anishare Documentation

Release 1.1

Holger Dinkel

Jun 06, 2018

Contents:

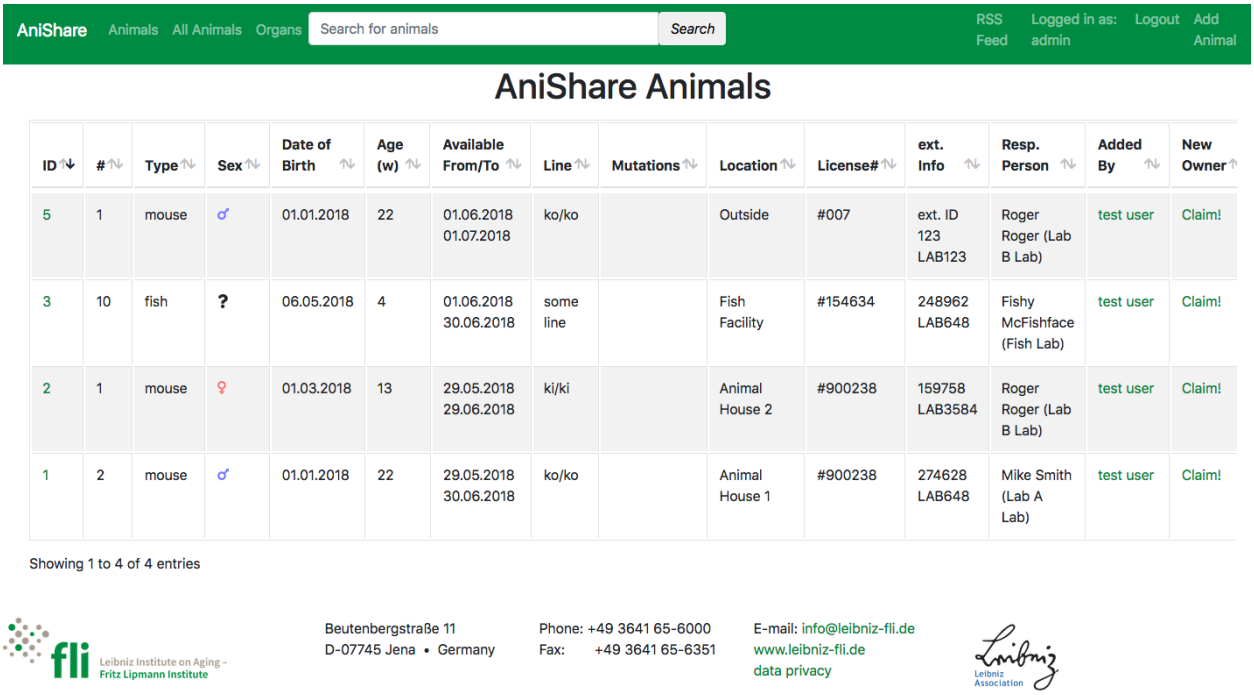
Chapter 1

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.

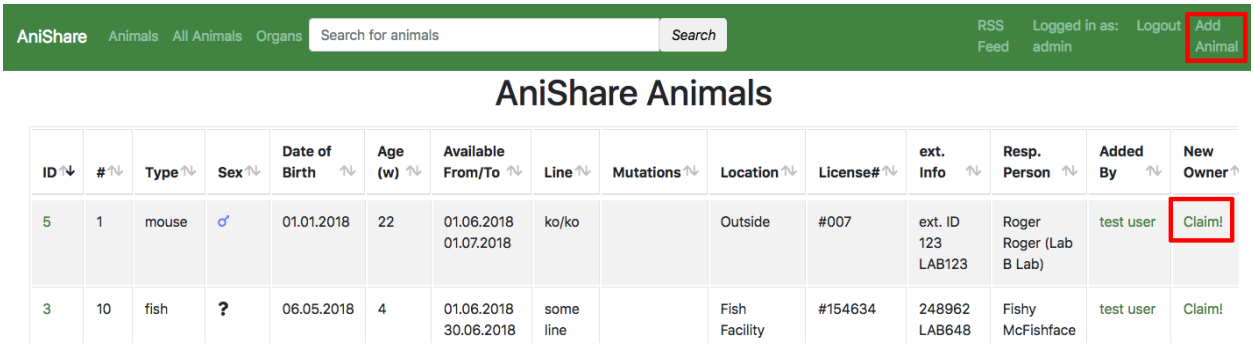
Anishare is a simple database of animals offered for reuse and a easy way to claim an animal with automatic generation of email messages as well as an RSS feed for updates.



