - `SODARAPIPermissionTestMixin` for API view permission tests
  - New helper methods in `SODARAPIViewTestMixin`
  - Provide live server URL for Taskflow in `TestTaskflowBase.request_data` (#479)
  - `TestTaskflowAPIBase` for testing API views with SODAR Taskflow (#488)
  - Permission tests using Knox tokens (#476)
  - Base Ajax view classes in `projectroles.views_ajax` (#465)
  - Allow assigning roles for categories (#463)
  - Allow displaying project apps in categories with `category_enable` (#447)

  - Allow category delegates and owners to create sub-categories and projects (#464)
  - `get_role_display_name()` helper in `projectroles_common_tags` (#505)
  - `get_owners()`, `is_owner()` and `get_all_roles()` helpers for `Project` (#464)
  - Allow using legacy UI test login method with `PROJECTROLES_TEST_UI_LEGACY_LOGIN` (#509)
  - Allow moving categories and projects under different categories (#512)
  - `SODARForm` and `SODARModelForm` base classes for forms
  - Enable retrieving flat recursive list of children objects in `Project.get_children()`
  - Support for `data` in permission test `assert_response()` method (#155)
- **Taskflowbackend**
  - `get_inherited_roles()` helper (#464)
- **Timeline**
  - `get_models()` helper
- **Tokens**
  - Add app from varfish-web (#452)

Changed

- **General**
  - Upgrade minimum Django version to v1.11.29 (#520)
  - Upgrade JQuery to v3.4.1 (#519)
  - Upgrade Bootstrap to v4.4.1 (#460)
  - General upgrade for Python package requirements (#124, #459)
  - Reorganize view classes and URL patterns (#480)
  - Refactor Ajax views (#465, #475)
  - Update `CONTRIBUTING.rst`
  - Use `SODARForm` and `SODARModelForm` base classes in forms
- **Projectroles**
  - Suppress peer site removal logging if nothing was removed (#478)
  - Refactor `SODARCoreAPIBaseView` into `SODARCoreAPIBaseMixin` (#461)
  - Allow providing single user to `assert_response()` in permission tests (#474)
  - Move `SODARAPIViewTestMixin` into `test_views_api` and rename (#471)
  - Move `KnoxAuthMixin` functionality into `SODARAPIViewTestMixin`
  - `get_accept_header()` in API tests returns header as dict
  - Refactor base permission test classes (#490)
  - Move `utils.set_user_group()` to `SODARUser.set_group()` (#483)
  - Call `set_group()` in `SODARUser.save()` (#483)

  - Replace `projectroles_tags.is_app_hidden()` with `is_app_link_visible()`
  - Inherit owner permissions from parent categories (#464)
  - Refactor project roles template (#505)
  - Disable owner updating in project update form (#508)
  - Allow updating project parent via SODAR Taskflow (#512)
- **Taskflowbackend**
  - Refactor `synctaskflow` management command and add logging
- **Timeline**
  - Display app for categories (#447)

Fixed

- **General**
  - Duplicate `contributing.rst` redirection file in docs (#481)
  - `.tox` not ignored in `black.sh`
  - Coverage checks in Travis-CI (#507)
- **Projectroles**
  - Swapping owner and delegate roles not allowed if at delegate limit (#477)
  - Remote sync for owner role failing with specific user order in data (#439)
  - Redundant updating of `Project.submit_status` during project creation
  - Make `test_widget_user_options()` more reliable (#253)
  - Missing permission check by role type in `RoleAssignmentDeleteView.post()` (#492)
  - Unordered queryset warnings from the `User` model (#494)
  - Incorrect user iteration in `test_user_autocomplete_ajax()` (#469)
  - Redundant input validation preventing search with valid characters (#472)
  - Local users disabled in local development configuration (#500)
  - Member link not visible in responsive project dropdown (#466)
  - CSS issues with Bootstrap 4.4.1 in search pagination (#372, #460)
  - Raise `ImproperlyConfigured` for missing parameters in `ProjectAccessMixin` (#516)
- **Timeline**
  - CSS issues with Bootstrap 4.4.1 (#460)

Removed

- **Projectroles**
  - `SODARAPIBaseView` base class, replaced by API view mixins (#461)
  - `KnoxAuthMixin` from view tests
  - `get_selectable_users()` from `forms`
  - Redundant render/redirect helpers from `TestPermissionBase`: use `assert_response()` instead (#484)
  - `APIPermissionMixin` for API views: use base API/Ajax view classes instead (#467)
  - `is_app_hidden()` from `projectroles_tags`

5.20.7 v0.7.2 (2020-01-31)

