
RDMO Documentation Documentation

Release 1.1

RDMO project

Aug 04, 2020

Contents

1	Installation	3
1.1	Install prerequisites	3
1.2	Obtaining the app directory	5
1.3	Install Python packages	5
1.4	Setup the application	6
2	Deployment	7
2.1	Development server	7
2.2	Apache and mod_wsgi	8
2.3	Nginx and Gunicorn	9
3	Configuration	11
3.1	General settings	11
3.2	Databases	12
3.3	E-Mail	15
3.4	Authentication	15
3.5	Themes	20
3.6	Export formats	22
3.7	Cache	22
3.8	Logging	23
3.9	Plugins	25
3.10	Multisite	28
4	Administration	33
4.1	Site configuration	33
4.2	Users and Groups	34
4.3	Social accounts	35
4.4	API	36
5	Management	41
5.1	Questions	43
5.2	Domain	44
5.3	Options	46
5.4	Conditions	48
5.5	Views	50
5.6	Tasks	54
5.7	Export and Import	56

5.8	Role concept	56
6	Upgrade	59
6.1	Upgrade to version 0.9.0	60
6.2	Upgrade to version 0.14	60

RDMO is a tool to support the systematic planning, organisation and implementation of the data management throughout the course of a research project. RDMO is funded by the Deutsche Forschungsgemeinschaft (DFG).

Home Page <https://rdmorganiser.github.io>

Source code <https://github.com/rdmorganiser/rdmo>

Documentation <http://rdmo.readthedocs.io>

FAQ <http://www.forschungsdaten.org/index.php/FAQs>

Mailing list <https://www.listserv.dfn.de/sympa/subscribe/rdmo>

Slack <https://rdmo.slack.com>

Demo <https://rdmo.aip.de>

Warning: This software is currently under development and not production ready.

CHAPTER 1

Installation

For demonstration, development or testing purposes, RDMO can be installed on Linux, Windows and macOS. If you, however, to set up a production environment, serving RDMO over a Network or the Internet, we suggest that you use a recent Linux distribution, namely CentOS, Debian, or Ubuntu LTS.

The code is mainly written in Python and should work with a Python higher than 3.5.

An installation of RDMO contains of three parts:

1. A directory which holds all the settings and customisations, custom to your installation of RDMO. We will call this directory `rdmo-app`, but you can use any name you see fit.
2. The actual `rdmo` package, which is centrally maintained by the RDMO team, is installed as a dependency in a virtual environment.
3. A database to store the content, which is generated by the users of your RDMO installation. Currently, we support Postgres, MySQL, and SQLite.

This chapter shows how these components are set up. Optional components can be installed afterwards and are covered under [Configuration](#).

For testing and development, you can run RDMO using your regular user account. On a production system, a dedicated user account should be used. We suggest to create a user called `rdmo` with the group `rdmo` and the home directory `/srv/rdmo`: `sudo adduser rdmo --home /srv/rdmo`. We will use this user throughout this documentation.

Do not use the `root` user to run RDMO! It is a bad idea anyway and several steps of the installation will not work. `sudo` is used in the installation when needing root-privileges to install packages.

1.1 Install prerequisites

Installing the prerequisites for RDMO differs on the different operating systems and is therefore covered in different sections. Here, you need to use the superuser.

1.1.1 Linux

We recommend to install the prerequisites using the packaging system of your distribution. On Debian/Ubuntu use:

```
sudo apt install build-essential libxml2-dev libxslt-dev zlib1g-dev \
python3-dev python3-pip python3-venv \
git pandoc

# optional, for pdf output
sudo apt install texlive texlive-xetex
```

on RHEL/CentOS use:

```
sudo yum install gcc gcc-c++ libxml2-devel libxslt-devel \
python34-devel python34-pip python34-virtualenv \
git pandoc

# optional, for pdf output
sudo yum install texlive texlive-xetex texlive-mathspec texlive-euenc \
texlive-xetex-def texlive-xltxtra
```

On Ubuntu 14.04, `python3-venv` is not available. Please use `python3.5-venv` instead.

On RHEL/CentOS `selinux` is enabled by default. This can result in unexpected errors, depending on where you store the RDMO source code on the system. While the preferable way is to configure it correctly (which is beyond the scope of this documentation), you can also set `selinux` to permissive or disabled in `/etc/selinux/config` (and reboot afterwards).

1.1.2 macOS

We recommend to install the prerequisites using `brew`:

```
brew install python3
brew install git
brew install pandoc

# optional, for pdf export
brew install texlive
```

1.1.3 Windows

On Windows, the software prerequisites need to be downloaded and installed from their particular web sites.

For python:

- download from <https://www.python.org/downloads/windows/>
- don't forget to check 'Add Python to environment variables' during setup

For git:

- download from <https://git-for-windows.github.io/>

For the Microsoft C++ Build Tools:

- download from <https://wiki.python.org/moin/WindowsCompilers>

For Pandoc:

- download from <https://github.com/jgm/pandoc/releases>

For pdflatex (optional, for pdf export):

- download from <http://miktex.org/>

All further steps need to be performed using the windows shell `cmd.exe`. You can open it from the Start-Menu.

1.2 Obtaining the app directory

The next step is to create the `rdmo-app` directory by cloning the corresponding repository into the home directory of you `rdmo` user:

```
git clone https://github.com/rdmorganiser/rdmo-app
```

Note that this is not the main `rdmo` repository, only the configuration files. Inside this directory, you will find:

- a `config` directory, containing the main settings of your RDMO installation,
- a `requirements` directory, containing shortcuts to install the different mandatory and optional dependencies, and
- a `manage.py` script, which is the main way to interact with your RDMO installation on the command line. Most of the following steps will use this script.

The `rdmo-app` directory corresponds to a `project` in Django terms.

1.3 Install Python packages

After you have obtained the `rdmo-app`, you need to install the `rdmo` package and the other python dependencies.

Change to the `rdmo-app` directory and create a `Virtual Environment` (this is done as your user or the created `rdmo` user, not as `root`):

```
cd rdmo-app

python3 -m venv env

source env/bin/activate           # on Linux or macOS
call env\Scripts\activate.bat    # on Windows

pip install --upgrade pip setuptools  # update pip and setuptools
```

After the virtual environment is activated, the `rdmo` package can be installed using `pip`:

```
pip install rdmo
```

The virtual environment encapsulates your RDMO installation from the rest of the system. This makes it possible to run several applications with different python dependencies on one machine and to install the dependencies without root permissions.

Important: The virtual enviroment needs to be activated, using `source env/bin/activate` or `call env\Scripts\activate.bat`, everytime a new terminal is used.

1.4 Setup the application

1.4.1 Basic setup

To set up the application, create a new file `config/settings/local.py` in your cloned `rdmo-app` directory. For the example user with the home `/srv/rdmo`, this would now be `/srv/rdmo/rdmo-app/config/settings/local.py`.

You can use `config/settings/sample.local.py` as template, i.e.:

```
cp config/settings/sample.local.py config/settings/local.py    # on Linux or macOS
copy config\settings\sample.local.py config\settings\local.py  # on Windows
```

Most of the settings of your RDMO instance are specified in this file. The different settings are explained in detail [later in the documentation](#). For a minimal configuration, you need to set `DEBUG = True` to see verbose error messages and serve static files, and `SECRET_KEY` to a long random string, which you will keep secret. Your database connection is configured using the `DATABASES` variable. Database configuration is covered [later in the documentation](#). If no `DATABASE` setting is given `sqlite3` will be used as database backend.

Then, initialize the application, using:

```
python manage.py migrate                # initializes the database
python manage.py setup_groups            # creates groups with different permissions
python manage.py createsuperuser        # creates the admin user
```

1.4.2 Third party vendor files

By default third party vendor files (like jQuery or Bootstrap javascripts) are retrieved from the content delivery networks that they are hosted on. If you would like to avoid third party requests you could host them yourself. This can be achieved easily with two simple steps.

1. download the vendor files from the cdns by running the provided script

```
python manage.py download_vendor_files
```

2. make sure your `local.py` does contain the following line

```
VENDOR_CDN = False
```

1.4.3 RDMO development server

After these steps, RDMO can be run using Django's integrated development server:

```
python manage.py runserver
```

Then, RDMO is available on `http://127.0.0.1:8000` in your (local) browser. The different ways RDMO can be deployed are covered in the next chapter. The newly installed RDMO instance is still empty, i.e. no questionnaire or views are available. They need to be [imported](#) and/or created as described under [Management](#).

CHAPTER 2

Deployment

As already mentioned, RDMO can be run in two different setups:

1. For development or testing, using the build-in Django development server Information about how to do that can be found in the [docs of the RDMO repository](#).
 2. In production, using a web server and the [wsgi](#) protocol We suggest to use one of the following setups:
 - [Apache2 and mod_wsgi](#) (Shibboleth can only be used with this version.)
 - [nginx, gunicorn and systemd](#)
-

2.1 Development server

Django comes with an integrated development server. It can be started using:

```
python manage.py runserver
```

Then, RDMO is available on `http://localhost:8000` in your (local) browser. The development server is **not** suited to serve the application to the internet.

If you want the development server to be accessible from other machines in your local network you need to use:

```
python manage.py runserver 0.0.0.0:8000
```

where 8000 is the port and can be changed according to your needs. Please do not use this setup for more than testing and development, it is not secure. The Django development server does only serve the static files correctly if you have debug mode enabled. Therefore please do not forget to change the line `DEBUG = False` in your `config/settings/local.py` to `DEBUG = True` if you want to use it.

More information about the development server can be found in the [Django documentation](#).

2.2 Apache and mod_wsgi

In production, you should create a dedicated user for RDMO. All steps for the installation, which do not need root access, should be done using this user. As before, we assume this user is called `rdmo` and it's home is `/srv/rdmo` and therefore your `rdmo-app` is located in `/srv/rdmo/rdmo-app`.

Install the Apache server and `mod_wsgi` on Debian or Ubuntu using:

```
sudo apt install apache2 libapache2-mod-wsgi-py3
```

On CentOS7 you need to enable the [IUS repository](#) first. Then install using:

```
sudo yum install httpd python35u-mod_wsgi
```

Next, create a virtual host configuration. Unfortunately, the different distributions use different versions of Apache and `mod_wsgi` and therefore require a slightly different setup:

For Debian/Ubuntu in `/etc/apache2/sites-available/000-default.conf` use:

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    Alias /static /srv/rdmo/rdmo-app/static_root/
    <Directory /srv/rdmo/rdmo-app/static_root/>
        Require all granted
    </Directory>

    WSGIDaemonProcess rdmo user=rdmo group=rdmo \
        home=/srv/rdmo/rdmo-app python-home=/srv/rdmo/rdmo-app/env
    WSGIProcessGroup rdmo
    WSGIScriptAlias / /srv/rdmo/rdmo-app/config/wsgi.py process-group=rdmo
    WSGIPassAuthorization On

    <Directory /srv/rdmo/rdmo-app/config/>
        <Files wsgi.py>
            Require all granted
        </Files>
    </Directory>
</VirtualHost>
```

for CentOS 7 in `/etc/httpd/conf.d/vhosts.conf` use:

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost

    DocumentRoot /var/www/html/

    Alias /static /srv/rdmo/rdmo-app/static_root/
    <Directory /srv/rdmo/rdmo-app/static_root/>
        Require all granted
    </Directory>

    WSGIDaemonProcess rdmo user=rdmo group=rdmo home=/srv/rdmo/rdmo-app \
```

(continues on next page)

(continued from previous page)

```

python-path=/srv/rdmo/rdmo-app:/srv/rdmo/rdmo-app/env/lib/python2.7/site-
→packages
WSGIProcessGroup rdmo
WSGIScriptAlias / /srv/rdmo/rdmo-app/config/wsgi.py process-group=rdmo
WSGIPassAuthorization On

<Directory /srv/rdmo/rdmo-app/config/>
  <Files wsgi.py>
    Require all granted
  </Files>
</Directory>
</VirtualHost>

```

Restart the Apache server: `sudo service apache2 restart`. RDMO should now be available on YOURDOMAIN. Note that the Apache user needs to have access to `/srv/rdmo/rdmo-app/static_root/`.

As you can see from the virtual host configurations, the static assets such as CSS and JavaScript files are served independently from the WSGI-python script. In order to do so, they need to be gathered in the `static_root` directory. This can be achieved by running:

```
python manage.py collectstatic --clean
```

in your virtual environment (`--clean` removes existing files before collecting).

In order to apply changes to the RDMO code (e.g. after an `upgrade`), the webserver needs to be reloaded or the `config/wsgi.py` file needs to appear modified. This can be done using the `touch` command:

```
touch config/wsgi.py
```

Also, the `collectstatic` command has to be executed again. Both can be achieved by using

```
python manage.py deploy
```

in your virtual environment.

2.3 Nginx and Gunicorn

As mentioned several times, you should create a dedicated user for RDMO. All steps for the installation, which do not need root access, should be done using this user. Here we assume this user is called `rdmo` and it's home is `/srv/rdmo` and therefore your `rdmo-app` is located in `/srv/rdmo/rdmo-app`.

First install gunicorn inside your virtual environment:

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

Then, test gunicorn using:

```
gunicorn --bind 0.0.0.0:8000 config.wsgi:application
```

This should serve the application like `runserver`, but without the static assets, like CSS files and images. After the test kill the gunicorn process again.