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

Chapter 2

Using the software

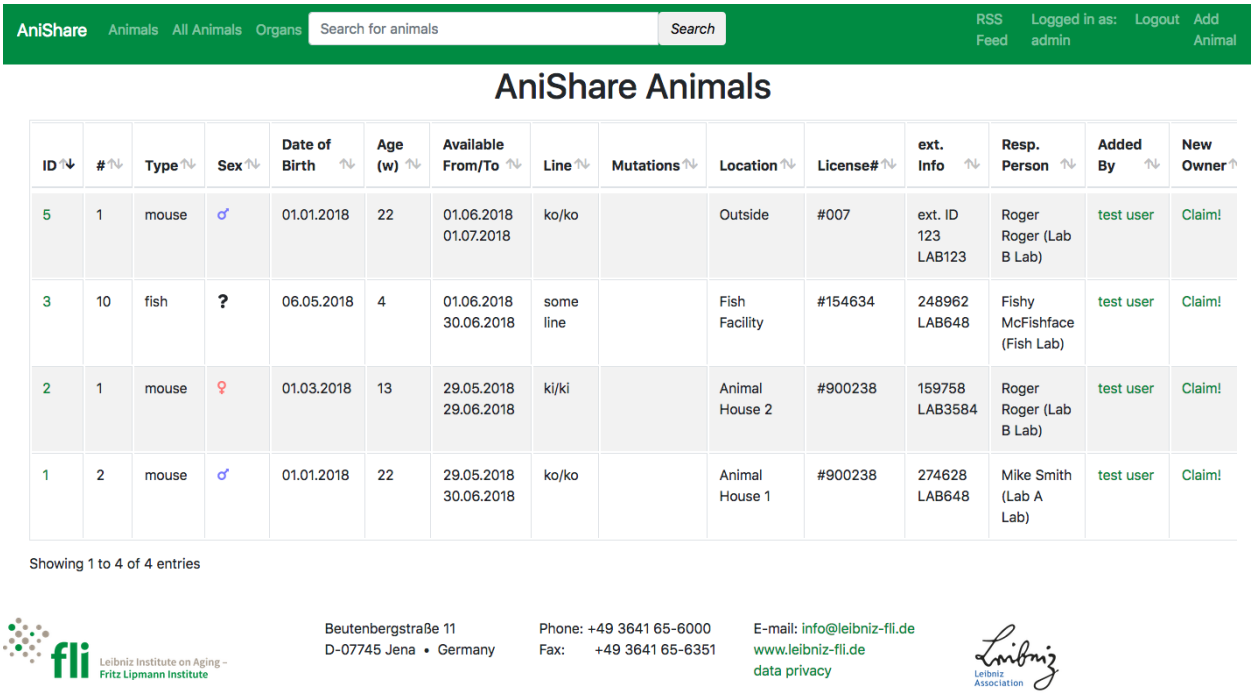


The webservice is split in two parts: The **animal input** method is via the Django Admin interface (See link "Add Animal" top right) and is meant for *animal managers* only. The **claim** method is via the normal web interface and is meant for normal users (who need to be authenticated, though).