Added

- **Projectroles**
  - `custom_order` argument in `get_active_plugins()` (#431)
  - Enable ordering custom project list columns in project app plugin (#427)
  - `SODARCoreAPIBaseView` base API view class for internal SODAR Core apps (#442)
  - API version enforcing in `RemoteProjectsSyncView` and `syncremote.py` (#444)
  - Allow extra keyword arguments in `get_backend_api()` (#397)
  - Example usage of `get_backend_api()` extra kwargs in `example_backend_app` (#397)
  - `SODARUserChoiceField` and `get_user_widget()` for user selection in forms (#455)
  - Setting `reply-to` headers for role change and invite emails (#446)
  - No reply note and related `PROJECTROLES_EMAIL_SENDER_REPLY` setting (#446)
  - Display hidden project app settings to superusers (#424)
- **Sodarcache**
  - Allow limiting `deletecache` to a specific project (#448)

Changed

- **General**
  - Upgrade minimum Django version to 1.11.27
  - Base `RemoteProjectGetAPIView` on `SODARCoreAPIBaseView` (#442)
  - Upgrade to Chromedriver v80 (#510)
- **Bgjobs**
  - Make `specialize_job()` more robust (#456)
- **Projectroles**
  - Accept null value for `AppSetting.value_json` (#426)

  - Use `PluginContextMixin` in `ProjectContextMixin` (#430)
  - Move `get_accept_header()` to `SODARAPIViewMixin` (#445)
  - Allow exceptions to be raised by `get_backend_plugin()` (#451)
  - Improve tour help CSS (#438)
  - Field order in `RoleAssignmentOwnerTransferView` (#441)
  - Redesign user autocomplete handling in forms (#455)
  - Rename `SODARUserAutocompleteWidget` and `SODARUserRedirectWidget` (#455)
  - Disable ownership transfer link if owner is the only project user (#454)

Fixed

- **Projectroles**
  - Potential crash in `_project_header.html` with ownerless kiosk mode category (#422)
  - Form crash when saving a JSON app setting with `user_modifiable=False` (#426)
  - Inconsistent plugin ordering in custom project list columns (#428)
  - Project app plugins included twice in `HomeView` (#432)
  - `ProjectPermissionMixin` query set override with `get_project_filter_key()`
  - Search disabled with unchanged input value on search page load (#436)
  - Subprojects queried for non-categories in `project_detail.html` (#434)
  - Current owner selectable in ownership transfer form (#440)
- **Taskflowbackend**
  - Potential crash in `TaskflowAPI` initialization

Removed

- **Projectroles**
  - Unused backend plugins queried for context data in `HomeView` (#433)
  - Unneeded `UserAutocompleteExcludeMembersAPIView` (#455)

5.20.8 v0.7.1 (2019-12-18)

Added

- **General**
  - Include CHANGELOG in documentation (#379)
- **Projectroles**
  - `widget_attrs` parameter for project and user settings (#404)
  - Remote project member management link for target projects (#382)
  - Current user in `get_project_list_value()` arguments (#413)

  - Display category owner in page header (#414)
  - Configuring UI test settings via Django settings or `TestUIBase` vars (#417)
  - Initial support for deploying site in kiosk mode (#406)
  - Optional disabling of default CDN Javascript and CSS includes (#418)
  - Defining custom global JS/CSS includes in Django settings (#418)

Changed

- **General**
  - Change "Breaking Changes" doc into "Major Changes" (#201)
  - Refactor and rename ownership transfer classes and template
  - Use RTD theme in documentation (#384)
  - Upgrade to Chromedriver v79
- **Adminalerts**
  - Rename `INACTIVE` alert state in UI (#396)
  - Rename URL name and pattern for activation API view (#378)
  - Improve alert detail page layout (#385)
- **Projectroles**
  - Improve unsupported browser warning (#405)
  - Move project list description into tooltip (#388)
- **Siteinfo**
  - Improve page title and heading (#402)
- **Sodarcache**
  - Clarify management command logging (#403)
- **Timeline**
  - Improve extra data status tab legend (#380)

Fixed

- **General**
  - PPA used for Python 3.6 installs no longer available (#416)
- **Filesfolders**
  - Invalid HTML in project list extra columns
- **Projectroles**
  - Dismissing login error alert in `login.html` not working (#377)
  - Current owner queries incorrectly filtered in `RoleAssignmentOwnerTransferView` (#393)
  - Hardcoded project type display name in sent emails (#398)
  - Silent failing of invalid app setting type in plugin definition (#390)

- Exception raised by hidden sidebar in sidebar height calculation (#407)
- Crash in `get_default_setting()` if default JSON value was not set (#389)
- Owner widget hidden in category update view (#394)
- Project list extra column header alignment not set (#412)
- `get_project_list_value()` template tag displaying "None" on null value (#411)