Systemd will launch the Gunicorn process on startup and keep running. Create a new systemd service file in `/etc/systemd/system/rdmo.service` and enter (you will need root/sudo permissions for that):

```
[Unit]
Description=RDMO gunicorn daemon
After=network.target

[Service]
User=rdmo
Group=rdmo
WorkingDirectory=/srv/rdmo/rdmo-app
ExecStart=/srv/rdmo/rdmo-app/env/bin/gunicorn \
    --bind unix:/srv/rdmo/rdmo.sock config.wsgi:application

[Install]
WantedBy=multi-user.target
```

This service needs to be started and enabled like any other service:

```
sudo systemctl start rdmo
sudo systemctl enable rdmo
```

Next, install Nginx:

```
sudo apt install nginx # on Debian/Ubuntu
sudo yum install nginx # on RHEL/CentOS
```

Edit the Nginx configuration as follows (again with root/sudo permissions):

```
# in /etc/nginx/sites-available/default on Debian/Ubuntu
# in /etc/nginx/conf.d/vhost.conf      on RHEL/CentOS
server {
    listen 80;
    server_name YOURDOMAIN;

    location / {
        proxy_pass http://unix:/srv/rdmo/rdmo.sock;
    }
    location /static/ {
        alias /srv/rdmo/rdmo-app/static_root/;
    }
}
```

Restart Nginx. RDMO should now be available on YOURDOMAIN. Note that the unix socket `/srv/rdmo/rdmo.sock` needs to be accessible by Nginx.

As you can see from the virtual host configurations, the static assets such as CSS and JavaScript files are served independently from the reverse proxy to the gunicorn process. In order to do so they need to be gathered in the `static_root` directory. This can be achieved by running:

```
python manage.py collectstatic --clean
```

in your virtual environment (`--clean` removes existing files before collecting).

In order to apply changes to the RDMO code (e.g. after an [upgrade](#)), the Gunicorn process needs to be restarted:

```
sudo systemctl restart rdmo
```

CHAPTER 3

Configuration

The RDMO application uses the [Django settings](#) module for its configuration. To separate the base configuration and your local adjustments and secret information (e.g. database connections), RDMO splits the settings into two files:

- `config/settings/base.py`, which is part of the git repository.
- `config/settings/local.py`, which is ignored by git.

As part of the installation `config/settings/local.py` should be created from the template `config/settings/sample.local.py`.

While technically the local settings file `config/settings/local.py` can be used to override all of the settings in `config/settings/sample.local.py`, it should be used to customize the settings already available in `config/settings/sample.local.py`.

This comprises [general settings](#), [database connections](#), how to send [emails](#), the different [authentication methods](#), the usage of [themes](#), and [caches](#).

3.1 General settings

A few general settings should be included in your `config/settings/local.py`. The first, and probably most important one, is if you run RDMO in [debug mode](#) or not:

```
DEBUG = True
```

In debug mode, verbose error pages are shown in the case something goes wrong and static assets such as CSS and JavaScript files are found by the development server automatically. The debug mode **must not** be enabled when running RDMO in production connected to the internet.

Django needs a [secret key](#), which “should be set to a unique, unpredictable value”:

```
SECRET_KEY = 'this is not a very secret key'
```

This key must be kept secret since otherwise many of Django's security protections fail.

In production, Django only [allows requests to certain urls](#), which you need to specify:

```
ALLOWED_HOSTS = ['localhost', 'rdmo.example.com']
```

If you want to run RDMO under an alias like `http://example.com/rdmo`, you need to set the base URL:

```
BASE_URL = '/rdmo'
```

Furthermore, you might want to choose the main language for RDMO and the timezone:

```
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'Europe/Berlin'
```

By default, RDMO runs with English as the first and German as the second language. It can be configured to run with up to 5 arbitrary languages. For this, the `LANGUAGES` setting need to be added to `config/settings/local.py`. E.g. for French:

```
from django.utils.translation import ugettext_lazy as _
LANGUAGE_CODE = 'fr-fr'
TIME_ZONE = 'Europe/Paris'
LANGUAGES = (
    ('fr', _('French')),
    ('en', _('English')),
)
```

This needs to be configured before data is imported or content is configured.

Note that in order to use RDMO with any language a `.po` file needs to be created (see also: <https://docs.djangoproject.com/en/stable/topics/i18n/translation/>), which contains the translation of all strings in the user interface. RDMO ships with such a file for English and German. Please contact the project if you intend to use RDMO with a new language. Although we will not be able to perform the translation ourselves, we are happy to support you and add the language file to the RDMO source code after review. This translation of stings in the user interface via `gettext` is independent from the creation/translation from any content (e.g. catalogs, option set), which need to be performed independently through the management interface.

If you run RDMO behind a reverse Proxy, which terminates the TLS/SSL traffic, you need to add the following:

```
USE_X_FORWARDED_HOST = True
SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')
```

in order for RDMO to pick up the `X-Forwarded-Host` and `X-Forwarded-Proto` HTTP headers from the proxy.

3.2 Databases

RDMO can be used with all kind of databases supported by the Django framework. The particular database connection is defined using the setting `DATABASE`. An overview about the Django database settings is given [here](#). In the following, we show the settings for PostgreSQL, MySQL, and SQLite.

3.2.1 PostgreSQL

PostgreSQL can be installed using:


```
# Debian/Ubuntu
sudo apt install postgresql

# CentOS
sudo yum install postgresql-server postgresql-contrib
sudo postgresql-setup initdb
sudo systemctl start postgresql
sudo systemctl enable postgresql
```

To use PostgreSQL as your database backend install `psycpg2` in your virtual environment:

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

Then, add the following to your `config/settings/local.py`:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycpg2',
        'NAME': 'rdmo',
        'USER': 'rdmo',
        'PASSWORD': '',
        'HOST': '',
        'PORT': '',
    }
}
```

where `Name` is the name of the database, `USER` the PostgreSQL user, `PASSWORD` her password, `HOST` the database host, and `PORT` the port PostgreSQL is listening on. Note that, depending on your setup, not all settings are needed. If you are using the peer authentication methods, you only need the `NAME` and `ENGINE` settings. The user and the database can be created using:

```
sudo su - postgres
createuser rdmo
createdb rdmo -O rdmo
```

This assumes peer authentication for the `rdmo` user.

The command

```
python manage.py migrate
```

should now create the RDMO database tables on PostgreSQL.

3.2.2 MySQL

MySQL (or community-developed fork MariaDB) can be installed using:

```
# Debian/Ubuntu
sudo apt install mysql-client mysql-server libmysqlclient-dev      # for MySQL
sudo apt install mariadb-client mariadb-server libmariadbclient-dev # for MariaDB

# CentOS
sudo yum install -y mysql mysql-server mysql-devel                # for_
↪MySQL
sudo yum install -y mariadb mariadb-server mariadb-devel          # for_
↪MariaDB
```

(continues on next page)

(continued from previous page)

```
sudo systemctl enable mariadb
sudo systemctl start mariadb
sudo mysql_secure_installation
```

To use MySQL as your database backend install `mysqlclient` in your virtual environment:

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

Then, add the following to your `config/settings/local.py`:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'rdmo',
        'USER': 'rdmo',
        'PASSWORD': 'not a good password',
        'HOST': '',
        'PORT': '',
        'OPTIONS': {
            # only if you want to connect over a non-default socket
            'unix_socket': '',
            # only for MySQL 5.7
            'init_command': "SET GLOBAL sql_mode=(SELECT REPLACE(@@sql_mode, 'ONLY_
↪FULL_GROUP_BY', ''));"
        }
    }
}
```

to your `config/settings/local.py`. Here, Name is the name of the database, USER the MySQL user, PASSWORD her password, HOST the database host, and PORT the port MySQL is listening on. If you don't use `/tmp/mysql.sock`, you can use `unix_socket` to specify its path. The user and the database can be created using:

```
CREATE USER 'rdmo'@'localhost' identified by 'not a good password';
GRANT ALL ON `rdmo`.* to 'rdmo'@'localhost';
CREATE DATABASE `rdmo` CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

on the MySQL-shell.

The command

```
python manage.py migrate
```

should now create the RDMO database tables on MySQL.

3.2.3 SQLite

SQLite ist the default option in RDMO and configured in `config/settings/base.py`. We recommend it only for a development/testing setup. It can be configured in `config/settings/local.py` by adding:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': '',
    }
}
```

where Name is the name of database file.

The command

```
python manage.py migrate
```

should now create RDMO database tables in the specified database file.

3.3 E-Mail

RDMO needs to send E-Mails to its users. The connection to the SMTP server is configured by several settings in your `config/settings/local.py`:

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'localhost'
EMAIL_PORT = '25'
EMAIL_HOST_USER = ''
EMAIL_HOST_PASSWORD = ''
EMAIL_USE_TLS = False
EMAIL_USE_SSL = False

DEFAULT_FROM_EMAIL = ''
```

Here, `EMAIL_HOST` is the URL or IP of the SMTP server, `EMAIL_PORT` is the port it is listening on (usually 25, 465, or 587), and `EMAIL_HOST_USER` and `EMAIL_HOST_PASSWORD` are credentials, if the SMTP server needs authentication.

For a STARTTLS connection (usually on port 587) `EMAIL_USE_TLS` needs to be set to `True`, while `EMAIL_USE_SSL` needs to be set to `True` for an implicit TLS/SSL connection (usually on port 465).

`DEFAULT_FROM_EMAIL` sets the FROM field for the E-mails send to the users.

For a development/testing setup a simple E-mail backend, which only displays the E-mail on the terminal, can be used:

```
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
EMAIL_FROM = 'info@example.com'
```

This is also the default backend, if no E-mail settings are added to `config/settings/local.py`.

3.4 Authentication

RDMO has three main modes for Authentication:

- Regular user accounts with registration using the [django-allauth](#) library.
- Using a (read-only) connection to a [LDAP server](#)
- Installing a [Shibboleth](#) service provider next to RDMO and connect to an identity provider or even a whole Shibboleth federation.

Important: These modes are only tested individually and may be treated **mutually exclusive** (unless proven otherwise).

If none of the modes is enabled, only a very basic login will be available and users need to be created using the Django Admin Interface.

3.4.1 Django-allauth

RDMO uses the excellent [django-allauth](#) as its main authorization library. It enables workflows for user registration and password retrieval, as well as authentication from 3rd party sites using OAUTH2.

Accounts

To enable regular accounts in RDMO add:

```
from rdm.core.settings import INSTALLED_APPS, AUTHENTICATION_BACKENDS

ACCOUNT = True
ACCOUNT_SIGNUP = True
ACCOUNT_TERMS_OF_USE = False

INSTALLED_APPS += [
    'allauth',
    'allauth.account',
]

AUTHENTICATION_BACKENDS.append('allauth.account.auth_backends.AuthenticationBackend')
```

to your config/settings/local.py. The setting `ACCOUNT = True` enables the general django-allauth features in RDMO, while `ACCOUNT_SIGNUP = True` enables new users to register with your RDMO instance. `ACCOUNT_TERMS_OF_USE = False` disables the Terms of Use. If you set it to `True` every registering user will have to agree to your policy. The last lines enable django-allauth to be used by RDMO.

The behavior of django-allauth can be further configured by the settings documented in the [django-allauth documentation](#). RDMO sets some defaults, which can be found in config/settings/base.py.

Social accounts

In order to use 3rd party accounts (facebook, github, etc.) with RDMO add:

```
from rdm.core.settings import INSTALLED_APPS, AUTHENTICATION_BACKENDS

ACCOUNT = True
ACCOUNT_SIGNUP = True
SOCIALACCOUNT = True
SOCIALACCOUNT_SIGNUP = False
SOCIALACCOUNT_AUTO_SIGNUP = False

INSTALLED_APPS += [
    'allauth',
    'allauth.account',
    'allauth.socialaccount',
    'allauth.socialaccount.providers.facebook',
    'allauth.socialaccount.providers.github',
    'allauth.socialaccount.providers.google',
    'allauth.socialaccount.providers.orcid',
    'allauth.socialaccount.providers.twitter',
    ...
]

AUTHENTICATION_BACKENDS.append('allauth.account.auth_backends.AuthenticationBackend')
```

to your `config/settings/local.py`. The setting `SOCIALACCOUNT = True` is used by RDMO to show certain parts of the user interface connected to 3rd party accounts, while as before, the lines after `INSTALLED_APPS` enable the feature to be used by RDMO. `SOCIALACCOUNT_AUTO_SIGNUP = True` forces new users to fill out a signup form even if the provider does provide the email address. Each provider has a separate app you need to add to `INSTALLED_APPS`. A list of all providers supported by django-allauth can be found [here](#).

Once the installation is complete, the credentials of your OAUTH provider need to be entered in the admin interface. This is covered in the [administration chapter](#) of this documentation.

3.4.2 LDAP

In order to use a LDAP backend with RDMO you need to install some prerequisites. On Debian/Ubuntu you can install them using:

```
sudo apt-get install libsasl2-dev libldap2-dev libssl-dev
```

On the python side, we use `django-auth-ldap` to connect to the LDAP server. As before, it should be installed inside the virtual environment created for RDMO using:

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

LDAP installations can be very different and we only discuss one particular example. We assume that the LDAP service is running on `ldap.example.com`. RDMO needs an account to connect to the LDAP. In order to create it, run:

```
ldapmodify -x -D "cn=admin,dc=ldap,dc=example,dc=com" -W
```

on the machine running the LDAP service and type in:

```
dn: uid=rdmo,dc=ldap,dc=example,dc=com
changetype: add
objectclass: account
objectclass: simplesecurityobject
uid: rdmo
userPassword: YOURPASSWORD
```

and end with a blank line followed by `ctrl-d`.

Then, in your `config/settings/local.py` add or uncomment:

```
import ldap
from django_auth_ldap.config import LDAPSearch
from rdmo.core.settings import AUTHENTICATION_BACKENDS

PROFILE_UPDATE = False
PROFILE_DELETE = False

AUTH_LDAP_SERVER_URI = "ldap://ldap.example.com"
AUTH_LDAP_BIND_DN = "cn=rdmo,dc=ldap,dc=example,dc=com"
AUTH_LDAP_BIND_PASSWORD = "YOURPASSWORD"
AUTH_LDAP_USER_SEARCH = LDAPSearch("dc=ldap,dc=example,dc=com", ldap.SCOPE_SUBTREE,
    ↪ "(uid=%(user)s)")

AUTH_LDAP_USER_ATTR_MAP = {
    "first_name": "givenName",
    "last_name": "sn",
```

(continues on next page)

(continued from previous page)

```
'email': 'mail'
}

AUTHENTICATION_BACKENDS.insert(
    AUTHENTICATION_BACKENDS.index('django.contrib.auth.backends.ModelBackend'),
    'django_auth_ldap.backend.LDAPBackend'
)
```

The setting `PROFILE_UPDATE = False` and `PROFILE_DELETE = False` tell RDMO to disable the update and deletion form for the user profile so that users can neither update their credentials nor delete their profile anymore. The other settings are needed by `django-auth-ldap` and are described in the [django-auth-ldap documentation](#).

Warning: The following feature is not available in the released version of RDMO yet. It will be part of a future version.

You can also map LDAP groups to Django groups, in particular to restrict the access to Catalogs and Views. This can be done by adding the following settings:

```
AUTH_LDAP_GROUP_SEARCH = LDAPSearch(
    "dc=ldap,dc=test,dc=rdmo,dc=org", ldap.SCOPE_SUBTREE, "(objectClass=groupOfNames)"
)
AUTH_LDAP_GROUP_TYPE = GroupOfNamesType(name_attr="cn")
AUTH_LDAP_MIRROR_GROUPS = ['special']
```

where `cn` is the name of the particular group in the LDAP and `AUTH_LDAP_MIRROR_GROUPS` denotes the groups which are actually mirrored from the LDAP.

3.4.3 Shibboleth

In order to use Shibboleth with RDMO, it needs to be deployed in a production environment using Apache2. The Setup is documented [here](#).

Next, install the Shibboleth Apache module for the service provider (SP) from your distribution repository, e.g. for Debian/Ubuntu:

```
sudo apt-get install libapache2-mod-shib2
```

Under Ubuntu 18.04 LTS the `libapache2-mod-shib2` package is broken. A working description on how to install the SP can be found under <https://www.switch.ch/aai/guides/sp/installation/?os=ubuntu>.

In addition, `django-shibboleth-remoteuser` needs to be installed in your RDMO virtual environment:

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

Configure your Shibboleth service provider using the files in `/etc/shibboleth/`. This may vary depending on your Identity Provider. RDMO needs the `REMOTE_SERVER` to be set and 4 attributes from your identity provider:

- a username (usually `eppn`)
- an E-mail address (usually `mail` or `email`)
- a first name (usually `givenName`)
- a last name (usually `sn`)

In our test environent this is accomplished by editing `/etc/shibboleth/shibboleth2.xml`:

```
<ApplicationDefaults entityID="https://sp.test.rdm.org/shibboleth"
  REMOTE_USER="eppn"
  cipherSuites="DEFAULT:!EXP:!LOW:!aNULL:!eNULL:!DES:!IDEA:!SEED:!RC4:!3DES:!kRSA:!
  →SSLv2:!SSLv3:!TLSv1:!TLSv1.1">
```

and /etc/shibboleth/attribute-map.xml:

```
<Attribute name="urn:oid:1.3.6.1.4.1.5923.1.1.1.6" id="eppn">
  <AttributeDecoder xsi:type="ScopedAttributeDecoder" caseSensitive="false"/>
</Attribute>
<Attribute name="urn:oid:0.9.2342.19200300.100.1.1" id="uid"/>
<Attribute name="urn:oid:2.5.4.4" id="sn"/>
<Attribute name="urn:oid:2.5.4.42" id="givenName"/>
<Attribute name="urn:oid:0.9.2342.19200300.100.1.3" id="mail"/>
```

Restart the Shibboleth service provider demon.

```
service shibd restart
```

In your Apache2 virtual host configuration, add:

```
<Location /Shibboleth.sso>
  SetHandler shib
</Location>
<LocationMatch /(account|domain|options|projects|questions|tasks|conditions|views)>
  AuthType shibboleth
  require shibboleth
  ShibRequireSession On
  ShibUseHeaders On
</LocationMatch>
```

In your config/settings/local.py add or uncomment:

```
from rdm.core.settings import INSTALLED_APPS, AUTHENTICATION_BACKENDS, MIDDLEWARE

SHIBBOLETH = True
PROFILE_UPDATE = False
PROFILE_DELETE = False

INSTALLED_APPS += ['shibboleth']

AUTHENTICATION_BACKENDS.append('shibboleth.backends.ShibbolethRemoteUserBackend')
MIDDLEWARE.insert(
    MIDDLEWARE.index('django.contrib.auth.middleware.AuthenticationMiddleware') + 1,
    'shibboleth.middleware.ShibbolethRemoteUserMiddleware'
)

SHIBBOLETH_ATTRIBUTE_MAP = {
    'uid': (True, 'username'),
    'givenName': (True, 'first_name'),
    'sn': (True, 'last_name'),
    'mail': (True, 'email'),
}

LOGIN_URL = '/Shibboleth.sso/Login?target=/projects'
LOGOUT_URL = '/Shibboleth.sso/Logout'
```

where the keys of SHIBBOLETH_ATTRIBUTE_MAP, LOGIN_URL, and LOGOUT_URL need to be modified accord-

ing to your setup. The setting `SHIBBOLETH = True` disables the regular login form in RDMO, and tells RDMO to disable the update or delete form for the user profile, so that users can neither update their credentials nor delete their profile anymore. Note that profile deletion is technically impossible because RDMO can delete from the Shibboleth database. The `INSTALLED_APPS`, `AUTHENTICATION_BACKENDS`, and `MIDDLEWARE` settings enable `django-shibboleth-remoteuser` to be used with RDMO.

Restart the webserver.

```
service apache2 restart
```

Warning: The following feature is not available in the released version of RDMO yet. It will be part of a future version.

Certain Attributes from the Shibboleth Identity Provider can be mapped to Django groups, in particular to restrict the access to Catalogs and Views. This can be done by adding the following settings:

```
SHIBBOLETH_GROUP_ATTRIBUTES = ['eduPersonScopedAffiliation']
```

In this case, the attribute `eduPersonScopedAffiliation` contains a comma separated list of groups which will be created in RDMO (if they don't exist yet) and the user will be added into these groups. Due to a limitation by `django-shibboleth-remoteuser` the user will also be **removed from all other groups**.

3.5 Themes

3.5.1 Introduction

RDMO is based on Django, which allows a high level of customization by modifying the Django *templates* as well as the *static assets* (CSS files, images, etc.). There is a powerful method by which the set of files getting imported can be changed. If you create a `theme` folder in your `rdmo-app` directory the framework will use the files in this folder and not their counterparts from the RDMO source code. By doing this you can override any template or static file you desire as long as you get the folder structure right. There are two ways to create a theme.

3.5.2 Create automatically

There is a `manage.py` script to simplify the theme creation. The script will create the `theme/static` and `theme/templates` folders, copy a basic set of files into them and add the necessary configuration line to `config/settings/local.py` to enable the theme. If you want to override other files, please put them into the necessary folders manually. To run the script do:

```
python manage.py make_theme
```

3.5.3 Create manually

If you want to manually create a theme you need to do the following:

1. Create a `theme` folder containing a `static` and a `templates` directory inside your `rdmo-app` directory:

```
mkdir -p theme/static theme/templates
```

2. Add the line below to your `config/settings/local.py`:


```
import os
from . import BASE_DIR

THEME_DIR = os.path.join(BASE_DIR, 'theme')
```

3. Copy the files you want to modify into the `theme` folder keeping their relative paths.

3.5.4 Working with themes

If you have completed the steps above templates and static files in the `theme` directory are used instead of the original files as long as they have the same relative path, e.g. the template `theme/templates/core/base_navigation.html` overrides `rdmo/core/templates/core/base_navigation.html` from the original code.

Usually, the RDMO template files are located in your virtual environment, e.g. `/srv/rdmo/rdmo-app/env/lib/python3.6/site-packages/rdmo/core/static/core/css/variables.scss`. The exact path depends on your Python version and platform. We recommended to download the original files from the [rdmo repository](https://raw.githubusercontent.com/rdmorganiser/rdmo/master/rdmo/core/static/core/css/variables.scss) instead. For the example above, this would be <https://raw.githubusercontent.com/rdmorganiser/rdmo/master/rdmo/core/static/core/css/variables.scss>. Please make sure to use the raw files when downloading from GitHub. If you accidentally grab the website's html source code RDMO will throw an error.

Some files you might want to override are:

SASS variables

<https://github.com/rdmorganiser/rdmo/blob/master/rdmo/core/static/core/css/variables.scss> can be copied to `theme/static/core/css/variables.scss` and be used to customize colors.

Navigation bar

https://github.com/rdmorganiser/rdmo/blob/master/rdmo/core/templates/core/base_navigation.html can be copied to `theme/templates/core/base_navigation.html` and be used to customize the navbar.

Home page text

https://github.com/rdmorganiser/rdmo/blob/master/rdmo/core/templates/core/home_text_en.html and https://github.com/rdmorganiser/rdmo/blob/master/rdmo/core/templates/core/home_text_de.html can be copied to `theme/templates/core/home_text_en.html` and `theme/templates/core/home_text_de.html` and be used to customize text on the home page.

Terms of Use

The content displayed in the Terms of Use dialogue can be customized by putting templates to the appropriate locations. Two different files resembling the available languages can be used and should be located at `theme/templates/account/terms_of_use_de.html` and `theme/templates/account/terms_of_use_en.html`. Set the variable `ACCOUNT_TERMS_OF_USE` to `True` in your `config/settings/local.py`.

Note that updates to the RDMO package might render your theme incompatible to the RDMO code and cause errors. In this case the files in `theme` need to be adjusted to match their RDMO counterparts in functionality.

3.6 Export formats

RDMO supports exports to certain formats using the excellent [Pandoc](#) converter. The list of formats to select can be customized by changing the `EXPORT_FORMATS` setting in your `config/settings/local.py`.

```
EXPORT_FORMATS = (  
    ('pdf', _('PDF')),  
    ('rtf', _('Rich Text Format')),  
    ('odt', _('Open Office')),  
    ('docx', _('Microsoft Office')),  
    ('html', _('HTML')),  
    ('markdown', _('Markdown')),  
    ('mediawiki', _('mediawiki')),  
    ('tex', _('LaTeX'))  
)
```

The different formats supported by Pandoc can be found on the [Pandoc homepage](#).

The page style and different other format settings can be adjusted using reference documents. Since Pandoc is used for document conversion you can find an exhaustive list of supported settings on the [Pandoc manual page](#) under the paragraph ‘–reference-doc’. Reference documents can be used for the export formats of `.docx` and `.odt`. This is achieved by defining reference documents in the `config/settings/local.py`. If these settings are not in your config default styles will be applied.

```
EXPORT_REFERENCE_DOCX='FULL PATH OF YOUR REFERENCE DOCX'  
EXPORT_REFERENCE_ODT='FULL PATH OF YOUR REFERENCE ODT'
```

3.7 Cache

RDMO uses a cache for some of its pages. In the development setup, this is done using local-memory caching. In production, we suggest using [memcached](#). Memcached can be installed on Debian/Ubuntu using:

```
sudo apt install memcached
```

On RHEL/CentOS a few more steps are needed. First install the package using:

```
sudo yum install memcached
```

Then edit the settings file to prevent external connections:

```
# in /etc/sysconfig/memcached  
PORT="11211"  
USER="memcached"  
MAXCONN="1024"  
CACHE_SIZE="64"  
OPTIONS="-l 127.0.0.1"
```

Then start the service:

```
systemctl start memcached  
systemctl enable memcached
```

Back in your virtual environment, you need to install `python-memcached`:

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

and add the following to your config/settings/local.py:

```
CACHES = {
    {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
        'KEY_PREFIX': 'rdmo_default'
    },
    'api': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
        'KEY_PREFIX': 'rdmo_api'
    }
}
```

3.8 Logging

Logging in Django can be very complex and is covered extensively in the [Django documentation](#). For a suitable logging of RDMO you can add the following to your config/settings/local.py:

```
import os

LOG_LEVEL = 'INFO'          # or 'DEBUG' for the full logging
LOG_DIR = '/var/log/rdmo/'  # this directory needs to be writable by the rdmo user
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'filters': {
        'require_debug_false': {
            '()': 'django.utils.log.RequireDebugFalse'
        },
        'require_debug_true': {
            '()': 'django.utils.log.RequireDebugTrue'
        }
    },
    'formatters': {
        'default': {
            'format': '[%(asctime)s] %(levelname)s: %(message)s'
        },
        'name': {
            'format': '[%(asctime)s] %(levelname)s %(name)s: %(message)s'
        }
    },
    'handlers': {
        'mail_admins': {
            'level': 'ERROR',
            'filters': ['require_debug_false'],
            'class': 'django.utils.log.AdminEmailHandler'
        },
        'error_log': {
            'level': 'ERROR',
            'class': 'logging.FileHandler',
            'filename': os.path.join(LOG_DIR, 'error.log'),
```

(continues on next page)

(continued from previous page)