2.1 Main user interface

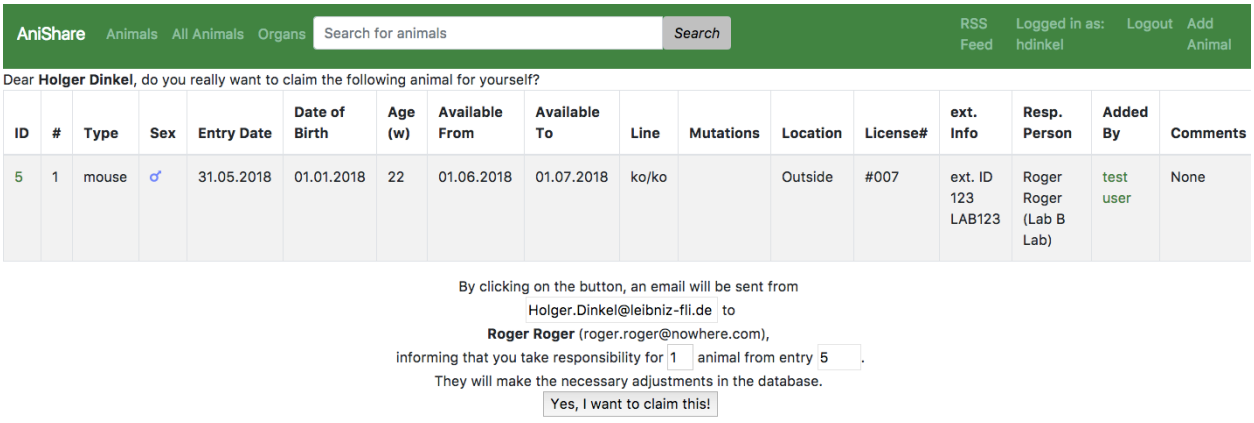
2.1.1 Animals

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).

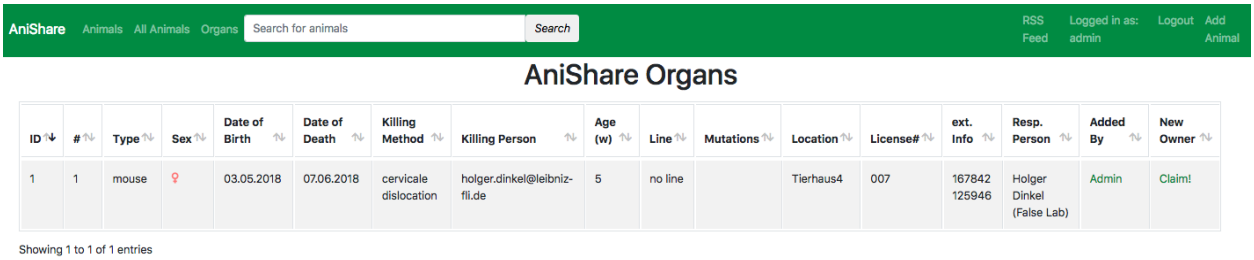
Note: If more than one animal is available, the user can adjust the number they want to claim. The remaining animals will still be available for claim.



