AniShare Documentation

Release 1.4

Holger Dinkel, Fabian Monheim

Jul 20, 2018

Contents:

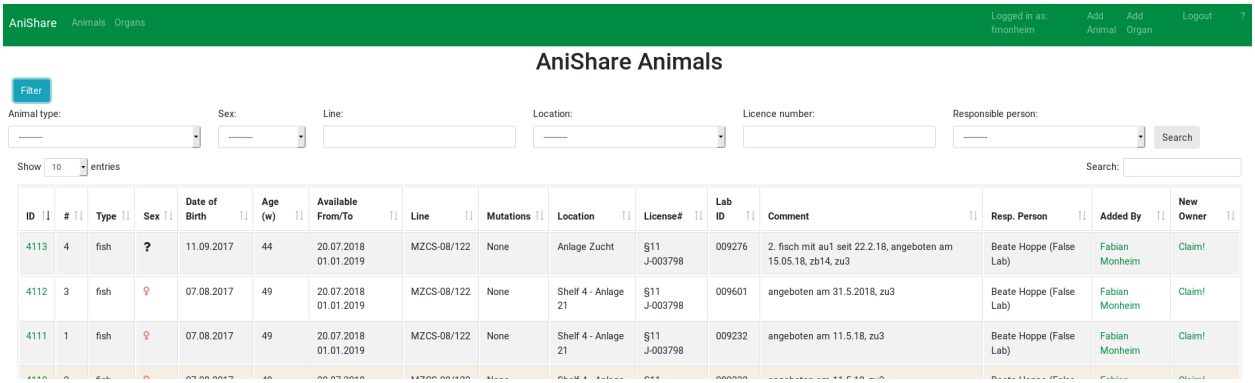
Chapter 1

Introduction

anishare is a webservice for research institutes to share animals with the goal to re-use animals and thus minimize total animal usage.

It has been developed at the Leibniz institute for aging research in Jena. This django app is meant to be used by researchers who want to share research animals with their colleagues. The basic idea is that animals are bred for experiments; however, sometimes, not all parts of the animal are used or sometimes an experiment gets cancelled for whatever reason. By sharing animals within the institute, less animals in total have to be sacrificed for research.

Anishare is a simple database of animals offered for reuse and a easy way to claim an animal with automatic generation of email messages as well as an RSS feed for updates.



At the moment, the software/database is geared towards handling of mice, however, it can be adjusted to handle any kind of research animal.

This documentation can also be downloaded as pdf file: Anishare Documentation

Chapter 2

Contact

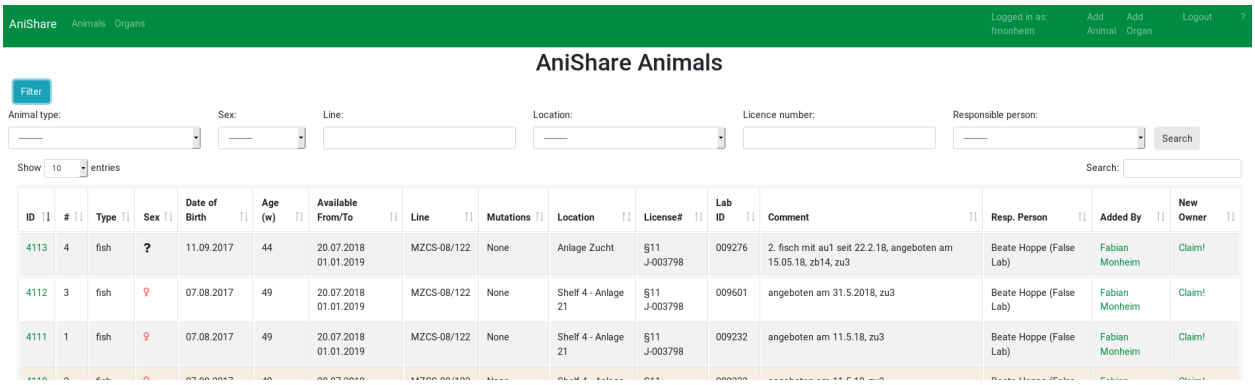
Technical and application support: Fabian Monheim, fabian.monheim@leibniz-fli.de, 03641-65-6872

Content support: mouse@leibniz-fli.de, fish@leibniz-fli.de

2.1 Main user interface

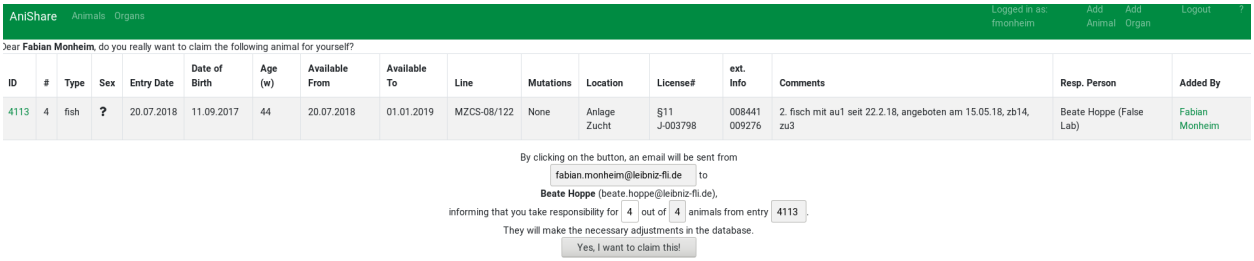
2.1.1 Animals

The main user-facing site is the list of animals to be shared. A user can browse this list, sort it via the headers or search for a term using the search bar.



If a user is interested in an animal, they should click on the button "Claim" which will bring up another page (see below) in which they can review their claim before finally submitting. When they click on "Yes, I want to claim this!", then they will be entered as *new owner* of this animal and an email will be send to them as well as the responsible/contact person informing them about this transaction. Further steps might need to be necessary such as transferring the animal in the LIMS (eg. PyRat).