```

        'formatter': 'default'
    },
    'ldap_log': {
        'level': 'DEBUG',
        'class': 'logging.FileHandler',
        'filename': os.path.join(LOG_DIR, 'ldap.log'),
        'formatter': 'name'
    },
    'rules_log': {
        'level': 'DEBUG',
        'class': 'logging.FileHandler',
        'filename': os.path.join(LOG_DIR, 'rules.log'),
        'formatter': 'name'
    },
    'rdmo_log': {
        'level': 'DEBUG',
        'class': 'logging.FileHandler',
        'filename': os.path.join(LOG_DIR, 'rdmo.log'),
        'formatter': 'name'
    }
},
'loggers': {
    'django.request': {
        'handlers': ['mail_admins', 'error_log'],
        'level': 'ERROR',
        'propagate': True
    },
    'django_auth_ldap': {
        'handlers': ['ldap_log'],
        'level': LOG_LEVEL,
        'propagate': True
    },
    'rules': {
        'handlers': ['rules_log'],
        'level': LOG_LEVEL,
        'propagate': True,
    },
    'rdmo': {
        'handlers': ['rdmo_log'],
        'level': LOG_LEVEL,
        'propagate': True
    }
}
}

```

This produces several logs:

- `/var/log/rdmo/error.log` will contain exception messages from application errors (status code: 500). The message is the same that is shown when `DEBUG = True`, which should not be the case in a production environment. In addition to the log entry, an E-mail is sent to all admins specified in the `ADMINS` setting.
- `/var/log/rdmo/rdmo.log` will contain additional logging information from the RDMO code.
- `/var/log/rdmo/ldap.log` and `/var/log/rdmo/rules.log` can be used to debug the LDAP authentication or the internal permission system.

3.9 Plugins

Warning: This is an advanced feature of RDMO.

Plugins can be used to customize or extend specific actions in RDMO using custom Python code outside of the centrally maintained code base. This can be used to perform actions which are specific to a RDMO instance. With the possibility to add code to RDMO comes the danger of introducing additional bugs and security issues. Please be extra careful when using this **advanced** feature.

Plugins are created by implementing Python classes (e.g. in the local RDMO app), and register them in the `config/settings/local.py` file. These classes need to inherit from prepared base classes in the RDMO source code.

As of now the following plugins can be created:

- Project exports (from `rdmo.projects.export.Export`)
- Project imports (from `rdmo.projects.imports.Import`)

To be usable by RDMO, the plugins need to be available in the virtual environment in which RDMO is running. There are several possibilities to archive this, but we suggest to use one of the following:

1. If only the plugins provided in [rdmo-catalog](#) should be used and no modifications are desired, they can be directly installed from GitHub using:

```
pip install git+https://github.com/rdmorganiser/rdmo-plugins
```

2. If you intent to write plugins yourself or if you want to modify plugins, you should create a python module in your `rdmo-app`, which is just a directory containing an empty `__init__.py` file. Python files in this directory are then automatically available to your RDMO instance.

```
mkdir my_plugins
touch my_plugins/__init__.py
# place, e.g., madmp.py in my_plugins/
```

As a last step, the plugins need to be registered in `config/settings/local.py`. The setting depends on the class of plugin and is described below.

3.9.1 Project export plugins

Custom project exports can be created by implementing a class inheriting `rdmo.projects.export.Export`. They can be used to create a custom export format, to be selected by the users to download an export of their project data.

The Plugin class needs to implement a `render()` function which takes no arguments and returns a `django.http.HttpResponse`. The export can be created from `self.project`, `self.snapshot` and `self.values` instance variables.

The export plugin needs to be added to the `PROJECT_EXPORTS` in `config/settings/local.py`. The default settings are:

```
PROJECT_EXPORTS = [
    ('xml', _('as RDMO XML'), 'rdmo.projects.exports.RDMOXMLExport'),
    ('csvcomma', _('as CSV (comma separated)'), 'rdmo.projects.exports.CSVCommaExport
    ↪ '),
```

(continues on next page)

(continued from previous page)

```

    ('csvsemicolon', _('as CSV (semicolon separated)'), 'rdmo.projects.exports.
↪CSVSemicolonExport'),
]

```

In order to use the plugins in `rdmo-catalog`, add the following to your `config/settings/local.py`:

```

PROJECT_EXPORTS = [
    ('xml', _('as RDMO XML'), 'rdmo.projects.exports.RDMOXMLExport'),
    ('csvcomma', _('as CSV (comma separated)'), 'rdmo.projects.exports.CSVCommaExport
↪'),
    ('csvsemicolon', _('as CSV (semicolon separated)'), 'rdmo.projects.exports.
↪CSVSemicolonExport'),
    ('madmp', _('as maDMP JSON'), 'rdmo_plugins.exports.madmp.MaDMPExport'),
    ('datacite', _('as DataCite XML'), 'rdmo_plugins.exports.datacite.DataCiteExport
↪'),
    ('radar', _('as RADAR XML'), 'rdmo_plugins.exports.radar.RadarExport')
]

```

Please refer to <https://github.com/rdmorganiser/rdmo/blob/master/rdmo/projects/exports.py> for the default project export plugins and code examples. The code which uses the plugin is located in <https://github.com/rdmorganiser/rdmo/blob/master/rdmo/projects/views.py> (ProjectExportView).

3.9.2 Project import plugins

Similarly, custom project imports can be created implementing a class inheriting `rdmo.projects.imports.Import`. They can be used to import project data from files which are uploaded by the user.

The Plugin class needs to implement a `check()` function which takes no arguments and only returns `True` if an uploaded file can be imported by this plugin. The `self.file_name` instance variable can be used for this. In most cases, this will include opening and parsing the file.

In addition, a `process()` needs to be implemented which takes no arguments and returns `None`, but extracts the data from the file and populates the `self.project`, `self.catalog`, `self.values`, `self.snapshots`, `self.tasks` and `self.views` instance variables.

The import plugin needs to be added to the `PROJECT_IMPORTS` in `config/settings/local.py`. The default settings are:

```

PROJECT_IMPORTS = [
    ('xml', _('from RDMP XML'), 'rdmo.projects.imports.RDMOXMLExport'),
]

```

In order to use the plugins in `rdmo-catalog`, add the following to your `config/settings/local.py`:

```

PROJECT_IMPORTS = [
    ('xml', _('from RDMP XML'), 'rdmo.projects.imports.RDMOXMLExport'),
    ('madmp', _('from maDMP'), 'rdmo_plugins.imports.madmp.MaDMPExport'),
    ('datacite', _('from DataCite XML'), 'rdmo_plugins.imports.datacite.DataCiteImport
↪'),
    ('radar', _('from RADAR XML'), 'rdmo_plugins.imports.radar.RadarImport'),
]

```

Please refer to <https://github.com/rdmorganiser/rdmo/blob/master/rdmo/projects/imports.py> for the default project import plugins and code examples. The code which uses the plugin is located in <https://github.com/rdmorganiser/rdmo/blob/master/rdmo/projects/views.py> (ProjectCreateUploadView, ProjectCreateImportView, ProjectUpdateUploadView, ProjectUpdateImportView).

3.9.3 Examples of how to install

We will mention two possibilities of how to install `rdmo-plugins`. Depending on your use case you should pick the one that fits you more. The first one using `pip` does a good job if you are just planning to use the plugins from the repo without modifying these. The other method requires to copy the necessary python files into your local app folder which has advantages if you are planning to modify existing or create your own plugins.

Please note that the imports require Django's translation utils and that we usually use these with an underscore. If you are getting import errors please check if you have the first of the following lines at the beginning of your `local.py`. You also need to import `PROJECT_EXPORTS` and `PROJECT_IMPORTS` as we will later append our imports to these lists. Make sure you also have the second line in your `local.py`.

```
from django.utils.translation import ugettext_lazy as _
from rdmo.core.settings import PROJECT_EXPORTS, PROJECT_IMPORTS
```

Use pip

To install directly from github simply run

```
pip install git+https://github.com/rdmorganiser/rdmo-plugins
```

Afterwards you need to configure the plugins you are willing to use in your `local.py`. Here is an example of how to add different export formats. The three strings in brackets configure the url under which the export will be available, the name of the entry in the export menu and the path from where to import the plugin. The path resembles the structure in the `rdmo plugins` repository. If you look into it you will find the files containing the imported classes mentioned below. At the beginning of the path you use `rdmo_plugins` because you added `rdmo-plugins` to your installed python libraries. The dash is replaced by an underscore because python is not very fond of dashes when it comes to imports.

```
PROJECT_EXPORTS.append(('madmp', _('as madmp'), 'rdmo_plugins.exports.madmp.
↳MaDMPExport'))
PROJECT_EXPORTS.append(('datacite', _('as datacite'), 'rdmo_plugins.exports.datacite.
↳DataCiteExport'))
PROJECT_EXPORTS.append(('radar', _('as radar'), 'rdmo_plugins.exports.radar.
↳RadarExport'))
```

Copy files

The other method we recommend when you are planning to modify the plugins or create your own ones is to simply copy the python files into your local app folder. Let's say you create a subfolder `plugins` containing another folder `exports` in your app directory. Then you copy the `madmp.py` from the `rdmo-plugins` into `exports`. Having done this you can import this file in your local app without specifying a module path. It is found by the importer as it resides in your `rdmo app` folder. Your import lines in the `local.py` would look like this:

```
PROJECT_EXPORTS.append(('madmp', _('as madmp'), 'plugins.exports.datacite.
↳DataCiteExport'))
```

3.10 Multisite

Warning: This is an advanced feature of RDMO.

RDMO can be operated in a multi site setup, connecting several different `rdmo-apps` with different URLs and Themes, on one Server with one common Database. The different Sites will share their Catalogs, Views, etc., but the availability of this Content can be restricted among the Sites. Users can log in into any of the sites (if the authentication method allows) and projects can be shared among sites. Each of these RDMO Sites has its own `rdmo-app` directory and its own configuration (`config/settings/local.py`).

3.10.1 Setup

To setup such a multi site installation, you need to start with a regular RDMO instance as described in [installation](#). In principle, an existing RDMO instance can be extended to a multi site installation, but in this documentation we assume a fresh installation.

- Create the virtual enviroment outside the `rdmo-app`, e.g. in `/srv/rdmo/env`. In a multi site setup all RDMO sites use **the same** virtual enviroment. RDMO and the other python dependencies need to be updated only once for the whole installation.
- Otherwise, follow the instructions as usual, but add `MULTISITE = True` to `config/settings/local.py`. This enables the multi site features in the user interface.
- After the installation, login to the admin interface and add your additional sites in the Sites section as described [here](#). Note the numerical ID of the different Sites as shown in the URL when editing the Site (e.g. `http://localhost:8000/admin/sites/site/2/change/`).
- Then clone a second `rdmo-app` next to the first one, but use the same virtual environnement as before (so no `pip install` is required). Setup `rdmo-app2/config/settings/local.py` as usual. Include `MULTISITE = True` and `SITE_ID = X` in `rdmo-app2/config/settings/local.py`, where `X` is the ID of the site from the step before. For `DATABASE` use the **same** settings as in `rdmo-app`.

If you now run `./manage.py runserver 0.0.0.0:8000` in `rdmo-app` and `./manage.py runserver 0.0.0.0:8002` in `rdmo-app2`, the two sites should be working already.

3.10.2 Deployment

The two sites are deployed seperately, regardless if using Apache or nginx. Create seperate virtual host configurations which map the particular site url to the corresponding `rdmo-app`. E.g. for Apache:

```
<VirtualHost *:443>
    ServerName example.com

    ...

    Alias /static /srv/rdmo/rdmo-app/static_root/
    <Directory /srv/rdmo/rdmo-app/static_root/>
        Require all granted
    </Directory>

    WSGIDaemonProcess rdmo_app user=rdmo group=rdmo \
        home=/srv/rdmo/rdmo-app python-home=/srv/rdmo/env
    WSGIProcessGroup rdmo_app
```

(continues on next page)

(continued from previous page)

```

WSGIScriptAlias / /srv/rdmo/rdmo-app/config/wsgi.py process-group=rdmo_app
WSGIPassAuthorization On

<Directory /srv/rdmo/rdmo-app/config/>
  <Files wsgi.py>
    Require all granted
  </Files>
</Directory>
</VirtualHost>

```

and

```

<VirtualHost *:443>
  ServerName example.org

  ...

  Alias /static /srv/rdmo/rdmo-app2/static_root/
  <Directory /srv/rdmo/rdmo-app2/static_root/>
    Require all granted
  </Directory>

  WSGIDaemonProcess rdmo_app2 user=rdmo group=rdmo \
    home=/srv/rdmo/rdmo-app2 python-home=/srv/rdmo/env
  WSGIProcessGroup rdmo_app2
  WSGIScriptAlias / /srv/rdmo/rdmo-app2/config/wsgi.py process-group=rdmo_app2
  WSGIPassAuthorization On

  <Directory /srv/rdmo/rdmo-app2/config/>
    <Files wsgi.py>
      Require all granted
    </Files>
  </Directory>
</VirtualHost>

```

3.10.3 Shibboleth

In order to run multiple separate sites on one machine the Service Provider needs to be configured differently. For each site but the first one, an `ApplicationOverride` needs to be configured in `/etc/shibboleth/shibboleth2.xml`, e.g.