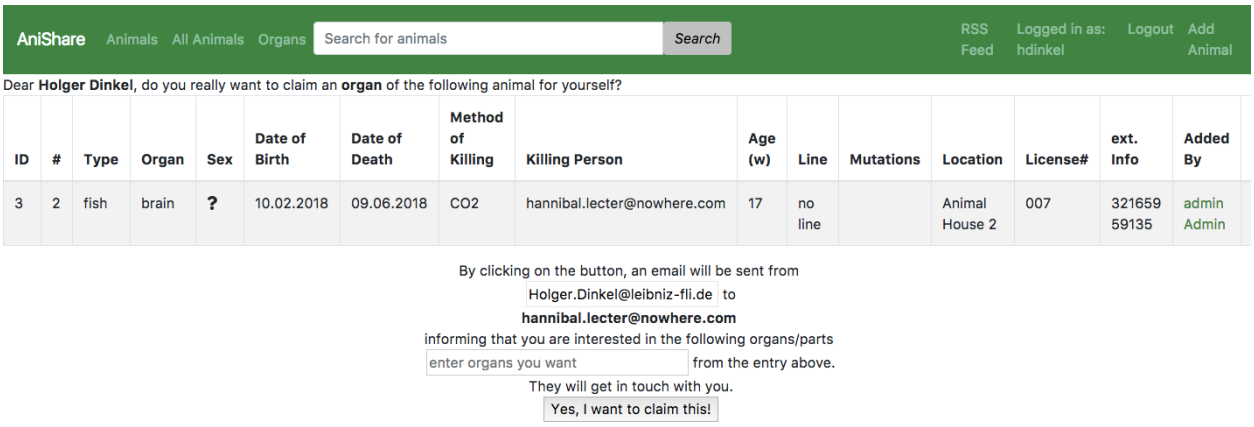
2.1.2 Organs

There exists an individual page for animal organ share. It is very similar to the animal page, however only individual organs are for offer. Also there is no availability period, but a day at which the animal gets sacrificed. The person responsible for killing 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:

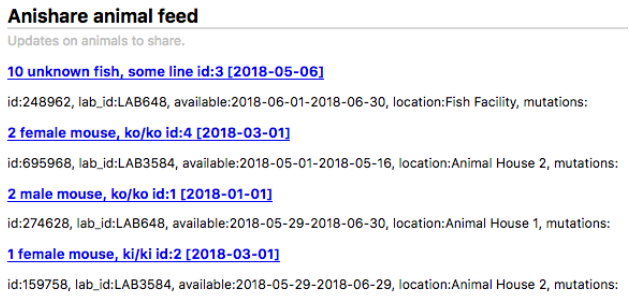


Organ claim view:



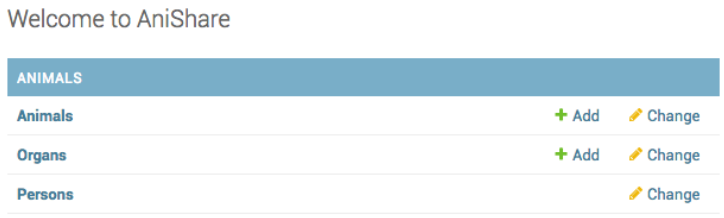
2.1.3 RSS Feed

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.



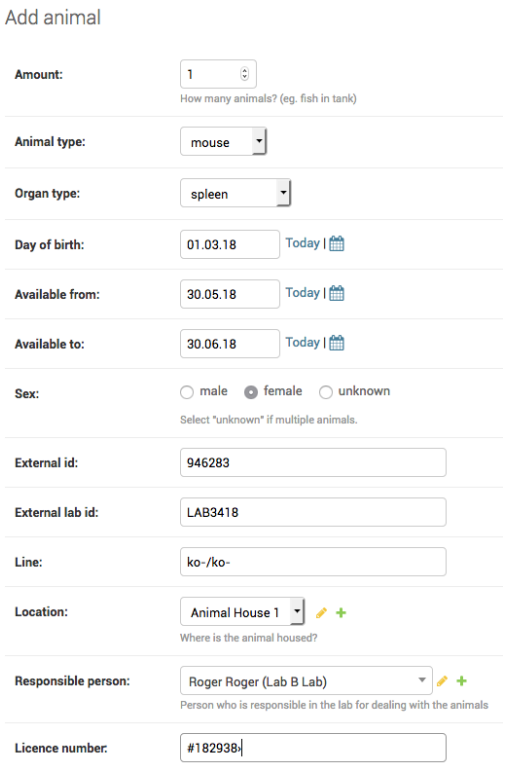
2.2 Main animal manager tasks

An *animal manager* can add animals and organs to the database.