5.20.9 v0.7.0 (2019-10-09)

Added

- **General**
  - Development env file example `env.example` (#297)
  - Postgres database development setup script (#302)
  - `ENABLE_DEBUG_TOOLBAR` setting for local development (#349)
  - `local_target2.py` config for peer remote site development (#200)
- **Adminalerts**
  - Activate/suspend button in alert list (#42)
- **Bgjobs**
  - Pagination for background job list (#335)
  - `BGJOBS_PAGINATION` Django setting (#335)
- **Projectroles**
  - `get_backend_include()` common template tag (#261)
  - `css_url` member variable in `BackendPluginPoint` (#261)
  - Example of on-demand Javascript/CSS inclusion in example apps (#261)
  - Remote project link display toggle for target sites (#276)
  - Project UUID clipboard copying button (#290)
  - Support for app settings in site apps (#308)
  - Initial implemenetation for common clipboard copying visualization (#333)
  - Send email for owner role assignment (#325)
  - Common pagination include template `_pagination.html` (#334)
  - Synchronization and display of `PEER` sites in remote site management (#200)
  - Link for copying remote site secret token in remote site list (#332)
  - Project ownership transfer from member list (#287)
  - UI notification for disabled member management on target sites (#301)
  - Management command `addremotesite` for adding remote sites (#314)
  - JSON support for app settings (#268)
  - `get_setting_def()` in app settings API

  - Timeline logging of app settings in project creation (#359)
  - "Project and user" scope for app settings (#266)
  - `REVOKED` status for remote projects with revoked access (#327)
  - `Project.is_revoked()` helper (#327)
  - Disabling access for non-owner/delegate for revoked projects in `ProjectPermissionMixin` (#350)
- **Timeline**
  - Display event extra data as JSON (#6)
- **Userprofile**
  - User setting for project UUID clipboard copying (#290, #308)

Changed

- **General**
  - Upgrade Chromedriver to version 77.0.3865.40
  - Use `CurrentUserFormMixin` instead of repeated code (#12)
  - Run tests in parallel where applicable
  - Upgrade minimum Django version to 1.11.25 (#346)
  - General upgrade for Python package requirements (#282)
- **Adminalerts**
  - Use common pagination template
- **Projectroles**
  - Improve user name placeholder in `login.html` (#294)
  - Backend app Javascript and CSS included on-demand instead of for all templates (#261)
  - Make sidebar hiding dynamic by content height (#316)
  - Replace `login_and_redirect()` in UI tests with a faster cookie based function (#323)
  - Refactor remote project display on details page (#196)
  - Refactor AppSettingAPI (#268)
  - Enable calling `AppSettingAPI.get_setting_defs()` with app name instead of plugin object
  - Use `ProjectPermissionMixin` on project detail page (#350)
- **Timeline**
  - Use common pagination template (#336)

Fixed

- **Projectroles**
  - Output of template tag `get_project_link()`
  - Redundant inheritance in `CurrentUserFormMixin` (#12)
  - Trailing slashes not parsed correctly in remote project URLs (#319)
  - Crash in `get_project_column_count()` with no active project app plugins (#320)
  - UI test helper `build_selenium_url()` refactored to work with Chrome v77 (#337)
  - Disallow empty values in `RemoteSite.name`
  - Remote sync of parent category roles could fail with multiple subprojects
  - `RemoteProject` modifications not saved during sync update
  - Timeline events not created in remote project sync (#370)
  - DAL select modifying HTML body width (#365)
  - Footer overflow breaking layout (#367, #375)
- **Timeline**
  - Crash from exception raised by `get_object_link()` in a plugin (#328)

Removed

- **Projectroles**
  - Duplicate database indexes from `RoleAssignment` (#285)
  - Deprecated `get_setting()` tag from `projectroles_common_tags` (#283)
  - Project owner change from project updating form (#287)
  - `ProjectSettingMixin` from `projectoles.tests.test_views` (#357)

5.20.10 v0.6.2 (2019-06-21)

Added

- **General**
  - Badges for Readthedocs documentation and Zenodo DOI (#274)
- **Bgjobs**
  - `BackgroundJobFactory` for tests from Varfish-web
- **Projectroles**
  - Unit test to assure owner user creation during project update when using SODAR Taskflow (sodar_taskflow#49)
  - Common template tag `get_app_setting()` (#281)
  - Hiding app settings from forms with `user_modifiable` (#267)
  - `AppSetting.value_json` field (#268)

- **Sodarcache**
  - Logging in `delete_cache()` (#279)
- **Userprofile**
  - Support for `AppSetting.user_modifiable` (#267)