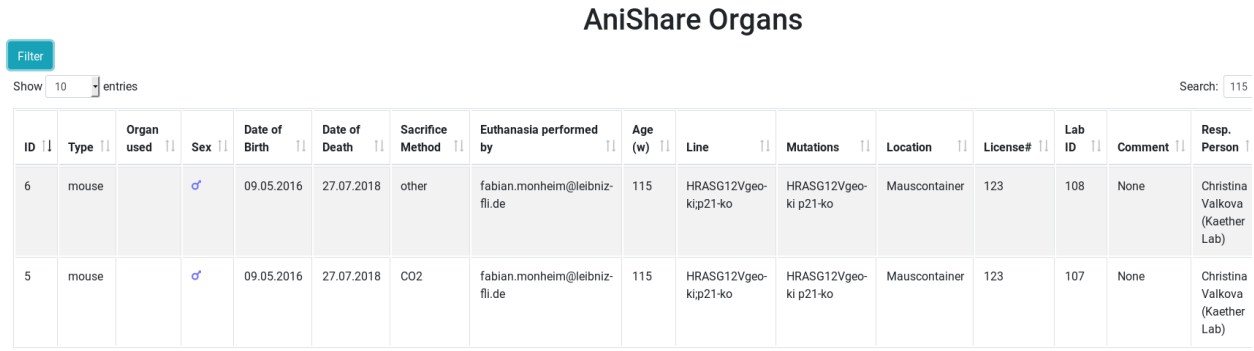
Note: If more than one animal is available, the user can adjust the number they want to claim. The remaining animals will still be available for claim. Because of uniqueness it's only possible to offer exactly one mouse per dataset. In contrast fishes can be offer in a group.



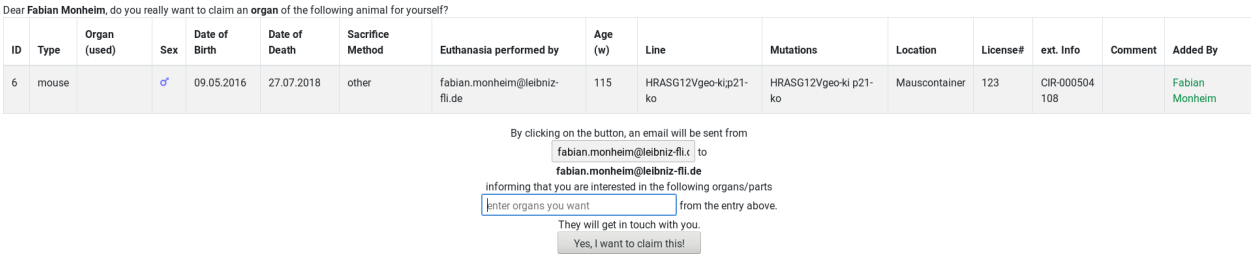
2.1.2 Organs

There exists an individual page for animal organ share. It is very similar to the animal page, however only individual organs are for offer. The entry at the column `Organ (used)` indicates all organs which can not be claimed. Also there is no availability period, but a day at which the animal gets sacrificed. The person responsible for sacrifice the animal will be informed via email if anybody claims some of the available organs. The entry will remain available to others (as they might want to claim other organs).

Organ index view:

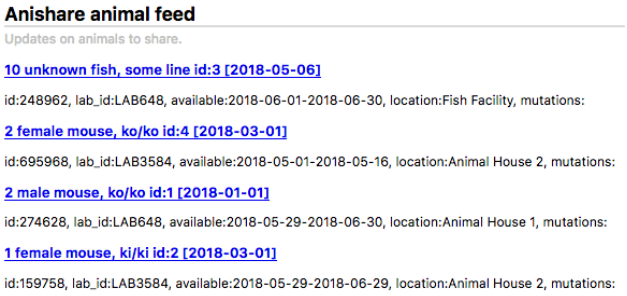


Organ claim view:



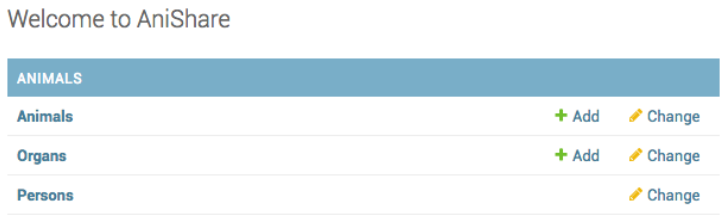
2.1.3 RSS Feed

An RSS feed containing the latest ten animals and organs is automatically generated and can be found at */animals/feed*. Users can subscribe (Most email clients allow the subscription to RSS feeds) to this feed to stay up-to-date with the animal catalogue. By clicking on a link in the feed, they are directed to the claim page of the individual animal/organ.



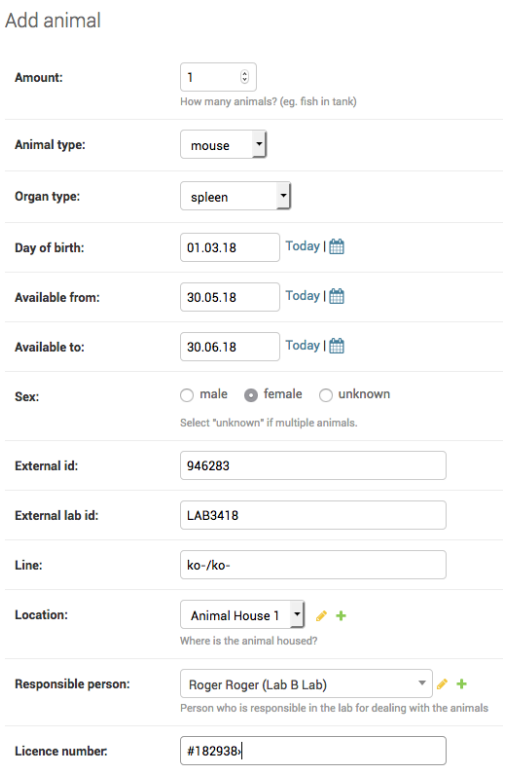
2.2 Main animal manager tasks

An *animal manager* can add animals and organs to the database on two ways. First, it is possible to add entrys manually. Secondly, it is possible to import an Excel sheet. At the FLI Jena there are two databases to manage animals. Now the databases are not connected. To transfer more than one or two datasets to anishare it's recommend to use the export function of *PyRAT* or *tick@lab* and the import function of anishare. To use the export/import process please read the topic **Animals import** or **Organs import**.