```

<SPConfig xmlns="urn:mace:shibboleth:3.0:native:sp:config"
  xmlns:conf="urn:mace:shibboleth:3.0:native:sp:config"
  clockSkew="180">

  <OutOfProcess tranLogFormat="%u|%s|%IDP|%i|%ac|%t|%attr|%n|%b|%E|%S|%SS|%L|%UA|%a
  ↪ " />

  <ApplicationDefaults entityID="https://sp.test.rdm.org/shibboleth"
    REMOTE_USER="eppn"
    cipherSuites="DEFAULT:!EXP:!LOW:!aNULL:!eNULL:!DES:!IDEA:!SEED:!RC4:!3DES:!
  ↪ kRSA:!SSLv2:!SSLv3:!TLSv1:!TLSv1.1">

    <Sessions lifetime="28800" timeout="3600" relayState="ss:mem"
      checkAddress="false" handlerSSL="true" cookieProps="https">

```

(continues on next page)

(continued from previous page)

```

        <SSO entityID="https://idp.test.rdm.org/idp/shibboleth"
            discoveryProtocol="SAMLDS" discoveryURL="https://ds.example.org/DS/
↳WAYF">SAML2</SSO>

        ...

    </Sessions>

    ...

    <AttributeFilter type="XML" validate="true" path="attribute-policy.xml"/>
    <CredentialResolver type="File" use="signing" key="sp-key.pem"
↳certificate="sp-cert.pem"/>
    <CredentialResolver type="File" use="encryption" key="sp-key.pem"
↳certificate="sp-cert.pem"/>

    <MetadataProvider type="XML" validate="true" path="idp-metadata.xml"/>

    <ApplicationOverride id="sp2" entityID="https://sp2.test.rdm.org/shibboleth">
        <Sessions lifetime="28800" timeout="3600" relayState="ss:mem"
            checkAddress="false" handlerSSL="true" cookieProps="https">
            <SSO entityID="https://idp2.test.rdm.org/idp/shibboleth"
                discoveryProtocol="SAMLDS" discoveryURL="https://ds.example.org/
↳DS/WAYF">SAML2</SSO>

            ...

        </Sessions>

        <CredentialResolver type="File" use="signing" key="sp2-key.pem"
↳certificate="sp2-cert.pem"/>
        <CredentialResolver type="File" use="encryption" key="sp2-key.pem"
↳certificate="sp2-cert.pem"/>
        <MetadataProvider type="XML" validate="true" path="idp2-metadata.xml"/>
    </ApplicationOverride>

</ApplicationDefaults>

...

</SPConfig>

```

where `https://idp.test.rdm.org` und `https://idp2.test.rdm.org/idp/shibboleth` are two different IdP for the two RDMO sites. Again, your Shibboleth setup might differ.

In the virtual host configuration for each but the first site, `ShibRequestSetting` `applicationId` `<id>` needs to be added to both `<Location /Shibboleth.sso>` and `<LocationMatch /(...)>`. `<id>` is the `id` attribute of the `ApplicationOverride` node, e.g.:

```

<Location /Shibboleth.sso>
    SetHandler shib
    ShibRequestSetting applicationId example.org
</Location>
<LocationMatch / (account|domain|options|projects|questions|tasks|conditions|views)>
    AuthType shibboleth
    require shibboleth
    ShibRequireSession On

```

(continues on next page)

(continued from previous page)

```
ShibUseHeaders On
ShibRequestSetting applicationId example.org
</LocationMatch>
```


CHAPTER 4

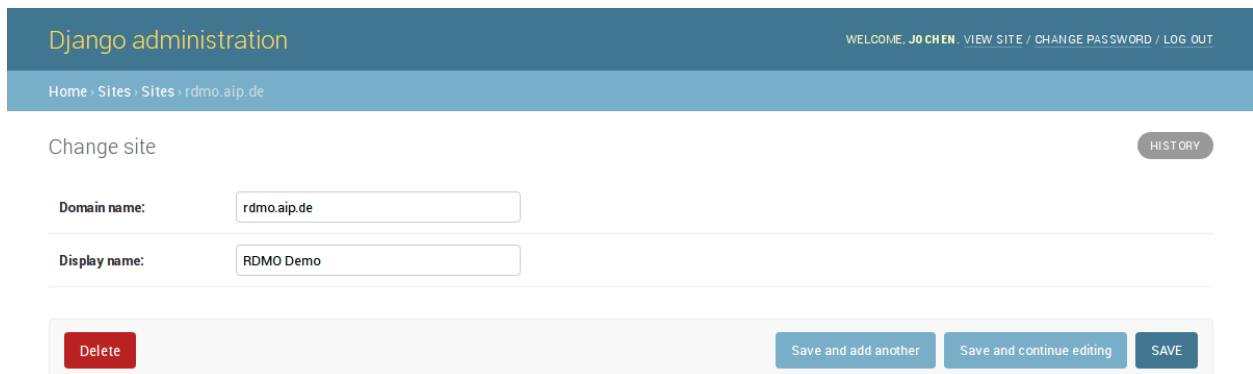
Administration

The Django framework offers a rich administration (or short admin) interface, which allows you to directly manipulate most of the entries in the database directly. Obviously, only users with the correct permissions are allowed to use this interface. The user created during the installation process using `./manage.py createsuperuser` has this *superuser* status.

The admin interface is available under the link *Admin* in the navigation bar. It will only be needed on rare occasions, since most of the configurations of the questionnaire and the other RDMO functions can be done using the more user-friendly Management interface described [in the following chapter of this documentation](#).

That being said, the admin interface is needed, especially after installation, to set the title and URL of the [site](#), to configure [users and groups](#), to configure the connection to [OAUTH providers](#), and to create tokens to be used with the [API](#).

4.1 Site configuration



The screenshot shows the Django administration interface for site configuration. At the top, there's a header with "Django administration" on the left and "WELCOME, JO CHEN | VIEW SITE / CHANGE PASSWORD / LOG OUT" on the right. Below the header is a breadcrumb trail: "Home > Sites > Sites > rdmo.aip.de". The main content area is titled "Change site" and includes a "HISTORY" button. There are two input fields: "Domain name:" with the value "rdmo.aip.de" and "Display name:" with the value "RDMO Demo". At the bottom, there are four buttons: "Delete" (red), "Save and add another" (blue), "Save and continue editing" (blue), and "SAVE" (blue).

Screenshot of the site admin interface.

RDMO used Django's [sites framework](#). It is therefore necessary to configure the **Domain name** and the **Display name** of your RDMO installation. This can be done in the admin interface under *SITE*. To configure your site:

1. Click on the already configured domain name *example.com*.
2. Enter the URL of your RDMO installation as **Domain name** (e.g. *rdmo.aip.de*).
3. Enter title of your RDMO installation as **Display name** (e.g. *RDMO Demo*).
4. Click **Save** to save the site.

4.2 Users and Groups

The users and groups of your RDMO instance can be managed under **AUTHENTICATION AND AUTHORIZATION**. You can create and update users and set their password directly, but most of the time this will be done by the users themselves using the account menu.

The user created in the installation process can access all features of RDMO. In order to allow other users to access the management or the admin interface, they need to have the required permissions assigned to them. This can be done in two ways: through groups or using the superuser flag.

4.2.1 Groups

During the installation, the `./manage setup_groups` command created 3 groups:

Editor

Users of the group editor can access the [management interface](#) and can edit all elements of the data model, except the user data entered through the structured interview.

Reviewer

Users of the group reviewer can access the [management interface](#), like editors, but are not allowed to change them (Save will not work). This group can be used to demonstrate the management backend of RDMO to certain users.

API

Users of the group api can use the programmable API to access all elements of the data model. They will need a [token](#) to use an api client.

Existing users can be assigned to these groups to gain access to these functions:

1. Click **Users** under **AUTHENTICATION AND AUTHORIZATION** in the admin interface.
2. Click on the user to be changed.
3. Click on the group to be assigned to the user in the **Available groups** field.
4. Click on the little arrow to move the group to the **Chosen groups** field.
5. Save the user.

4.2.2 Roles

Users can be set as `site_managers` for a specific (or the only) site under **ACCOUNTS** and **Roles**. Site managers have access to all projects of this site.

4.2.3 Superuser

Superusers have all permissions available and all permission checks will return positive to them. This does not only allow them to access the management and admin interfaces, but also **access all data from other user** (including the project pages).

To make a user superuser:

1. Click **Users** under **AUTHENTICATION AND AUTHORIZATION** in the admin interface.
2. Click on the user to be changed.
3. Tick the box **Superuser status**.
4. Save the user.

4.3 Social accounts

If you use allauth as your mode of authentication and configured RDMO to use one or more OAUTH provider (as described in the [Configuration chapter](#), you need to register your RDMO site with theses services. This process is different from provider to provider. Usually, you need to provide a set of information about your site. Always included is a redirect or callback url. In the following we will use `http://127.0.0.1:8000` as an example (which will work on the development server) and you will need to replace that with the correct url of your RDMO application in production.

4.3.1 ORCID

Login into <https://orcid.org> and go to the developer tools page at <https://orcid.org/developer-tools>. Create an app with the Redirect URI

```
http://127.0.0.1:8000/account/orcid/login/callback/
```

4.3.2 Github

Login into github and go to <https://github.com/settings/applications/new> and create a new app. Use

```
http://127.0.0.1:8000/account/github/login/callback/
```

4.3.3 Facebook

Login into facebook and go to <https://developers.facebook.com/>. Click on the top right menu *My Apps* and choose *Add a new app*. Create a new app. In the following screen choose Facebook login -> Getting started and choose *Web* as the platform. Put in a URL under which your application is accessible (Note: 127.0.0.1 will not work here.). Back on the dashboard, go to Settings -> Basic and copy the App ID and the App Secret.

4.3.4 Twitter

Login into twitter and go to <https://apps.twitter.com/app/new> and create a new app. Use

```
http://127.0.0.1:8000/account/facebook/login/callback/
```

as the Authorized redirect URI. Copy the Client-ID and the Client key.

4.3.5 Google

Login into google and go to <https://console.developers.google.com>. Create a new project. After the project is created go to Credentials on the left side and configure the OAuth Authorization screen (second tab). Then create the credentials (first tab), more precisely a OAuth Client-ID. Use

```
http://127.0.0.1:8000/account/google/login/callback/
```

as the Authorized redirect URI. Copy the Client-ID and the Client key.

Once the credentials are obtained, you need to enter them in the admin interface. To this purpose, go to **Social applications** under **SOCIAL ACCOUNTS** and click on **Add social application**. Then:

1. Select the corresponding **provider**
2. Enter a **Name** of your choice
3. Enter the **Client id** (or App ID) and the **Secret key** (or Client secret, Client key, App Secret)
4. Add your site to the chosen sites.
5. Click save.

4.4 API

RDMO has an API which can be used to retrieve data in a machine readable manner. The API can be accessed with any http speaking tool that is able to provide the necessary authentication token in the request's header. In the following examples we will use `curl`.

4.4.1 Authentication

In order to access any data entered into RDMO through the programmable API, a user needs to have a token associated with him. This is done under **AUTH TOKEN / Tokens**. To create a token, click **Add token** on the button at the right and:

1. Select the **user** for the new token.
2. Save the token.

The created token can be used instead of the username and the password when making HTTP requests from a non-browser client. The token needs to be provided in the HTTP header in following form.

```
Authorization: Token 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b
```

Only `Superusers` are allowed to access the API. You can check if your user has the required permissions by having a look into the user's details. If the user does not have the Superuser status any API request will fail producing status code 403.

4.4.2 API Layout / Swagger

Concerning questions about the API Layout `Swagger` comes in as a handy tool because it can be used to get a detailed description of the API.

What is Swagger?

Swagger is a set of tools built around the OpenAPI Specification. These tools help to design, build and document REST APIs. OpenAPI is already implemented in RDMO. The Swagger page in RDMO is provided by a python library called [Django REST Swagger](#).

The interactive Swagger page can help you to design API queries because it gives a well-arranged interactive overview of all the available endpoints and query parameters. If you need help to get an idea about the possibilities of the RDMO API you should have a look.

Use Swagger Tools

Swagger Tools are enabled by default. The relevant entry is in your app's `urls.py` and looks like the following.