Changed

- **General**
  - Upgrade minimum Django version to 1.11.21 (#278)
- **Projectroles**
  - `get_setting()` template tag renamed into `get_django_setting()` (#281)
  - Implement project app descriptions on details page with `get_info_link()` (#277)

Fixed

- **General**
  - Documentation sections for Readthedocs

5.20.11 v0.6.1 (2019-06-05)

Added

- **Filesfolders**
  - Example project list columns (#265)
  - Setting `FILESFOLDERS_SHOW_LIST_COLUMNS` to manage example project list columns (#265)
- **Projectroles**
  - Optional project list columns for project apps (#265)
- **Sodarcache**
  - `delete_cache()` API function (#257)

Changed

- **Projectroles**
  - Refactor `RemoteProject.get_project()` (#262)
  - Use `get_info_link()` in remote site list (#264)
  - Define `SYSTEM_USER_GROUP` in `SODAR_CONSTANTS` (#251)
  - Make pagedown textarea element resizeable and increase minimum height (#273)
- **Sodarcache**
  - Handle and log raised exceptions in `synccache` management command (#272)
- **Userprofile**

  - Disable user settings link if no settings are available (#260)

Fixed

- **General**
  - Chrome and Chromedriver version mismatch in Travis-CI config (#254)
- **Projectroles**
  - Remove redundant `get_project_list()` call from `project_detail.html`

Removed

- **Projectroles**
  - Unused project statistics in the home view (#269)
  - App settings deprecation protection (#245)
- **Sodarcache**
  - Unused `TaskflowCacheUpdateAPIView` (#205)

5.20.12 v0.6.0 (2019-05-10)

Added

- **Filesfolders**
  - Provide app statistics for siteinfo (#18)
- **Projectroles**
  - User settings for settings linked to users instead of projects (#16)
  - `user_settings` field in project plugins (#16)
  - Optional `label` key for settings
  - Optional "wait for element" args in UI test helpers to ease Javascript testing (#230)
  - `get_info_link()` template tag (#239)
  - `get_setting_defs()` API function for retrieving project and user setting definitions (#225)
  - `get_all_defaults()` API function for retrieving all default setting values (#225)
  - Human readable labels for app settings (#9)
- **Siteinfo**
  - Add app for site info and statistics (#18)
- **Sodarcache**
  - Optional `--project` argument for the `synccache` command (#232)
- **Timeline**
  - Provide app statistics for siteinfo (#18)
- **Userprofiles**

  - View and form for displaying and updating user settings (#16)

Changed

- **General**
  - Upgrade to ChromeDriver v74 (#221)
- **Bgjobs**
  - Job order to match downstream Varfish
- **Filesfolders**
  - Update app settings (#246)
- **Projectroles**
  - Rename `project_settings` module to `app_settings` (#225)
  - App settings API updated to support project and user settings (#225)
  - Write an empty dict for `app_settings` by default

Fixed

- **Bgjobs**
  - Date formatting in templates (#220)
- **Sodarcache**
  - Crash from `__repr__()` if project not set (#223)
  - Broken backend plugin icon (#250)

Removed

- **Timeline**
  - Unused and deprecated project settings (#246)

5.20.13 v0.5.1 (2019-04-16)

Added

- **General**
  - Bgjobs/Celery updates from Kiosc (#175)
  - Default error templates in `projectroles/error/*.html` (#210)
- **Projectroles**
  - Optional `user` argument in `ProjectAppPlugin.update_cache()` (#203)
  - Migration for missing `RemoteProject` foreign keys (#197)
- **Sodarcache**
  - API logging (#207)

  - Indexing of identifying fields (#218)

Changed

- **General**
  - Extend `projectroles/base.html` for all site app templates, update docs (#217)
  - Use projectroles error templates on the example site (#210)
- **Sodarcache**
  - Make `user` field optional in models and API (#204)
  - Rename app configuration into `SodarcacheConfig` to follow naming conventions (#202)
  - Rename `updatecache` management command to `synccache` (#208)

Fixed

- **General**
  - Add missing curl dependency in `install_os_dependencies.sh` (#211)
  - Django debug toolbar not displayed when using local configuration (#213)
- **Projectroles**
  - Nested app names not properly returned by `utils.get_app_names()` (#206)
  - Forced width set for all Bootstrap modals in `projectroles.css` (#209)
  - Long category paths breaking remote project list (#84)
  - Incorrect table rows displayed during project list initialization (#212)
  - Field `project` not set for source site `RemoteProject` objects (#197)
  - Crash from `project_base.html` in site app if not overriding title block (#216)