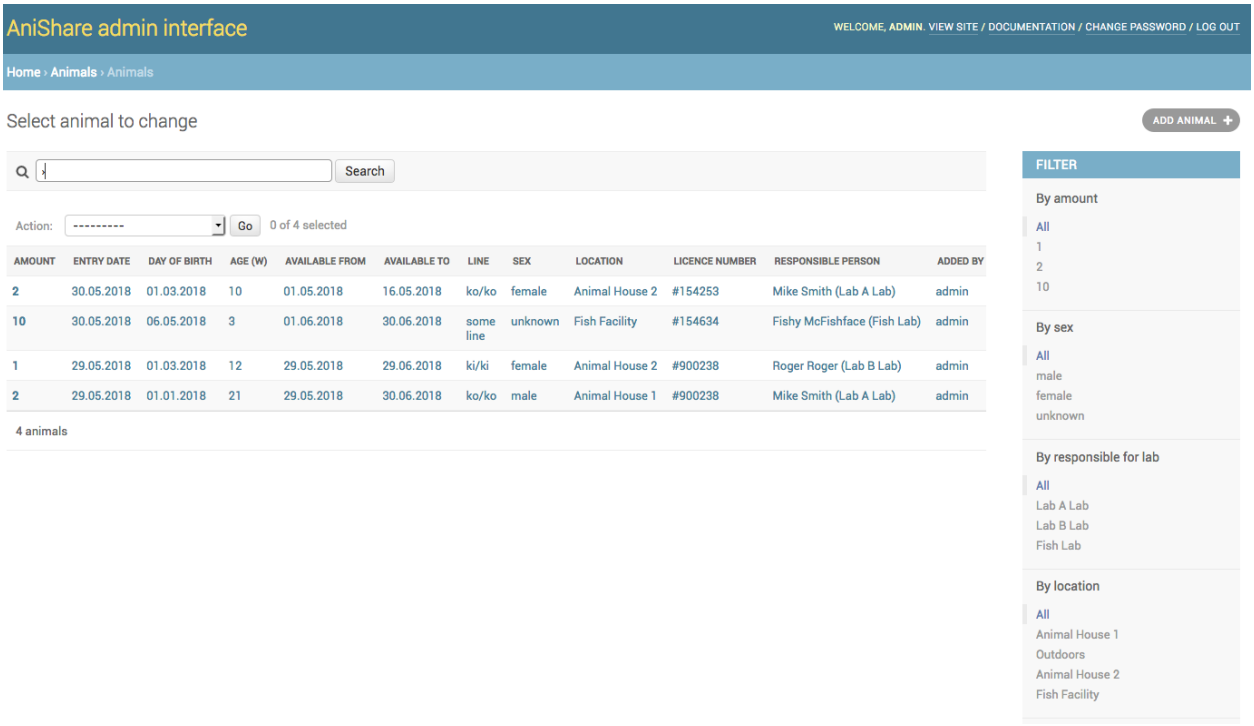
2.2.1 Add Animals manually

Click on `Animals` -> `Add` to add an animal.



All fields in bold **need** to be filled in, the others are optional.

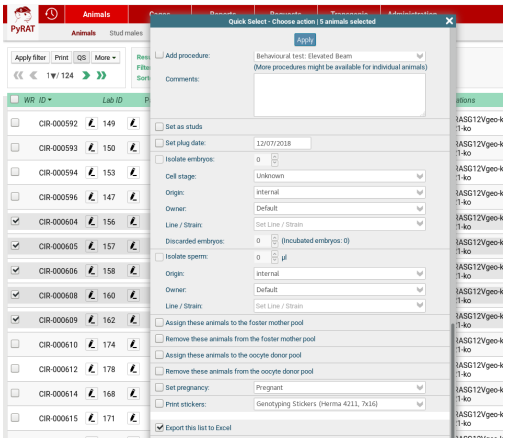
After adding several animals, the main (index) view should look like this:



2.2.2 Animals import

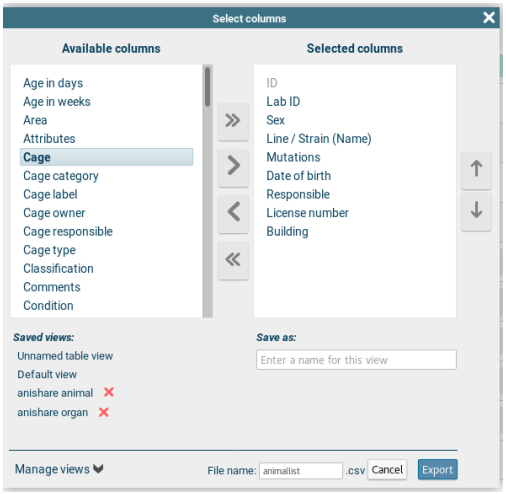
From PyRAT

First login to PyRAT and switch to the english version of PyRAT if it is no preset. Then select the animals which should be import to anishare. Click on `QS` (Quick Select) and activate the option `Export this list to Excel`. Push the button `Apply`.



Now it's important to select all mandatory fields: **ID, Lab ID, Sex, Line / Strine (Name), Mutations, Date of birth, Responsible, License number, Building**

It's possible to save the selected columns as a `Manage View` for reuse (fold out `Manage View` on the left side of the `File name`)



After downloading the file it's nessecary to edit the file because the coloumns **Animal type, Available from, Available to** are missing. Please add the mentioned columns (wherever) and fill it out. As **Animal type** the two values `mouse` or `fish` are possible. Please use the same date format for the coloumns **Available from, Available to** like at the exported coloumn **DOB** (Deate of birth). To simplify this process there are macros for LibreOffice and MS Office. The macros automatically add the missing coloumns and add the values `mouse` (Animal type), `Current Date` (Available from), `Current Date + 14 days` (Available to). Please refeer to the macro site to downloading the macros and further informations.

After adding the missing coloumns the file can be save, for example as xlsx file. Now go to the anishare admin interface to `Home › Animals › Animals` and click the button `IMPORT` (above the filter). Select the file and choose the file format. Upload the file. After submitting all datasets will show to the user if all requirements match.

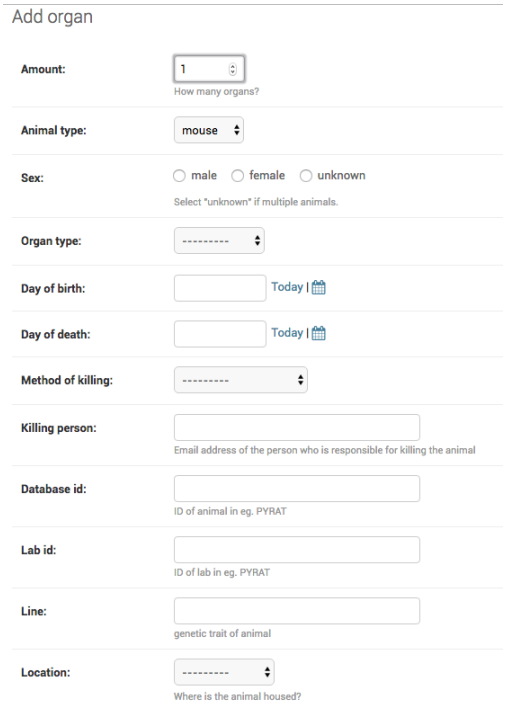
From *tick@lab*

First login to *Tick@lab* and open the population site. All visible entrys can be exported with the button *Export to Excel*. It isn't yet possible to export only selected animals. Therefore use the filter option.