```
path('api/v1/', include('rdmo.core.urls.swagger')),
```

Remove it or comment it out if you do not want swagger to be available. As you can see the default swagger page's url is `http://$YOUR_RDMO/api/v1/`.

If you request `http://$YOUR_RDMO/api/v1/?format=openapi` you get a detailed machine processable API description in JSON format.

API Structure

```
/api/v1/conditions/conditions
/api/v1/conditions/conditions/{id}

/api/v1/core

/api/v1/domain/attributes
/api/v1/domain/attributes/{id}
/api/v1/domain/entities
/api/v1/domain/entities/{id}

/api/v1/options/options
/api/v1/options/optionsets
/api/v1/options/optionsets/{id}
/api/v1/options/options/{id}

/api/v1/projects/projects
/api/v1/projects/projects/{id}
/api/v1/projects/snapshots
/api/v1/projects/snapshots/{id}
/api/v1/projects/values
/api/v1/projects/values/{id}

/api/v1/questions/catalogs
/api/v1/questions/catalogs/{id}
/api/v1/questions/questions
```

(continues on next page)

(continued from previous page)

```

/api/v1/questions/questionsets
/api/v1/questions/questionsets/{id}
/api/v1/questions/questions/{id}
/api/v1/questions/sections
/api/v1/questions/sections/{id}

/api/v1/tasks/tasks
/api/v1/tasks/tasks/{id}

/api/v1/views/views
/api/v1/views/views/{id}

```

The API structure is roughly determined by the element types of which data can be fetched. The `{id}` placeholder at the end of the URLs is indicating that an element ID can be provided to fetch data of a distinct single element. So basically we have to types of requests. The first returning lists of elements and the second returning single elements. Many of the endpoints provide multiple filter functions that are accessible using request parameters. If you for example would like to retrieve information about the admin user you could use `?username=admin` as filter parameter at the end of the url.

For practical reasons all the different filters will not be named here. But you can have a look into the [JSON description](#) of the API generated using Swagger. It covers every piece of information regarding all the endpoints and their available filters. As JSON can be a hard read sometimes you could also copy and paste the content of file into the [Swagger Editor](#) to get a more consumable version of the description. If you would like to use Swagger to interact with your RDMO's API in a web browser you could set it up for your RDMO installation.

4.4.3 Curl Request Examples

Note that the examples below use the `-L` parameter. It tells curl to follow redirects and is necessary because the token authentication process involves a redirect. If you do not follow it you will get an empty-bodied response having status code 301.

You could for example request...

1. The project having the id 1

```
curl -LH "Authorization: Token $YOUR_TOKEN" \
  "https://$YOUR_RDMO/api/v1/project/projects/1"
```

1. A list of all projects

```
curl -LH "Authorization: Token $YOUR_TOKEN" \
  "https://$YOUR_RDMO/api/v1/project/projects"
```

1. A list of users in a certain project

```
curl -LH "Authorization: Token $YOUR_TOKEN" \
  "http://$YOUR_RDMO/api/v1/accounts/users/?project=1"
```

1. A list of options belonging to optionset 1

```
curl -LH "Authorization: Token $YOUR_TOKEN" \
  "http://$YOUR_RDMO/api/v1/options/options/?optionset=1"
```

1. A list of options having uri `https://rdmorganiser.github.io/terms/options/research_fields/216` Note that strings of course need to be url encoded.

```
curl -LH "Authorization: Token $YOUR_TOKEN" \  
      "http://localhost/api/v1/options/options/?uri=https%3A%2F%2Frdmorganiser.  
↳github.io%2Fterms%2Foptions%2Fresearch_fields%2F216"
```

4.4.4 RDMO Client

There is an RDMO Client written in Python which may help you to get started using the API. It is available on [GitHub](#).

A freshly installed instance of RDMO is not very useful without a questionnaire to fill out by the user and a set of DMP templates later to be populated by the given answers. The main idea of RDMO is that every question and every output can be customized by you. This, however, introduces a certain level of complexity. RDMO employs a datamodel organized along different Django apps and models (representing database tables). A graphical overview is given in the figure below:

Overview of the RDMO data model

Here, we explain the different parts of the data model. Each section has a link to a more detailed explanation how to create and edit the relevant elements.

For most users, the structured interview will be the most visible part of RDMO. It is configured using **catalogs**, **sections**, **questionsets**, and **questions**. A single installation of RDMO can have several catalogs. When creating a new project, a user can select one of these catalogs to be used with this project. A catalog has a number of sections, which themselves have question sets. Questions can be directly added to question sets. A question has a text, which will be shown in bold to the user and an optional help text. It also has a widget type, which determines which interface widget is presented to the user (e.g. text field, select field, radio buttons). The questionnaire is configured under / `questions` available in the management menu. More documentation about the questions management can be found [here](#).

The **domain model** is the central part of the data model and connects the questions from the questionnaire with the user input. It is organized as a tree-like structure. Every piece of information about a user's project is represented by an **attribute**. In this sense these attributes can be compared to a variable in source code. Attributes are the leaves of the domain model tree, that organize the connections between the different entities assigned to them. Much like files are organized along directories on a disk. Every question or questionset set must have an attribute to be connected to. An example would be the attribute with the path `project/schedule/project_start` for the start date of the project. The attribute itself has the key `project_start` and resides in the attribute `schedule`, which itself is located in the `project`.

Conditions can be connected to questionsets. Conditions control if the questionsets is valid in the current context. If questionset is not valid, it will not be shown to the user. Conditions are also needed to disable/enable option sets, tasks and can be used in views. Conditions are configured with a source attribute, which will be evaluated, a relation like "equal" or "greater than", and a target. The target is a text string or an option. As an example, if the

source is the attribute `project/legal_aspects/ipr/yesno`, the relation is “equal to”, and the target text is “1”. The condition will be true for a project where the answer to the question connected to the attribute `project/legal_aspects/ipr/yesno` is “1” (or “yes” for a yesno widget). Conditions configured under `/conditions` are available in the management menu. More documentation about the conditions management can be found [here](#).

Views allow for custom DMP templates in RDMO. To this purpose every view has a template which can be edited using the Django template syntax, which is based on HTML. Views have also a title and a help text to be shown in the project overview. Views are configured under `/views` available in the management menu. More documentation about editing views can be found [here](#).

After filling out the interview, the user will be presented with follow up **tasks** based on his/her answers. A task has a title and a text. **Time frames** can be added to tasks, which themselves are evaluating attributes of the value type “datetime”, to use answers such as the beginning or the end of a project to compute meaningful tasks. Most of the time tasks will have a condition connected to them, to determine if this task is needed for a particular project or not. Tasks configured under `/tasks` are available in the management menu. More documentation about editing views can be found [here](#).

The different elements of the RDMO datamodel have various parameters, which control their behavior in RDMO and can be configured using the different management pages, which are described on the following pages. In addition, all elements contain a set of common parameters:

* URI prefix

The URI prefix is the first part of the URI. As every element has a URI, every element does obviously also have a URI prefix. Semantically the prefix is only relevant when different RDMO instances share data between each other. In this case the URI prefix is used to determine which instance the data belong to. You may think of it as a kind of unique instance identifier.

When you import a question catalog or any other content from another institution these imports do have a URI prefix different from yours. If you change elements from these third party imports we strongly recommend to always adjust the URI prefix into your own one to make the changes persistent. This is necessary because a re-import of the third party content will overwrite by using the URI as identifier. Data in your database having the same URI as the imported ones will get updated and so overwritten. Please do also look into [Export and Import](#) page for a little more detail.

By convention the URI prefix has to look like a URL. It does not have to be a valid URL in terms of being resolvable. In principle you could use any kind of string as long as it fits the scheme but we recommend to use the URL of your RDMO instance. The URI prefix has to start with `http://` or `https://`. Afterwards there has to be a host name. Anything further like for instance a path is optional. Valid URI prefixes for example are: `https://rdmorganiser.github.io/terms` or `https://rdmo.aip.de`.

In edit forms of elements you will find a button looking like this . It can be used to automatically put the default URI prefix into the open form. This is very useful especially if you do not exactly remember or know the default value of your RDMO installation. The default URI prefix that this button gets is defined in the `local.py`. The button is only helpful if the value is set. We strongly recommend to add an entry like the following to your config of course having your URI prefix as value.