Removed

- **General**
  - Django debug toolbar workarounds from `project.css` and `project.scss` (#215)
- **Projectroles**
  - `PROJECTROLES_ADMIN_OWNER` deprecation protection: use `PROJECTROLES_DEFAULT_ADMIN` (#190)

5.20.14 v0.5.0 (2019-04-03)

Added

- **Projectroles**
  - Warning when using an unsupported browser (#176)
  - Setting `PROJECTROLES_BROWSER_WARNING` for unsupported browser warning (#176)
  - Javascript-safe toggle for `get_setting()` template tag
  - ID attributes in site containers (#173)
  - Setting `PROJECTROLES_ALLOW_LOCAL_USERS` for showing and syncing non-LDAP users (#193)
  - Allow synchronizing existing local target users for remote projects (#192)
  - Allow selecting local users if in local user mode (#192)
  - `RemoteSite.get_url()` helper
  - Simple display of links to project on external sites in details page (#182)
- **Sodarcache**
  - Create app (#169)

Changed

- **General**
  - Upgrade to Bootstrap 4.3.1 and Popper 1.14.7 (#181)
- **Projectroles**
  - Improve remote project sync logging (#184, #185)
  - Rename `PROJECTROLES_ADMIN_OWNER` into `PROJECTROLES_DEFAULT_ADMIN` (#187)
  - Update login template and `get_login_info()` to support local user mode (#192)

Fixed

- **Projectroles**
  - Crash in `get_assignment()` if called with AnonymousUser (#174)
  - Line breaks in templates breaking `badge-group` elements (#180)
  - User autocomplete for users with no group (#199)

Removed

- **General**
  - Deprecated Bootstrap 4 workaround from `project.js` (#178)

5.20.15 v0.4.5 (2019-03-06)

Added

- **Projectroles**
  - User autocomplete widgets (#51)
  - Logging in `syncgroups` and `syncremote` management commands
  - `PROJECTROLES_DELEGATE_LIMIT` setting (#21)

Changed

- **General**
  - Upgrade minimum Django version to 1.11.20 (#152)
  - Use user autocomplete in forms in place of standard widget (#51)
- **Filesfolders**
  - Hide parent folder widgets in item creation forms (#159)
- **Projectroles**
  - Enable allowing multiple delegates per project (#21)

Fixed

- **Filesfolders**
  - File upload wiget error not displayed without Bootstrap 4 workarounds (#164)
- **Projectroles**
  - Potential crash in `syncremote` if run as Celery job (#160)

Removed

- **General**
  - Old Bootstrap 4 workarounds for django-crispy-forms (#157)

5.20.16 v0.4.4 (2019-02-19)

Changed

- **Projectroles**
  - Modify `modifyCellOverflow()` to work with non-table containers (#149)
  - Non-Pagedown form textarea height no longer adjusted automatically (#151)

Fixed

- **Projectroles**
  - Crash in remote project sync caused by typo in `remoteproject_sync.html` (#148)
  - Textarea element CSS override breaking layout in third party components (#151)

5.20.17 v0.4.3 (2019-01-31)

Added

- **General**
  - Codacy badge in `README.rst` (#140)
- **Projectroles**
  - Category and project display name configuration via `SODAR_CONSTANTS` (#141)
  - `get_display_name()` utils function and template tag to retrieve `DISPLAY_NAMES` (#141)
  - Django admin link warning if taskflowbackend is enabled

Changed

- **General**
  - Use `get_display_name()` to display category/project type (#141)
- **Projectroles**
  - Hide immutable fields in forms (#142)
  - Rename Django admin link in user dropdown

Fixed

- **Projectroles**
  - View access control for categories (#143)

Removed

- **General**
  - Redundant `rules.is_superuser` predicates from rules (#138)
- **Projectroles**
  - `get_project_type()` template tag (use `get_display_name()` instead)
  - Unused template `_roleassignment_import.html`
  - `PROJECT_TYPE_CHOICES` from `SODAR_CONSTANTS`
  - `force_select_value()` helper no longer used in forms (#142)

5.20.18 v0.4.2 (2019-01-25)

Added

- **General**
  - Flake8 and Codacy coverage in Travis-CI (#122)
  - Flake8 in GitLab-CI (#127)
- **Projectroles**
  - Automatically pass CSRF token to unsafe Ajax HTTP methods (#116)
  - Queryset filtering in `ProjectPermissionMixin` from digestiflow-web (#134)
  - Check for `get_project_filter_key()` from digestiflow-web (#134)