To import the file it's nessecary to do a lot of changes to the structure of the data. So please use the macro which do the changes automatic. Please refeer to the macro site to downloading the macro and further informations.

2.2.3 Add Organs manually

Click on `Organs` -> `Add` to add an organ.



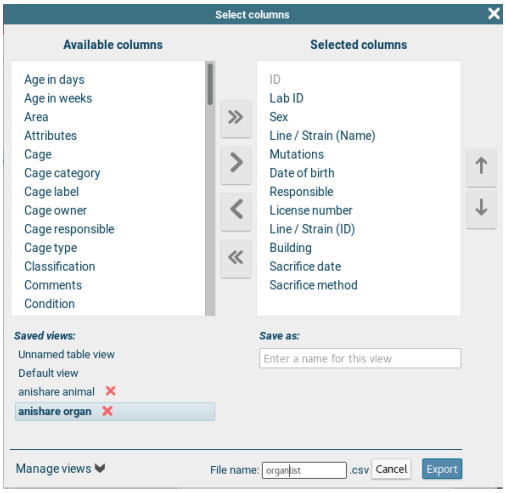
All fields in bold **need** to be filled in, the others are optional.

2.2.4 Organs import

From PyRAT

First login to PyRAT and switch to the english version of PyRAT if it is no preset. Then select the animals which should be import to anishare. Click on `QS` (Quick Select) and activate the option `Export this list to Excel`. Push the button `Apply`.

Now it's important to select all mandatory fields: **ID, Lab ID, Sex, Line / Strine (Name), Mutations, Date of birth, Responsible, License number, Building, Sacrifice date, Sacrifice method**



It's possible to save the selected columns as a `Manage View` for reuse (fold out `Manage View` on the left side of the `File name`)

After downloading the file it's nessecary to edit the file because the coloumns **Animal type, Available from, Available to** are missing. Furthermore the headings and the format of the coloumn **Responsible** needs to be adapt. To simplify this process there are macros for LibreOffice and MS Office. The macros automatically add the missing coloumns and add the values `mouse` (Animal type), `Current Date` (Available from), `Current Date + 14 days` (Available to). Please refeer to the macro site to downloading the macros and further informations.

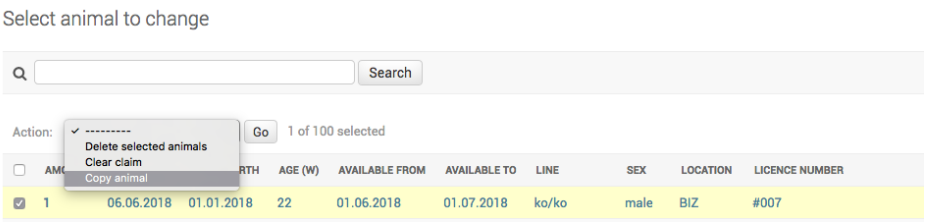
After adding the missing coloumns the file can be save, for example as xlsx file. Now go to the anishare admin interface to `Home › Animals › Animals` and click the button `IMPORT` (above the filter). Select the file and choose the file format. Upload the file. After submitting all datasets will show to the user if all requirements match.

From *tick@lab*

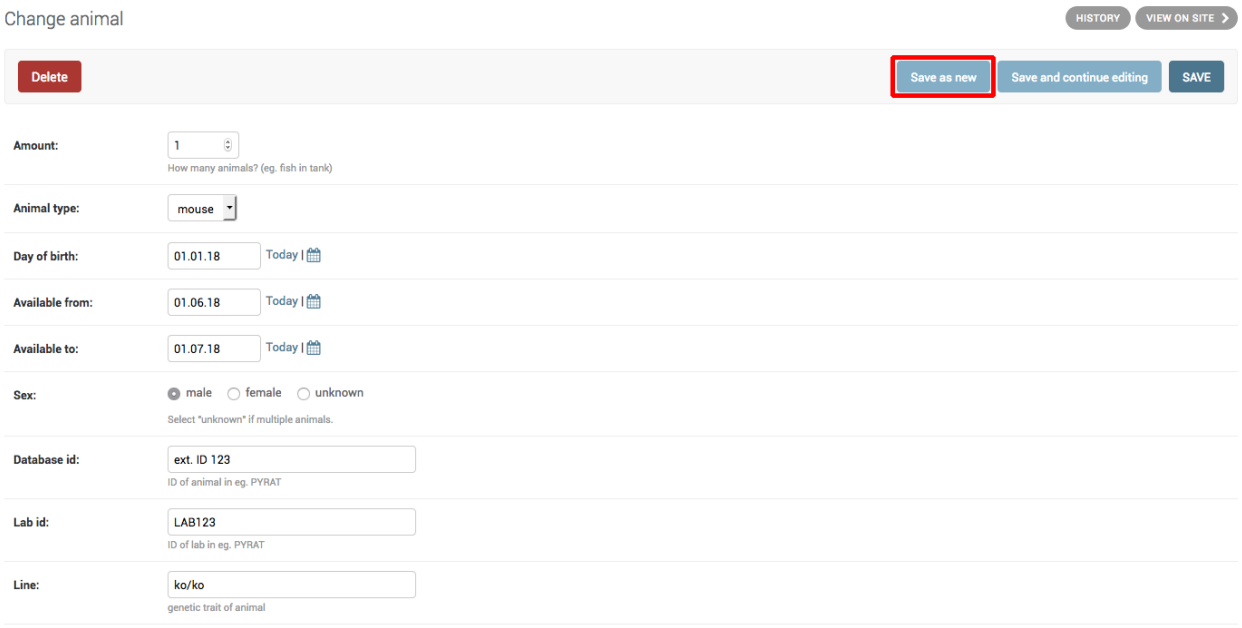
Because we expect only a small quantity of importing organs from *tick@lab* it's only possible to add entries manually.

2.2.5 Duplicating entries

For input of multiple similar entries, it is possible to duplicate an animal or organ entry. For this, select one or more entries in the list (see figure below) and select "*copy animal*" from the dropdown menu and click "*Go*".