```
DEFAULT_URI_PREFIX = 'https://rdmo.uni-xyz.de/terms/'
```

* Key

A key that is used as an internal identifier and determines, together with the URI prefix, the URI of the element.

* Internal Comment

An internal comment to share information to be seen by users with access to the management backend.

5.1 Questions

The questions management is available under *Questions* in the management menu in the navigation bar. This page will be empty after a fresh RDMO installation. We suggest to **first import our domain model** and, if you like, our general questionnaire. The corresponding XML-files are available at <https://github.com/rdmorganiser/rdmo-catalog>.

If there is at least one questionnaire imported, it will be shown automatically. Other catalogs can be selected in the sidebar on the right-hand side afterwards.

The screenshot displays the RDMO Questions management interface. At the top, a navigation bar includes 'RDMO Demo', 'Management', 'Admin', and a 'Language' dropdown. The main content area is titled 'Questions' and 'RDMO'. It shows a hierarchical tree of sections, subsections, question sets, and questions for the 'rdmo' catalog. The right sidebar contains options for catalog management, export, and import.

Questions

RDMO

Section General `rdmo/general` + ✎ 🗑️

Subsection Topic `rdmo/general/topic` + ✎ 🗑️

Question set `rdmo/general/topic/research_question` + ✎ 🗑️

Question What is the main research question of the project?
`rdmo/general/topic/research_question/title` → `project/research_question/title` ✎ 🗑️

Question Please give some keywords describing the research question.
`rdmo/general/topic/research_question/keywords` → `project/research_question/keywords` ✎ 🗑️

Question set `rdmo/general/topic/research_field` + ✎ 🗑️

Question Which research field(s) does this project belong to?
`rdmo/general/topic/research_field/research_field` → `project/research_field/title` ✎ 🗑️

Subsection Project schedule `rdmo/general/project-schedule` + ✎ 🗑️

Question set `rdmo/general/project-schedule/schedule` + ✎ 🗑️

Question When does the project start?
`rdmo/general/project-schedule/schedule/project_start` → `project/schedule/project_start` ✎ 🗑️

Question When does the project end?
`rdmo/general/project-schedule/schedule/project_end` → `project/schedule/project_end` ✎ 🗑️

Subsection Project partners `rdmo/general/project-partners` + ✎ 🗑️

Question set `rdmo/general/project-partners/name` + ✎ 🗑️

Question Which persons or institutions are responsible for the project coordination?
`rdmo/general/project-partners/name/name` → `project/coordination/name` ✎ 🗑️

Question set `rdmo/general/project-partners/partner` → `project/partner` + ✎ 🗑️

Catalog

RDMO

Filter

✕

Options

Update catalog details
Delete catalog
Create new catalog
Create new section
Create new subsection
Create new questionset
Create new question

Export


PDF
Rich Text Format
Open Office
Microsoft Office
HTML
Markdown
mediawiki
LaTeX
XML




Import

Select xml file Upload

Screenshot of the questions management interface

On the left-hand side is the main display of sections, questionsets, and questions for the current catalog. For sections and questionsets the title and the key is shown. For questions and question set the key and the key of the attribute they are connected to is shown. The order of the different elements is the same as in the structured interview shown to the user. On the right side of each elements panel, icons indicate ways to interact the element. The following options are available:

- Add () a new section, a new questionset, or a new question.

- **Update** () an element to change its properties.
- **Copy** () a question or questionset. This will open the same modal as update. You can change some of the properties and save the element as a new one. This can save time when creating several similar questions.
- **Delete** () an element and all of its decedents (e.g. a question set and all the questions it contains). **This action cannot be undone!**

The sidebar on the right-hand side shows additional interface items:

- **Catalog** switches the view to a different catalog.
- **Filter** filters the view according to a user given string. Only elements containing this string in their `path` will be shown.
- **Options** offers additional operations:
 - Update the details of the current catalog
 - Delete the current catalog
 - Create a new (empty) catalog
 - Create a new (empty) section
 - Create a new (empty) subsection
 - Create a new (empty) question set
 - Create a new (empty) question
- **Export** exports the current catalog to one of the displayed formats. While the text based formats are mainly for showing the full catalog, the XML export can be used to transfer this catalog to a different installation of RDMO.

The different elements of the questionnaire have different properties to control their behavior. As described in [the introduction](#), all elements have an URI prefix, a key, and an internal comment only to be seen by other managers of the RDMO installation. In addition, you can edit the parameters below:

5.1.1 Parameters

Catalog

Section

Question sets

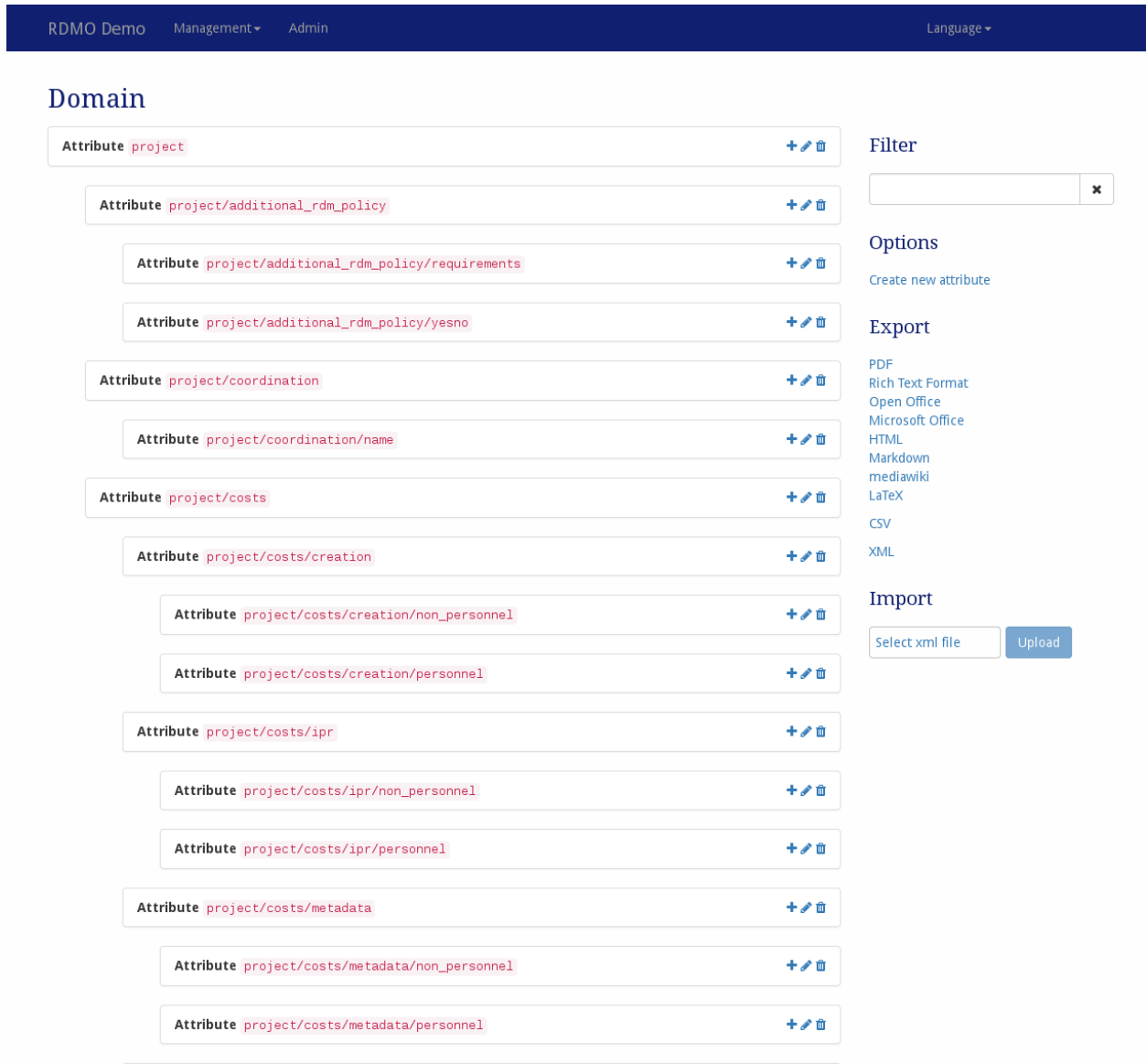
Questions

5.2 Domain

The domain model can be managed under *Domain* in the management menu in the navigation bar.




After the installation of RDMO the domain is initially empty. **We suggest that all RDMO operators initially import the domain model provided by the RDMO project.** Using a common domain over all RDMO instances allows us to exchange questionnaires, views and other content, and could lead to a common metadata application profile for data management planning. Our domain is available at <https://github.com/rdmorganiser/rdmo-catalog>. The domain is

meant to be extendable, but the core data model enables interoperability and cooperativity amongst RDMO instances and it is a good starting point to create questionnaires.



Screenshot of the domain management interface

On the left-hand side is the main display of all the attributes available in this installation of RDMO. On the right side of each elements panel, icons indicate ways to interact the element. The following options are available:

- **Add** () a new attribute.
- **Update** () an attribute to change its properties.
- **Delete** () an attribute and all of it's decendents. **The action will remove the attribute and all the attributes below. This action cannot be undone!**

The sidebar on the right-hand side shows additional interface items:

- **Filter** filters the view according to a user given string. Only elements containing this string in their path will be

shown.

- **Options** offers additional operations:
 - Create a new (empty) attribute
- **Export** exports the current catalog to one of the displayed formats. While the textual formats are mainly for presentation purposes, the XML export can be used to transfer the domain model to a different installation of RDMO.

The different elements of the domain model have different properties to control their behavior. As described in [the introduction](#), all elements have an URI prefix, a key, and an internal comment only to be seen by other managers of the RDMO installation. In addition, you can edit the parameters below:

5.2.1 Parameters

5.3 Options

Options and option sets can be managed under *Options* in the management menu in the navigation bar.

RDMO Demo
Management
Admin
Language

Options

Option set	control_tools	+ / ? /
Option	control_tools/1	Calibration procedure
Option	control_tools/2	Repeated measurements
Option	control_tools/3	Standards for data recording
Option	control_tools/4	Controlled vocabularies or standard terminology
Option	control_tools/5	Validation of the data collection
Option	control_tools/6	Peer reviews of the data
Option	control_tools/7	Others

Option set	data_protection_laws	+ / ? /
Option	data_protection_laws/21	Bundesdatenschutzgesetz (BDSG, Federal Data Protection Act)
Option	data_protection_laws/22	Landesdatenschutzgesetz Baden-Württemberg (State Data Protection Act of Baden-Württemberg)
Option	data_protection_laws/23	Landesdatenschutzgesetz Bayern (State Data Protection Act of Bavaria)
Option	data_protection_laws/24	Landesdatenschutzgesetz Berlin (State Data Protection Act of Berlin)
Option	data_protection_laws/25	Landesdatenschutzgesetz Bremen (State Data Protection Act of Bremen)
Option	data_protection_laws/26	Landesdatenschutzgesetz Brandenburg (State Data Protection Act of Brandenburg)
Option	data_protection_laws/27	Landesdatenschutzgesetz Hamburg (State Data Protection Act of Hamburg)
Option	data_protection_laws/28	Landesdatenschutzgesetz Mecklenburg-Vorpommern (State Data Protection Act of Mecklenburg-Vorpommern)
Option	data_protection_laws/29	Landesdatenschutzgesetz Hessen (State Data Protection Act of Hesse)
Option	data_protection_laws/30	Landesdatenschutzgesetz Nordrhein-Westfalen (State Data Protection Act of North Rhine-Westphalia)
Option	data_protection_laws/31	Landesdatenschutzgesetz Rheinland-Pfalz (State Data Protection Act of Rhineland-Palatinate)
Option	data_protection_laws/32	Landesdatenschutzgesetz Niedersachsen (State Data Protection Act of Lower Saxony)

Filter

Options

Create new option set

Create new option

Export

PDF

Rich Text Format

Open Office

Microsoft Office

HTML

Markdown

mediawiki

LaTeX

XML

Import

Select xml file

Upload

Screenshot of the options management interface

On the left-hand side is the main display of all the option sets and options available in this installation of RDMO. Option sets show their key, while options show their path and their text. On the right side of each elements panel, icons indicate ways to interact the element. The following options are available:

- **Add** (+) a new option to an option set.
- **Update** () an option set or option to change its properties.
- **Update conditions** (?) of an option set. A question connected to an option set with one or more conditions, will not show the options of the set in the questionnaire, if the condition is evaluated to be false. The conditions themselves are configured in [the conditions management](#).
- **Delete** () an option set or option and, in the case of an option set, all of its options. **This action cannot be undone!**

The sidebar on the right shows additional interface items:

- **Filter** filters the view according to a user given string. Only elements containing this string in their path will be shown.
- **Options** offers additional operations:
 - Create a new (empty) option set
 - Create a new (empty) option
- **Export** exports the options sets to one of the displayed formats. While the textual formats are mainly for presentation purposes, the XML export can be used to transfer the options sets to a different installation of RDMO.

Option sets and options model have different properties to control their behavior. As described in [the introduction](#), all elements have an URI prefix, a key, and an internal comment only to be seen by other managers of the RDMO installation. In addition, you can edit the parameters below:

5.3.1 Parameters

Option set

Option

5.4 Conditions

Conditions can be created and managed under *Conditions* in the management menu in the navigation bar. They are later connected to Question sets, Option sets, or Tasks.

RDMO Demo
Management
Admin
Language

Conditions

Condition `additional_rdm_policy`

`project/additional_rdm_policy/yesno` is equal to (==) `other_requirements_options/146` ("Yes")

Condition `data_sharing`

`project/dataset/sharing/yesno` is equal to (==) `dataset_sharing_options/69` ("Yes, externally for everyone")

Condition `infrastructure_resources`

`project/dataset/usage_infrastructure` is equal to (==) `infrastructure_resources/261` ("The following infrastructure resources are needed")

Condition `intellectual_property_rights`

`project/legal_aspects/ipr/yesno` is equal to (==) "1"

Condition `international`

`project/legal_aspects/international_yesno` is equal to (==) "1"

Condition `law_international`

`project/legal_aspects/international_yesno` is equal to (==) "1"

Condition `other_sensitive_data`

`project/dataset/sensitive_data/other/yesno` is equal to (==) "1"

Condition `personal_data`

`project/dataset/sensitive_data/personal_data_yesno/yesno` is equal to (==) "1"

Condition `personal_inf_bdsg_3_9`

`project/dataset/sensitive_data/personal_data/bdsg_3_9` is equal to (==) "1"

Condition `rights_owner_clear`

Filter

Options

Create new condition

Export

PDF
Rich Text Format
Open Office
Microsoft Office
HTML
Markdown
mediawiki
LaTeX
XML



Import

Select xml file

Upload

Screenshot of the conditions management interface.

On the left-hand side is the main display of all the conditions available in this installation of RDMO. Conditions show their key and a textual representation of what they evaluate. On the right side of each conditions panel, icons indicate ways to interact the element. The following options are available:

- **Update** () a condition to change its properties.
- **Delete** () a condition. **This action cannot be undone!**

The sidebar on the right shows additional interface items:

- **Filter** filters the view according to a user given string. Only conditions containing this string in their path will be shown.
- **Options** offers additional operations:
 - Create a new condition

- **Export** exports the conditions to one of the displayed formats. While the textual formats are mainly for presentation purposes, the XML export can be used to transfer the conditions to a different installation of RDMO.

Conditions have different properties to control their behavior. As described in [the introduction](#), all elements have an URI prefix, a key, and an internal comment only to be seen by other managers of the RDMO installation. In addition, you can edit the parameters below.

5.4.1 Configuration

Conditions are configured with a source attribute, which will be evaluated, a relation like “equal” or “greater than” and a target. The target is a text string or an option. As an example, if the source is the attribute `project/legal_aspects/ipr/yesno`, the relation is “equal to”, and the target text is “1”, the condition will be true for a project where the answer to the question connected to the attribute `project/legal_aspects/ipr/yesno` is “1” (or “yes” for a yesno widget).

5.4.2 Parameters

5.5 Views

1. *Parameters*

1. *View*

2. *View Templates*

1. *Calculations*

Views can be configured under *Views* in the management menu in the navigation bar.

RDMO Demo
Management
Admin
Language

Views

View <code>bielefeld</code> Bielefeld <div> <div></div> <div></div> <div></div> </div>	DMP template from the University of Bielefeld.
View <code>citec</code> CITEC DMP <div> <div></div> <div></div> <div></div> </div>	DMP template from the University of Bielefeld for CITEC funded research projects.
View <code>dmponline</code> DMPonline template <div> <div></div> <div></div> <div></div> </div>	Template from DMPonline, online https://dmponline.dcc.ac.uk
View <code>dmptool</code> DMPTool template <div> <div></div> <div></div> <div></div> </div>	Template from DMPTool, based on "NSF-GEN: Generic", online: https://dmptool.org
View <code>horizon2020</code> Horizon 2020 FAIR Data Management Plan template <div> <div></div> <div></div> <div></div> </div>	Template for Horizon 2020, from "Guidelines on FAIR Data Management in Horizon 2020", online: http://ec.europa.eu/research/participants/data/ref/h2020/grants_manual/hi/oa_pilot/h2020-hi-oa-data-mgt_en.pdf
View <code>snf</code> SNF Template <div> <div></div> <div></div> <div></div> </div>	DMP of the SNF (Switzerland)

Filter

Options

Create new view

Export

PDF
Rich Text Format
Open Office
Microsoft Office
HTML
Markdown
mediawiki
LaTeX
XML




Import

Select xml file

Upload

Screenshot of the views management interface

On the left-hand side is the main display of all the views available in this installation of RDMO. Views show their key, title and description. On the right side of each views panel, icons indicate ways to interact the element. The following options are available:

- **Update** () a view to change its properties.
- **Edit the template** () of a view.
- **Delete** () a view. **This action cannot be undone!**

The sidebar on the right shows additional interface items:

- **Filter** filters the view according to a user given string. Only views containing this string in their path will be shown.
- **Options** offers additional operations:
 - Create a new view
- **Export** exports the conditions to one of the displayed formats. While the textual formats are mainly for presentation purposes, the XML export can be used to transfer the views to a different installation of RDMO.

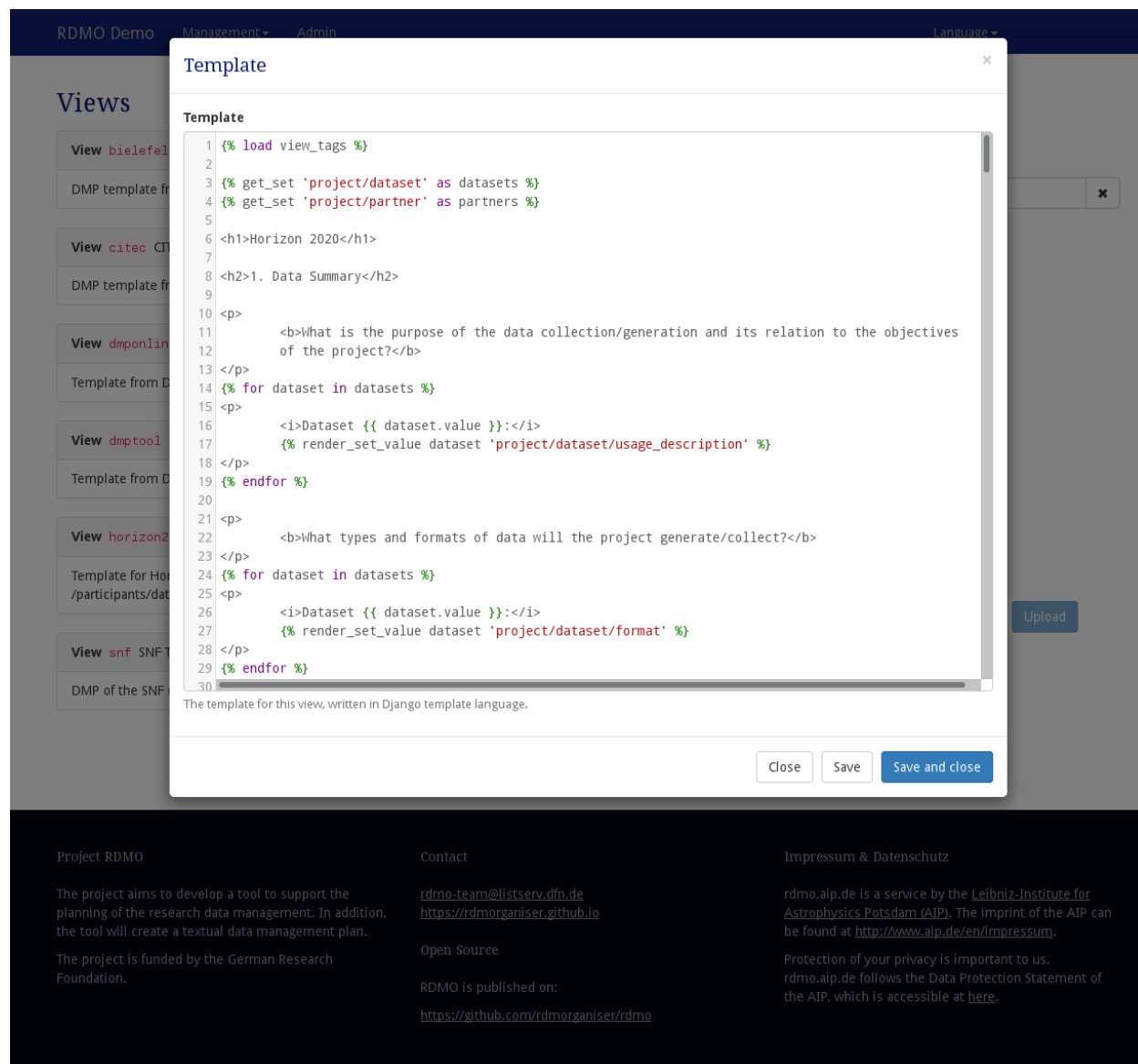
Views have different properties to control their behavior. As described in [the introduction](#), all elements have an URI prefix, a key, and an internal comment only to be seen by other managers of the RDMO installation. In addition, you can edit the parameters below:

5.5.1 Parameters

View

Tab English | -|-| |Title|The English title for the view. The title will be shown in the projects overview.| |Help|The English help text for the view. The help text will be shown in the projects overview| **Tab German**|*contains the same elements as the English one but obviously for German language content*| **Tab Groups**| |Groups|Displays the groups for this view. If at least one group is selected, only users of these groups will see this view for a project.| **Tab Sites**| |Sites| (*Only in a multi site installation*) Displays the sites for this view. Only users of these groups will see this view for a project.|

5.5.2 View Templates



Screenshot of the template modal

Each view has a template, which determines how the answers given by the user are mapped to a textual document. The template is composed using the [Django template](#) syntax, which is a combination of regular HTML, variables, which get replaced with values when the template is evaluated (`{{ a_variable }}`), and tags, which control the logic of the template (`{% a_tag %}`).

In the first line of the view template you will find the load command for the available view tags. It makes the logic of the template available and should always be there.