Changed

- **General**
  - Upgrade minimum Django version to 1.11.18 (#120)
  - Upgrade Python dependencies (#123)
  - Update .coveragerc
  - Upgrade to Bootstrap 4.2.1 (#23)
  - Upgrade to JQuery 3.3.1 (#23)
  - General code cleanup
  - Code formatting with Black (#133)
- **Filesfolders**
  - Refactor `BatchEditView` and `FileForm.clean()` (#128)
- **Projectroles**
  - Use `alert-dismissable` to dismiss alerts (#13, #130)
  - Update DataTables dependency in `search.html` template
  - Refactor `ProjectModifyMixin` and `RemoteProjectAPI` (#128)
  - Disable `USE_I18N` in example site settings (#117)

  - Refactor `ProjectAccessMixin._get_project()` into `get_project()` (#134)
  - Rename `BaseAPIView` into `SODARAPIBaseView`
- **Timeline**
  - Refactor `get_event_description()` (#30, #128)

Fixed

- **General**
  - Django docs references (#131)
- **Projectroles**
  - `sodar-list-dropdown` layout broke down with Bootstrap 4.2.1 (#23)
  - `TASKFLOW_TEST_MODE` not checked for allowing SODAR Taskflow tests (#126)
  - Typo in `update_remote` timeline event description (#129)
  - Textarea height modification (#125)
  - Text wrapping in `sodar-list-btn` and `sodar-list-dropdown` with Bootstrap 4.2.1 (#132)
- **Taskflowbackend**
  - `TASKFLOW_TEST_MODE` not checked for allowing `cleanup()` (#126)
  - `FlowSubmitException` raised instead of `CleanupException` in `cleanup()`

Removed

- **General**
  - Legacy Python2 `super()` calls (#118)
- **Projectroles**
  - Custom alert dismissal script (#13)
- **Example Site App**
  - Example file `test.py`

5.20.19 v0.4.1 (2019-01-11)

Added

- **General**
  - Travis-CI configuration (#90)
- **Adminalerts**
  - Option to display alert to unauthenticated users with `require_auth` (#105)
- **Projectroles**
  - `TaskflowAPIAuthentication` for handling Taskflow API auth (#47)
  - Handle `GET` requests for Taskflow API views (#47)

  - API version settings `SODAR_API_ALLOWED_VERSIONS` and `SODAR_API_MEDIA_TYPE` (#111)
  - Site app support in `change_plugin_status()`
  - `get_sodar_constants()` helper (#112)
- **Taskflowbackend**
  - API logging

Changed

- **General**
  - Upgrade minimum Python version requirement to 3.6 (#102)
  - Update and cleanup Gitlab-CI setup (#85)
  - Update Chrome Driver for UI tests
  - Cleanup Chrome setup
  - Enable site message display in login view (#105)
  - Cleanup and refactoring for public GitHub release (#90)
  - Drop support for Ubuntu Jessie and Trusty
  - Update installation utility scripts (#90)
- **Filesfolders**
  - Move inline javascript into `filesfolders.js`
- **Projectroles**
  - Refactor `BaseTaskflowAPIView` (#47)
  - Rename Taskflow specific API views (#104)
  - Unify template tag names in `projectroles_tags`
  - Change default SODAR API media type into `application/vnd.bihealth.sodar-core+json` (#111)
  - Allow importing `SODAR_CONSTANTS` into settings for modification (#112)
  - Move `SODAR_CONSTANTS` to `constants.py` (#112)
- **Timeline**
  - Rename Taskflow specific API views (#104)

Fixed

- **Filesfolders**
  - Overwrite check for zip archive upload if unarchiving was unset (#113)
- **Projectroles**
  - Potential Django crash from auth failure in Taskflow API views
  - Timeline description for updating a remote project

  - Project update with Taskflow failure if description not set (#110)
- **Timeline**
  - `TaskflowEventStatusSetAPIView` skipping `sodar_token` check (#109)

Removed

- **Filesfolders**
  - Unused dropup app buttons mode in templates (#108)
- **Projectroles**
  - Unused arguments in `email` API
  - Unused static file `shepherd-theme-default.css`
  - Disabled role importing functionality (#61, pending #17)
  - Unused dropup app buttons mode in templates (#108)
- **Timeline**
  - `ProjectEventStatus.get_timestamp()` helper

5.20.20 v0.4.0 (2018-12-19)