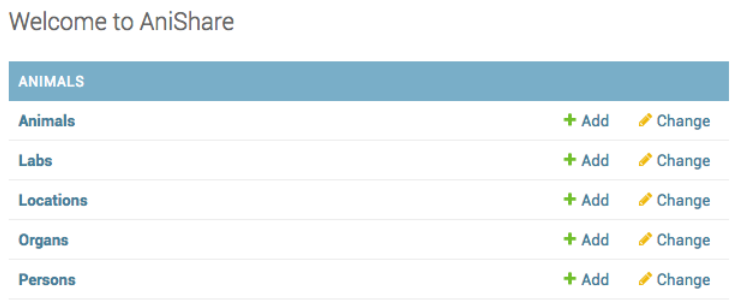


Another option is to edit an existing animal and click on "*Save as new*". This will save the currently edited animal as a new instance:



2.3 Main administrator tasks

The administrator can edit more objects in the admin interface, namely not just animals and organs but also labs, locations and persons:

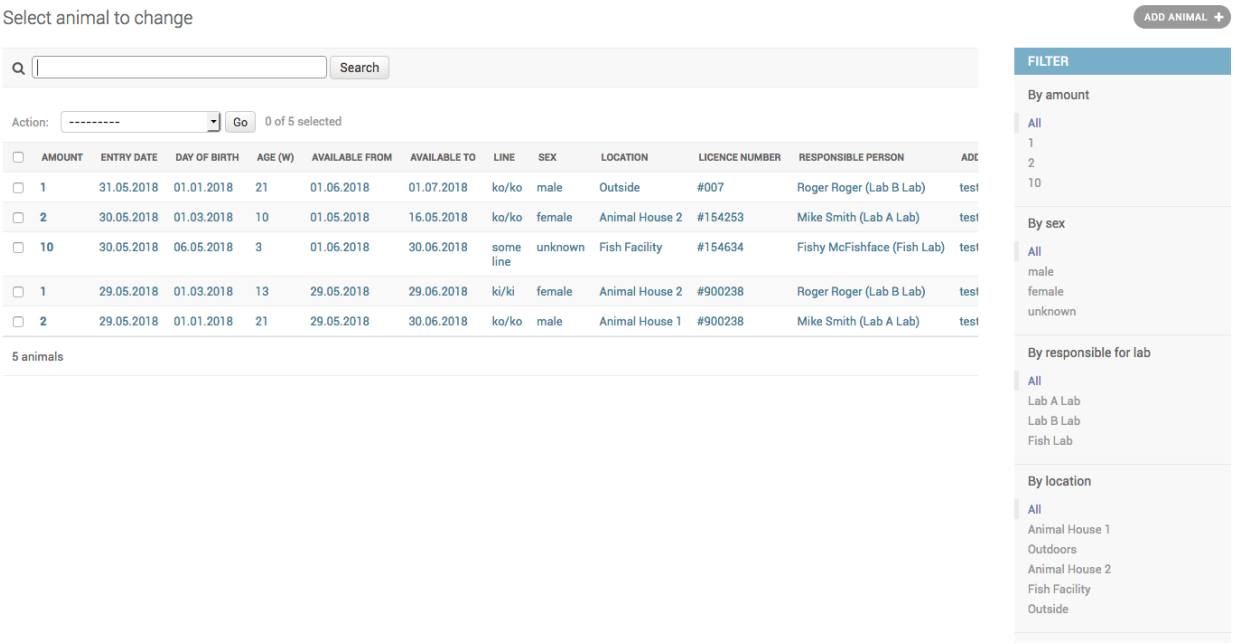


2.3.1 Organs used

These organs are standard values for the field **Organ used**.

2.3.2 Animals

The main category to administer are animals to share. Here, several filters (such as “sex”, “location”, etc.) are available to search for any set of animals.

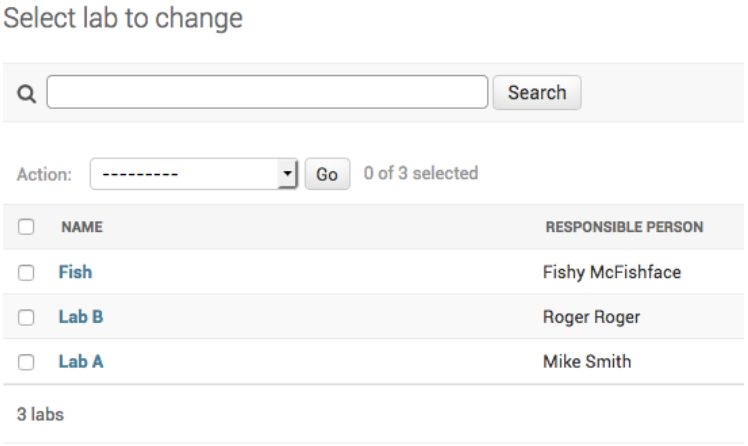


Note: in order to remove a claim (thus making the animal available again), either click on an animal and remove the email address from the field "new owner", or select one or multiple animals and select the "clear claim" *Action* and click "Go".

Note: Once created, an animal cannot be deleted, except by the administrator.

2.3.3 Labs

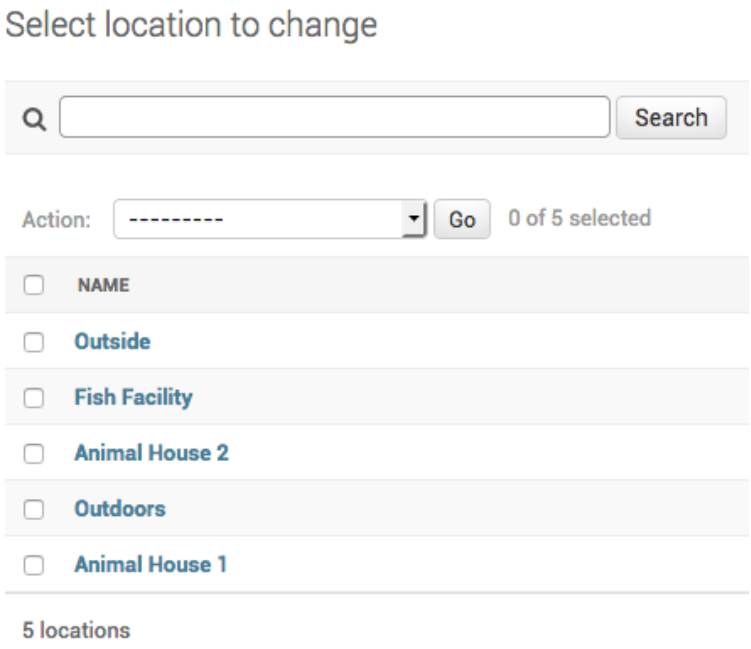
Labs are research labs/research groups and need to have at least one responsible/contact person each