```
{% load view_tags %}
```

Immediately afterwards there will probably be variable declarations which load sets into place holders and make them available throughout the whole template. These variables can for example be used in for loops as you will see later.


```
{% get_set 'project/partner' as partners %}
{% get_set 'project/dataset' as datasets %}
```

Consider an attribute `project/research_question/title` and a user, who answered the question connected to this attribute with “To boldly go where no man has gone before.”. The attribute would be available in the template as `project/research_question/title`.

```
The main research question of the project is: {% render_value 'project/research_
↪question/title' %}
```

would, when evaluated in the context by a user in his/her project, render:

```
The main research question of the project is: To boldly go where no man has gone_
↪before.
```

Lists of multiple values can also be rendered.

```
<p>
    {% render_value_inline_list 'project/research_question/keywords' %}
</p>
```

As equivalent for the snippet above you can also use the following which gives you more control over the list layout.

```
<ul>
{% get_values 'project/research_question/keywords' set_index=0 as text %}
    {% for value in text %}
        <li>{{ value.value }}</li>
    {% endfor %}
</ul>
```

For set entities, you can use the initially declared variables. Your code would look like this.

```
{% for dataset in datasets %}
    <p>
        {% render_set_value dataset 'project/dataset/id' %}
    </p>
{% endfor %}
```

Values can be used if they meet certain conditions. If you want to display something based on a certain value being true you can for example do this. Note that there is an `.is_false` function as well which can be used just as the mentioned counterpart.

```
{% get_value 'project/dataset/sharing/yesno' as val %}
{% if val.is_true %}
    This will be only rendered if personal_data resolves to be true.
{% endif %}
```

Or checking a value within a dataset.

```
{% for dataset in datasets %}
    {% get_set_value dataset 'project/dataset/id' as val %}
    {% if val.is_true %}
        {% render_set_value dataset 'project/dataset/id' %}
    {% endif %}
{% endfor %}
```

Calculations

You can do calculations in RDMO's view templates by using filters. The package RDMO utilizes is called `django-mathfilters`. The following operations are supported. For more information please have a look into the `django-mathfilters` documentation which can be found at the link mentioned before.

The following examples illustrate how to use the mathfilters. It is quite easy if you pay attention to two things. First step is to load the desired value into a variable. This can be achieved as usual by `get_value`. Afterwards this value can be used in calculations but it explicitly needs to be used as `as_number`.

For example you could do the following to get the sum of two values.

```
{% get_value 'project/costs/creation/personnel' as val1 %}
{% get_value 'project/costs/creation/non_personnel' as val2 %}

{{ val1.as_number | addition:val2.as_number }}
```

Note that filters can be piped after another as often as you like. You could easily do something like this. But pay attention to having the piped filters in a single line because Django templates do not support having filters spreading across multiple lines.

```
{% get_value 'project/costs/storage/personnel' as val1 %}
{% get_value 'project/costs/storage/non_personnel' as val2 %}
{% get_value 'project/costs/metadata/personnel' as val3 %}
{% get_value 'project/costs/metadata/non_personnel' as val4 %}

{{ val1.as_number | addition:val2.as_number | addition:val3.as_number | addition:val4.
↪as_number }}
```

Please consult the documentation of the Django template syntax for all the available tags and filters: <https://docs.djangoproject.com/en/stable/ref/templates/language>.

5.6 Tasks

Tasks can be configured under *Tasks* in the management menu in the navigation bar.

RDMO Demo
Management
Admin
Language

Tasks

Task `clarify_consequences_international_law` Clarify consequences of different international jurisdictions

Please get in touch with the legal department or a respective contact person at your institution to clarify if this has consequences for the project and its data management and if yes, what consequences these are.

Task `contact_data_security_officer` Contact data security officer about special requirements

At least one of the datasets contains particularly sensitive personal information according to BDSG § 3, 9. Please get in touch with the data protection officer of your institution to check which additional protection measures are necessary.

Task `contact_IT_department` Contact IT department to acquire infrastructure resources

Please get in touch with your IT department to arrange that the required infrastructure resources are provided.

Task `contact_IT_or_data_management_expert` Contact IT or data management expert

Please contact an IT or data management expert to get support in terms of data usage.

Task `contact_rights_owner` Contact rights owner

Clarify, if and how this restricts the possibility of long-term preservation and re-use and if the rights owner is willing to grant the necessary rights.

Task `create_naming_policy` Create data naming policy

Create a guideline for a consistent naming of the data within the project.

Task `create_data_organisation_policy` Create data organisation policy

Create a policy for a consistent data organisation in the project.

Task `determine_versioning_tool` Determine Versioning Tool

Determine, which versioning tool will be used for the versioning.

Task `finish-first-dmp` Finish first DMP

The first DMP should be finished about 3 months after the project has started.

Filter





Options

Export

Import

Screenshot of the tasks management interface

On the left-hand side is the main display of all the tasks available in this installation of RDMO. Tasks show their key, title and the text describing the task. On the right side of each task's panel, icons indicate ways to interact with the element. The following options are available:

- **Update** () a task to change its properties.
- **Update conditions** () of a task. A task will only be shown to the user, if all of its conditions are evaluated `True`. The conditions themselves are configured in [the conditions management](#).
- **Update the time frame** () of a task. The time frame is constructed from one or two dates from the user's answers. This allows for tasks about a specific deadline or special period.
- **Delete** () a task. **This action cannot be undone!**

The sidebar on the right-hand side shows additional interface items:

- **Filter** filters the view according to a user given string. Only tasks containing this string in their path will be shown.
- **Options** offers additional operations:
 - Create a new task
- **Export** exports the conditions to one of the displayed formats. While the textual formats are mainly for presentation purposes, the XML export can be used to transfer the tasks to a different installation of RDMO.

Tasks have different properties to control their behavior. As described in [the introduction](#), all elements have an URI prefix, a key, and an internal comment only to be seen by other managers of the RDMO installation. In addition, you can edit the parameters below:

5.6.1 Parameters

Task

Time frame

5.7 Export and Import

RDMO offers export and import functions which are available in the right handed sidebar. The two categories offering the equally named functions do appear for every datatype (e.g. Questions, Domain, etc.). Note that these functions are context sensitive. If you for example use the `Export` or `Import` being in `Tasks` you can only ex- or import `Tasks`. This is especially important to know for the import, because it will fail and display an error if you pick an xml file that does not contain fitting content.

5.7.1 Export

The export menu is displayed on the right-hand side of the screen. It consists of a list of the supported export formats which may vary depending on what page you are on. For information about how to configure the available export formats see [here](#).

5.7.2 Import

As mentioned above the import is context sensitive. It will only run if you select an xml file that fits to the page you are on and contains valid xml data. If the file is invalid or you accidentally picked a wrong document type, the parser will become aware of it and simply skip any import function. This will simply reload the page without any change of data.

You can select the file to import by clicking the `select xml file` text or by dragging and dropping your file onto the box containing the mentioned text.

A Project import will always create a new Project holding the imported data. *Note that for all other types (Options, Tasks, etc.) data already existing in your RDMO installation will be overwritten during an import.* The URI acts as identifier. If you import an element that equals the URI of an already existing one, the import will overwrite.

5.8 Role concept

You can involve other members to your projects. You have to assign them a `role` with certain permissions to make changes. At the moment there are the following roles:

- **Owner:** If you create a new project, you are automatically its owner. However, you can assign other members to be owners as well. Owners have all permissions including deleting a project.
- **Manager:** is not allowed to delete a project, but he has all other permissions including making snapshots and make changes.
- **Author:** has read and write permissions.
- **Guest:** has only read permission.

CHAPTER 6

Upgrade

The `rdmo` package can be conveniently upgraded using the `pip` command. However, before you perform any changes to your installation, please backup the important components to a save location.

A PostgreSQL or MySQL database can be database can be dumped into a file using:

```
pg_dump [DBNAME] > rdmo.sql # PostgreSQL  
mysqldump -uroot -p [DBNAME] > rdmo.sql
```

Your `rdmo-app` directory (including any `sqlite3` database) can be copied using the usual commands. Note that your virtual environment will not work after being moved to a different path.

In order to upgrade your RDMO installation go to your `rdmo-app` directory, activate your virtual environment, and upgrade the `rdmo` package using `pip`:

```
pip install --upgrade rdmo
```

In order to install a specific version (e.g. 0.9.0) of RDMO use:

```
pip install --upgrade rdmo==0.9.0
```

After the upgrade a database migration might be necessary:

```
python manage.py migrate
```

In any case you have to deploy the changes:

```
python manage.py deploy
```

Please check the release notes if this, or other, steps are necessary.

6.1 Upgrade to version 0.9.0

With version 0.9.0 we introduced the split into the `rdmo-app` and the centrally maintained `rdmo` package. Therefore a few additional steps are needed to upgrade any earlier version to 0.9.0 or beyond:

1. In any case perform a backup of your `rdmo` directory and your database as described above.
2. Perform the steps described in [clone](#) and [packages](#) as if you would install a new instance of RDMO.
3. Copy your old configuration from `/path/to/old/rdmo/rdmo/settings/local.py` to `/path/to/new/rdmo-app/config/settings/local.py`. The new `config` directory replaces the old `rdmo` directory.
4. If you already have a `theme` directory, copy it into the new `rdmo-app` folder.
5. Run a database migration (Unless you skipped several versions, the output should be `No migrations to apply.`):

```
python manage.py migrate
```

6. Download the front-end files from the CDN. We don't use bower and npm anymore.

```
python manage.py download_vendor_files
```

7. Update the path to the `wsgi.py` script in your Apache or nginx configuration. It is now under `/path/to/new/rdmo-app/config/wsgi.py`.
8. Redeploy RDMO as described under deployment of [Apache](#) or [Nginx](#).

If you have trouble with the upgrade process, don't hesitate to contact the RDMO team for support.

6.2 Upgrade to version 0.14

With version 0.14 the Python 2 support was dropped and we switched to Django 2.2. This demands two changes to the local `rdmo-app`:

- Adjust RDMO app's `config/urls.py` to Django2 schemes. The file is much simpler and shorter now. A working example can be found at <https://github.com/rdmorganiser/rdmo-app/blob/master/config/urls.py>
- `MIDDLEWARE_CLASSES` in `config/settings/local.py` needs to be renamed to `MIDDLEWARE` only