Added

- **General**
  - `SODAR_API_DEFAULT_HOST` setting for server host for API View URLs (sodar#396)
- **Bgjobs**
  - Add app from varfish-web (#95)
- **Filesfolders**
  - Add app from sodar v0.4.0 (#86)
- **Projectroles**
  - Setting `PROJECTROLES_ENABLE_SEARCH` (#70)
  - Re-enable "home" link in project breadcrumb (#80)
  - `get_extra_data_link()` in ProjectAppPluginPoint for timeline extra data (#6)
  - Allow overriding project class in ProjectAccessMixin
  - Optional disabling of categories and nesting with `PROJECTROLES_DISABLE_CATEGORIES` (#87)
  - Optional hiding of apps from project menus using `PROJECTROLES_HIDE_APP_LINKS` (#92)
  - Secure SODAR Taskflow API views with `TASKFLOW_SODAR_SECRET` (#46)
- **Taskflowbackend**
  - `test_mode` flag configured with `TASKFLOW_TEST_MODE` in settings (#67)
  - Submit `sodar_secret` for securing Taskflow API views (#46)
- **Timeline**

  - Display of extra data using `{extra-NAME}` (see documentation) (#6)

Changed

- **General**
  - Improve list button and dropdown styles (#72)
  - Move pagedown CSS overrrides into `projectroles.css`
  - Reduce default textarea height (#96)
- **Projectroles**
  - Make sidebar resizeable in CSS (#71)
  - Disable search if `PROJECTROLES_ENABLE_SEARCH` is set False (#70)
  - Allow appending custom items in project breadcrumb with `nav_sub_project_extend` block (#78)
  - Allow replacing project breadcrumb with `nav_sub_project` block (#79)
  - Disable remote site access if `PROJECTROLES_DISABLE_CATEGORIES` is set (#87), pending #76
  - Disable access to invite views for remote projects (#89)
  - Set "project guest" as the default role for new members (#94)
  - Make noncritical settings variables optional (#14)

Fixed

- **General**
  - Potential inheritance issues in test classes (#74)
  - LDAP dependency script execution (#75)
- **Projectroles**
  - Long words in app names breaking sidebar (#71)
  - Member modification buttons visible for superuser in remote projects (#73)
  - Breadcrumb project detail link display issue in `base.html` (#77)
  - "None" string displayed for empty project description (#91)
  - Crash in search from empty project description

5.20.21 v0.3.0 (2018-10-26)

Added

- **General**
  - Test config and script for SODAR Taskflow testing
- **Adminalerts**
  - Add app based on SODAR v0.3.3 (#27)

  - `TASKFLOW_TARGETS` setting
- **Projectroles**
  - `RemoteSite` and `RemoteProject` models (#3)
  - `RemoteSiteAppPlugin` site plugin (#3)
  - `PROJECTROLES_SITE_MODE` and `PROJECTROLES_TARGET_CREATE` settings (#3)
  - Remote site and project management site app (#3)
  - Remote project API (#3)
  - Generic SODAR API base classes
  - `SodarUserMixin` for SODAR user helpers in tests
  - Optional `readme` and `sodar_uuid` args for `_make_project()` in tests
  - `syncremote` management command for calling `RemoteProjectAPI.sync_source_data()`
  - `get_project_by_uuid()` and `get_user_by_username()` template tags
  - `get_remote_icon()` template tag (#3)
  - Predicates in rules for handling remote projects (#3)
  - `ProjectModifyPermissionMixin` for access control for remote projects (#3)
  - `is_remote()` and `get_source_site()` helpers in the `Project` model (#3)
  - Include template `_titlebar_nav.html` for additional title bar links
- **Taskflowbackend**
  - Add app based on SODAR v0.3.3 (#38)
- **Timeline**
  - `RemoteSite` model in `api.get_event_description()` (#3)

Changed

- **General**
  - Update documentation for v0.3 changes, projectroles usage and fixes to v0.2 docs (#26)
- **Adminalerts**
  - Make `ADMINALERTS_PAGINATION` setting optional
- **Projectroles**
  - Allow `LoggedInPermissionMixin` to work without a permission object for superusers
  - Enable short/full title selection and remote project icon in `get_project_link()` template tag
  - Refactor rules
  - Disable Taskflow API views if Taskflow backend is not enabled (#37)
  - DataTables CSS and JS includes loaded in the search template (#45)
- **Timeline**
  - Minor refactoring of `api.get_event_description()` (#30)

Fixed

- **General**
  - Pillow dependency typo in `requirements/base.txt` (#33)
  - Login page crash if `AUTH_LDAP*_DOMAIN_PRINTABLE` not found (#43)
- **Projectroles**
  - Sidebar create project visible for site apps if URL name was "create" (#36)
  - Enabling LDAP without a secondary backend caused a crash (#39)

Removed

- **General**
  - iRODS specific CSS classes from `projectroles.css`
  - App content width limit in `projectroles.css`
  - Domain-specific Login JQuery
  - DataTables CSS and JS includes from base template (#45)

5.20.22 v0.2.1 (2018-09-20)