Note: Only *administrators* are allowed to see and change Labs

2.3.4 Locations

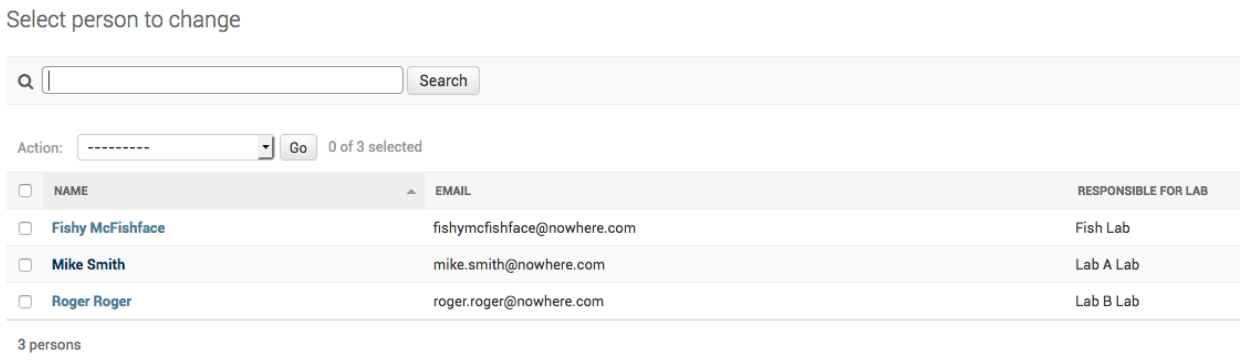
Locations are where animals are stored. Usually something like room numbers or "animal house" or "fish facility".



Note: Only *administrators* are allowed to see and change Locations

2.3.5 Persons

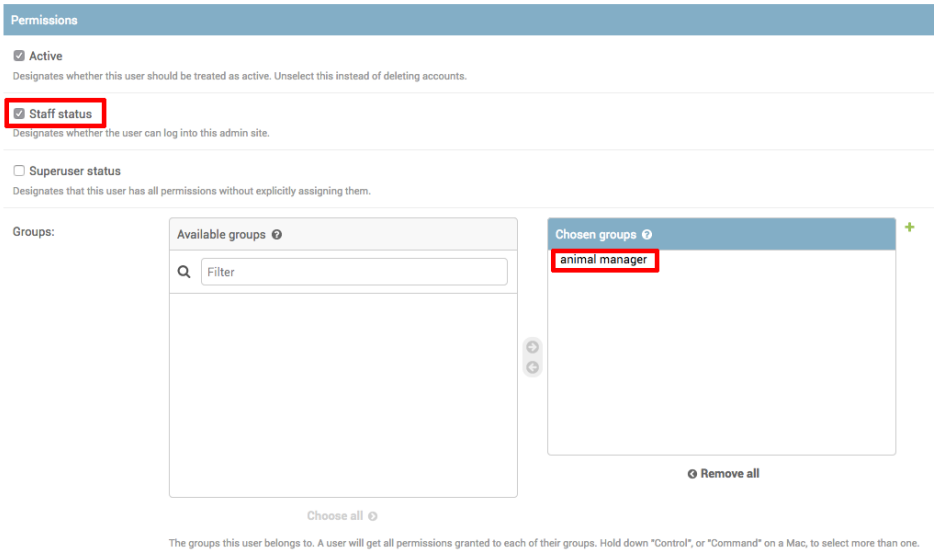
Persons responsible for the animals. Could be a vet or similar. Every animal needs to have a responsible person associated to them. This person then gets an email when the animal is being claimed.



Note: Only *administrators* are allowed to see and change Persons

2.3.6 Make a user an animal manager

The *administrator* is also responsible for user/rights management. In order to be able to add/edit animals, a user has to be in the group *animal manager* and have *staff status* in the django admin interface. For this, an *administrator* has to go to the user management in the admin interface by clicking "Home" -> "Authentication and Authorization" -> "Users". Here, they can make a *user* an *animal manager*, by setting these values (*staff* and group *animal manager*):



2.3.7 Anishare change history

New functions and bugfix at the system should be documented. So users can be informed about changes on the system. All changes are visible to authenticated users. Please refeer to the site AniShare Change History to see all changes.

Chapter 3

Installation

3.1 Requirements

We use the latest version of django, which requires python3. Install django and other dependancies (see file requirements.txt. We recommend using a virtual environment for this):

```
virtualenv -p python3 .
source bin/activate
pip install -r requirements.txt
```

3.2 First time setup

First, in the folder `anishare`, copy the file `local_settings.py.template` to `local_settings.py` and fill it in. If you want to use LDAP, comment in the respective lines. Most importantly, you should configure the following lines:

```
EMAIL_HOST = ''
SECRET_KEY = ''
ALLOWED_HOSTS = ['127.0.0.1', ]
```

Then, you can run migrations:

```
python manage.py migrate
```

Note: This will create the sqlite database `db.sqlite3` containing all the models (eg. tables) as defined in *`animals.models`*.

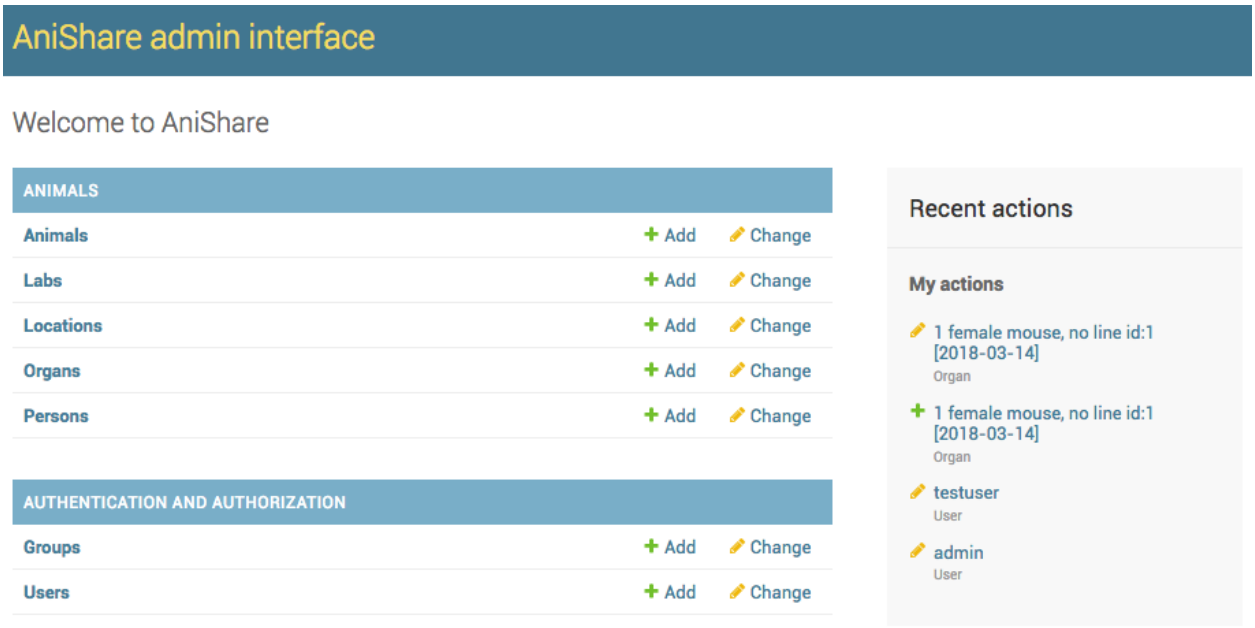
Now create a superuser:

```
python manage.py createsuperuser
```

You are now able to login to the admin interface, but first run the dev server:

```
python manage.py runserver
```

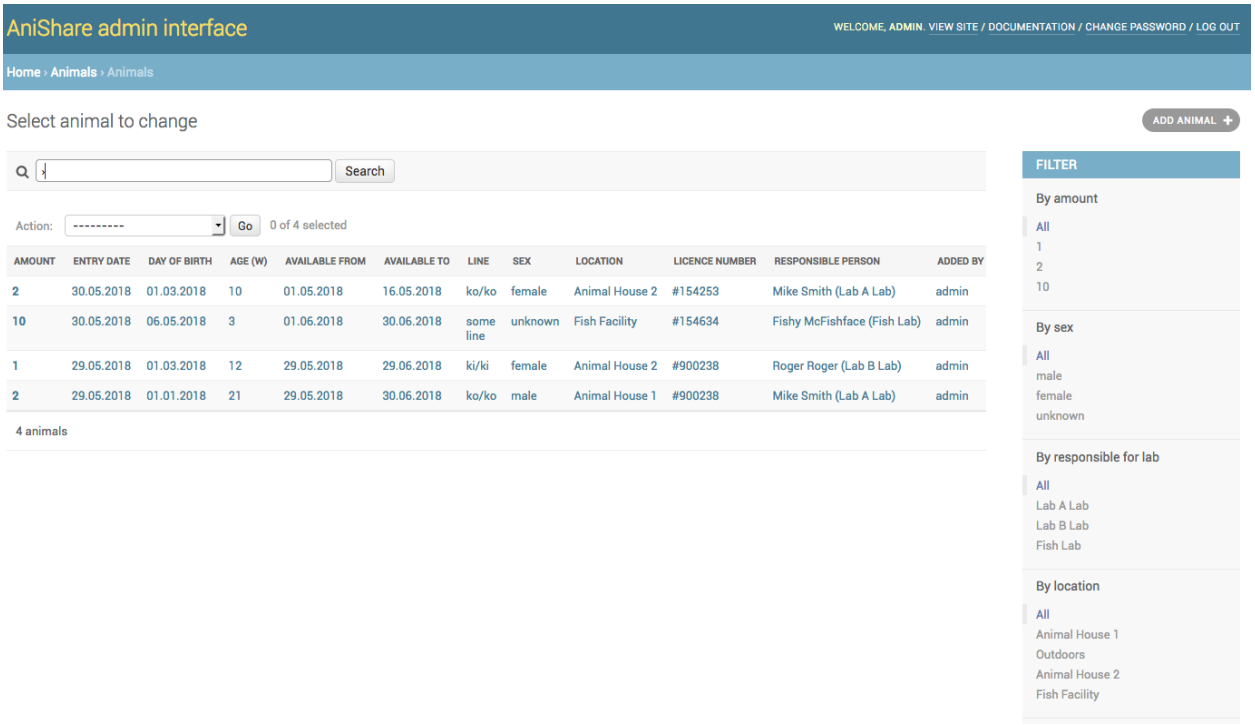
This will listen on `http://localhost:8000`, so browse to the admin page `http://localhost:8000/admin` and you should see this after login:



You can also import a dummy set of data using the `loaddata` command:

```
python manage.py loaddata initial_data.json
```

After loading the data, the main admin interface should look like this:



3.3 Importing existing data

For import of existing data in tabular (excel) format, a management command is available at `animals.management.commands.import_animals`

```
python manage.py import_animals
```

Note: See the file `example_import.xls` for an example...

		Information exported from PYRAT:								Information to add manually			
Nr.	Entry date	ID	Lab ID	DOB	Age (w)	Line / Strain	Sex	Building	Licence numb	Responsible p	Available fror	Available unt	New Owner
1	01.05.18	ext. ID 123	LAB123	01.01.18	10	ko/ko	m	Outside	#007	Roger Roger	01.06.18	01.07.18	

3.4 In-DB Caching

By default, database caching is enabled in settings. To create the necessary tables, run this command:

```
python manage.py createcachetable
```

This will create a cache table in the SQLite database, which will speed up queries.

3.5 Running Tests

Tests reside in `animals/tests.py`. You can invoke the django tests like so:

```
python manage.py test
```

```
"""
Tests for Anishare website
"""
from django.test import TestCase, Client
from django.contrib.auth.models import User
from django.core.management import call_command

class GetAnimalsTest(TestCase):
    """
    Test module to GET Animal pages
    """

    def setUp(self):
        """ Creating a user first and loading fixtures"""
        call_command('loaddata', 'initial_data.json', verbosity=0)  # Load fixtures
        self.user = User.objects.create_user(pk=1, username='testuser', password=
↪'12345')
        self.client = Client()

    def test_get_all_animals(self):
        """ try to retrieve all animals """
        response = self.client.get('/animals/')
        self.assertEqual(response.status_code, 302)
```

(continues on next page)

(continued from previous page)

```
        self.client.login(username='testuser', password='12345')
        response = self.client.get('/animals/')
        self.assertEqual(response.status_code, 200)

    def test_get_one_animal(self):
        """ try to retrieve individual animals """
        response = self.client.get('/animals/1')
        self.assertEqual(response.status_code, 302)
        self.client.login(username='testuser', password='12345')
        response = self.client.get('/animals/1')
        self.assertEqual(response.status_code, 200)

    def test_claim_one_animal(self):
        """ try to claim individual animals """
        response = self.client.get('/animals/claim/1')
        self.assertEqual(response.status_code, 302)
        self.client.login(username='testuser', password='12345')
        response = self.client.get('/animals/claim/1')
        self.assertEqual(response.status_code, 200)
```

3.5.1 Upgrading django

To upgrade django or any other python library for anishare, go into the anishare directory, and activate its virtualenv:

```
cd anishare
source bin/activate
```

Next, install/upgrade whatever library (here: django to the latest version):

```
pip install --upgrade django
```

Note: It's best to test the latest version in a local/development environment first!