Changed

- **General**
  - Change `omics_uuid` field in all apps' models to `sodar_uuid` (sodar#166)
- **Projectroles**
  - Rename abstract `OmicsUser` model into `SODARUser` (sodar#166)
  - Rename `OMICS_CONSTANTS` into `SODAR_CONSTANTS` (sodar#166)
  - Rename the `omics_constant()` template tag into `sodar_constant()` (sodar#166)
  - Rename `omics_url` in sodar_taskflow tests to `sodar_url` (see sodar_taskflow#36)
  - Rename `shepherd-theme-omics.css` to `shepherd-theme-sodar.css` (sodar#166)

5.20.23 v0.2.0 (2018-09-19)

Added

- **General**
  - `example_backend_app` for a minimal backend app example
  - Backend app usage example in `example_project_app`
- **Timeline**
  - Add timeline app based on SODAR v0.3.2 (#2)
  - App documentation

Changed

- **General**
  - Update integration documentation (#1)
  - Restructure documentation files and filenames for clarity
- **Timeline**
  - Update CSS classes and overrides
  - Rename list views to `list_project` and `list_objects`
  - Rename list template to `timeline.html`
  - Refactor `api.get_event_description()`
  - Make `TIMELINE_PAGINATION` optional
  - Improve exception messages in `api.add_event()`

Fixed

- **Timeline**
  - User model access in `timeline.api`
  - Misaligned back button (#4)
  - Deprecated CSS in main list
- **Projectroles**
  - Third party apps not correctly recognized in `get_app_names()`

5.20.24 v0.1.0 (2018-09-12)

Added

- **General**
  - Create app package for Projectroles and other reusable apps based on SODAR release v0.3.1
  - `example_project_app` to aid testing and work as a minimal example
  - `example_site_app` for demonstrating site apps
  - `SITE_TITLE` and `SITE_INSTANCE_TITLE` settings
  - `SITE_PACKAGE` setting for explicitly declaring site path for code
  - Documentation for integration and development
  - Separate LDAP config in `install_ldap_dependencies.sh` and `requirements/ldap.txt`
- **Projectroles**
  - `static_file_exists()` and `template_exists()` helpers in common template tags
  - Abstract `OmicsUser` model
  - `get_full_name()` in abstract OmicsUser model

  - `auth_backends.py` file for LDAP backends (sodar#132)
  - Versioneer versioning
  - `core_version()` in common template tags
  - Check for footer content in `include/_footer.html`
  - Example of the site base template in `projectroles/base_site.html`
  - Example of project footer in `projectroles/_footer.html`
- **Userprofile**
  - Add site app `userprofile` with user details
  - Display user UUID in user profile

Changed

- **Projectroles**
  - Move custom modal into `projectroles/_modal.html`
  - Check for user.name in user dropdown
  - Move content block structure and sidebar inside `projectroles/base.html`
  - Move site title bar into optional include template `projectroles/_site_titlebar.html`
  - Move search form into optional include template `projectroles/_site_titlebar_search.html`
  - Make title bar dropdown inclueable as `_site_titlebar_dropdown.html`
  - Title bar CSS and layout tweaks
  - Move `search.js` under projectroles
  - Move projectroles specific javascript into `projectroles.js`
  - Move `site_version()` into common template tags
  - Move title bar admin and site app links to user dropdown (sodar#342)
  - Move project specific CSS into optionally includable `projectroles.css`
  - Refactor and cleanup CSS
  - Move `set_user_group()` into `projectroles.utils`
  - Move `syncgroups` management command into projectroles
  - Copy improved multi LDAP backend setup from flowcelltool (sodar#132)
  - Move LDAP authentication backends into projectroles (sodar#132)
  - Move `login.html` into projectroles
  - Display `SITE_INSTANCE_TITLE` in email instead of a hardcoded string
  - Display the first contact in `settings.ADMINS` in email footer
  - Use `get_full_name()` in email sending
  - Get site version using `SITE_PACKAGE`
  - Get LDAP domain names to login template from settings

  - Rename custom CSS classes and HTML IDs from `omics-*` into `sodar-*` (sodar#166)
  - Move Shepherd theme CSS files into projectroles

Fixed

- **Projectroles**
  - Tests referring to the `filesfolders` app not included in this project
  - `TestHomeView.test_render()` assumed extra SODAR system user was present (see sodar#367)
  - Tour link setup placing
- **Userprofile**
  - Missing user name if `name` field not filled in `user_detail.html`

Removed

- **Projectroles**
  - Deprecated Javascript variables `popupWaitHtml` and `popupNoFilesHtml`
  - Unused template `irods_info.html`

CHAPTER

SIX

INDICES AND TABLES

- modindex
- genindex
- search

PYTHON MODULE INDEX

p

s

t

INDEX

E

F

G

R

S

T

U

V