3.5.2 Upgrading python

When upgrading the python version of the host operating system, it might be necessary to also upgrade the python in the virtualenv. Otherwise an error like the following might occur:

```
python: error while loading shared libraries: libpython3.4m.so.1.0:
cannot open shared object file: No such file or directory
```

In that case, go into the anishare directory, and delete the following directories:

- bin
- include
- lib
- lib64

Afterwards, create a new virtualenv and install the required libraries like so:

```
virtualenv -p python3 .
source bin/activate
pip install -r requirements.txt
```

Chapter 4

API documentation

4.1 Admin

Admin module

class animals.admin.**AnimalAdmin**(*model*, *admin_site*)
ModelAdmin for Animal model

age(*obj*)
Show the age in the admin as 'Age (w)' instead of 'age'

form
alias of *AnimalForm*

resource_class
alias of *AnimalResource*

save_model(*request*, *obj*, *form*, *change*)
Given a model instance save it to the database.

class animals.admin.**AnimalForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*)
Form for animal editing in admin

clean()
Hook for doing any extra form-wide cleaning after Field.clean() has been called on every field. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field named '__all__'.

class animals.admin.**AnimalResource**

get_instance(*instance_loader*, *row*)
Calls the InstanceLoader.

import_obj(*instance*, *row*, *dry_run*)
Traverses every field in this Resource and calls `import_field()`.

class animals.admin.**ChangeAdmin**(*model*, *admin_site*)
ChangeAdmin for Change model

class animals.admin.**LabAdmin**(*model*, *admin_site*)
ModelAdmin for Lab model

class animals.admin.**LocationAdmin**(*model*, *admin_site*)
ModelAdmin for Location model

class animals.admin.**OrganAdmin**(*model*, *admin_site*)
ModelAdmin for Organ model

age(*obj*)
Show the age in the admin as 'Age (w)' instead of 'age'

resource_class
alias of *OrganResource*

save_model(*request*, *obj*, *form*, *change*)
Given a model instance save it to the database.

class animals.admin.**OrganResource**

get_instance(*instance_loader*, *row*)
Calls the InstanceLoader.

import_obj(*instance*, *row*, *dry_run*)
Traverses every field in this Resource and calls `import_field()`.

class animals.admin.**OrgantypeAdmin**(*model*, *admin_site*)
ModelAdmin for Organ types

class animals.admin.**PersonAdmin**(*model*, *admin_site*)
ModelAdmin for Person model

animals.admin.**clear_claim**(*modeladmin*, *request*, *queryset*)
Convenience Function to delete a claim from several selected animals

animals.admin.**copy_animal**(*modeladmin*, *request*, *queryset*)
Copy an instance of an animal so similar entries can be easily created.

4.2 Models

This file describes all the models in the database.

class animals.models.**Animal**(**args*, ***kwargs*)
Main model containing the animals.

exception DoesNotExist

exception MultipleObjectsReturned

age()
Return the age of the animal, calculated by the difference to either the current date or the available_to date

available()
Returns True if the animal is still available

clean()
Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

description()
Return description of this model

get_absolute_url()
Get absolute url for this model. Important to link from the admin.

class animals.models.**Change**(**args*, ***kwargs*)
Model for documentation all changes to anishare

exception DoesNotExist

exception MultipleObjectsReturned

class animals.models.**Lab**(**args*, ***kwargs*)
Labs are only defined by a name and are referenced by Person(s) which are responsible (contact) person for this lab

exception DoesNotExist

exception MultipleObjectsReturned

responsible_person()
Retrieve only the person(s) which are responsible for this lab.

class animals.models.**Location**(**args*, ***kwargs*)
Location of animals. Eg. animal house, fish facilities etc.

exception DoesNotExist

exception MultipleObjectsReturned

class animals.models.**Organ**(**args*, ***kwargs*)
Model containing the organs

exception DoesNotExist

exception MultipleObjectsReturned

age()
Return the age of the animal, at the time of death

available()
Returns True if the animal is still available

description()
Return description of this model

get_absolute_url()
Get absolute url for this model. Important to link from the admin.

get_organtypes()
Get all organ types which are used

class animals.models.**Organtype**(**args*, ***kwargs*)
Model containing the organ types

exception DoesNotExist

exception MultipleObjectsReturned

class animals.models.**Person**(**args*, ***kwargs*)
The responsible (contact) person for each lab. This person gets an email when an animal is being claimed.

exception DoesNotExist

exception MultipleObjectsReturned

4.3 Views

Django Views contains all the functions for rendering objects (HTML display). It also contains an RSS Feed generator class to create an RSS feed from newly created animals

Important: When adding new functions, use the login_required decorator When adding new classes, use the LoginRequiredMixin

class animals.views.**AnimalDetailView**(***kwargs*)
Detail view for an animal. This is rarely used, rather use the claim page.

model
alias of *animals.models.Animal*

class animals.views.**LatestAnimalsFeed**
RSS Feed for new animals/organs.

item_description(*item*)
What to print as item description (use default description from model).

item_title(*item*)
What to print as item title (use default __str__ of model).

items()
Get latest animals as items.

animals.views.**claim**(*request*, *primary_key*)
View to claim an animal.

Parameters **primary_key** – the id/pk of the animal to retrieve

Returns rendered page with the claim form or 404 if animal not found

animals.views.**claim_organ**(*request*, *primary_key*)
View to claim an organ.

Parameters **primary_key** – the id/pk of the organ to retrieve

Returns rendered page with the claim form or 404 if organ not found

animals.views.**send_email_animal**(*request*)
Function to send an email about an animal being claimed.

Needs these variables in the POST request: email, pk, count

Parameters

- **email** – email address of the request user / new owner
- **pk** – primary_key of the animal(s) to be claimed
- **count** – how many animals are being claimed

animals.views.**send_email_organ**(*request*)
Function to send an email about an animal being claimed.

Needs these variables in the POST request: email, pk, count

Parameters

- **email** – email address of the request user / new owner
- **pk** – primary_key of the animal(s) to be claimed
- **organs_wanted** – organs wanted from the given animal

4.4 URLs

animals URL Configuration

The *urlpatterns* list routes URLs to views. For more information please see: https://docs.djangoproject.com/en/2.0/topics/http/urls/

Examples:

Function views:

1. Add an import: from my_app import views
2. Add a URL to urlpatterns: path(‘’, views.home, name=’home’)

Class-based views:

1. Add an import: from other_app.views import Home
2. Add a URL to urlpatterns: path(‘’, Home.as_view(), name=’home’)

Including another URLconf:

1. Import the include() function: from django.urls import include, path
2. Add a URL to urlpatterns: path(‘blog/’, include(‘blog.urls’))

Python Module Index

a

Index

P

R

S