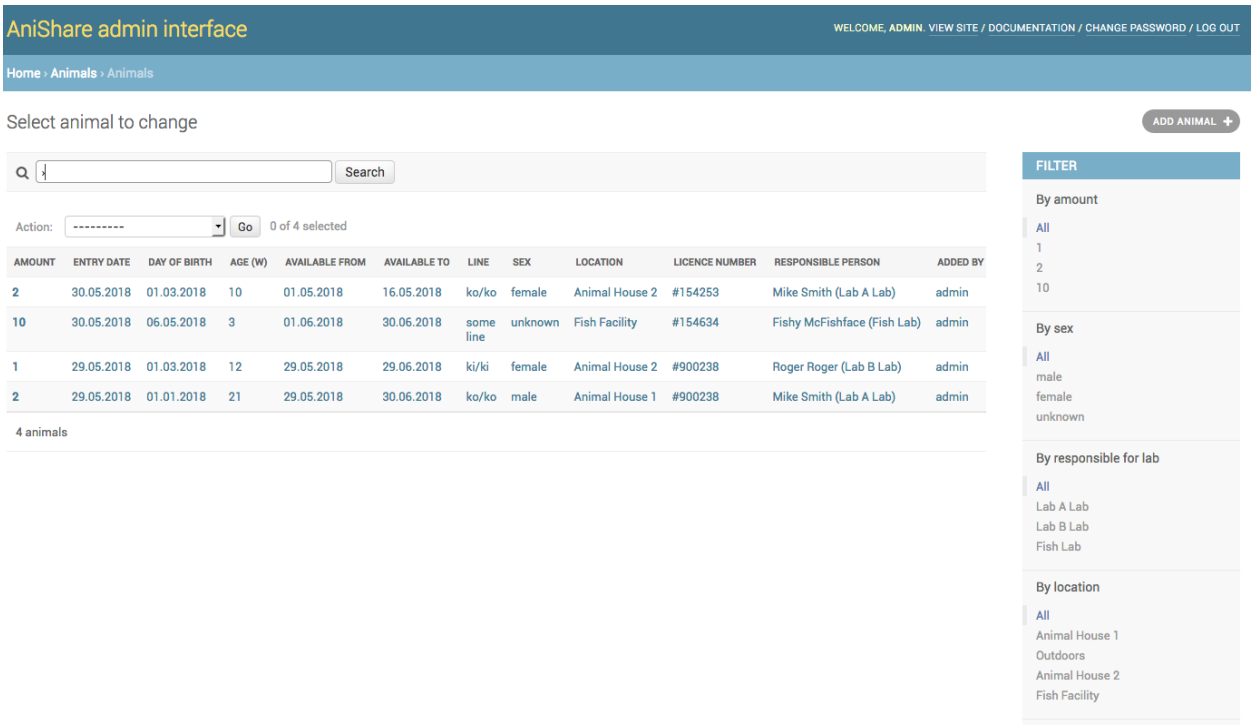
2.2.1 Animals

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



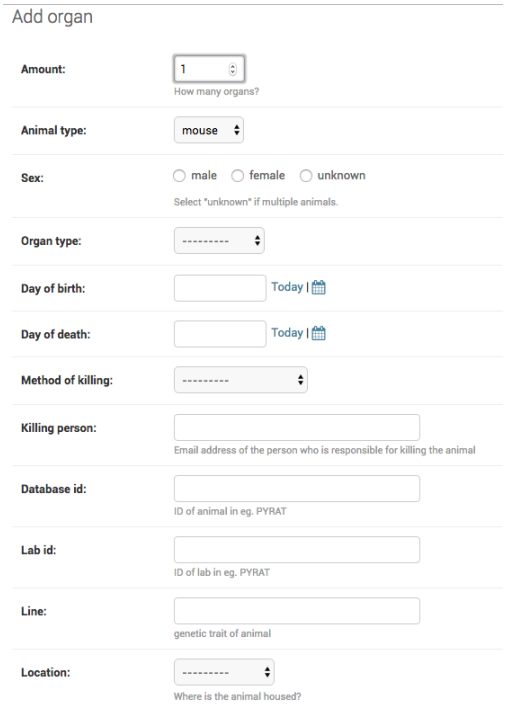
All fields in bold **need** to be filled in, the others are optional.

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



2.2.2 Organs

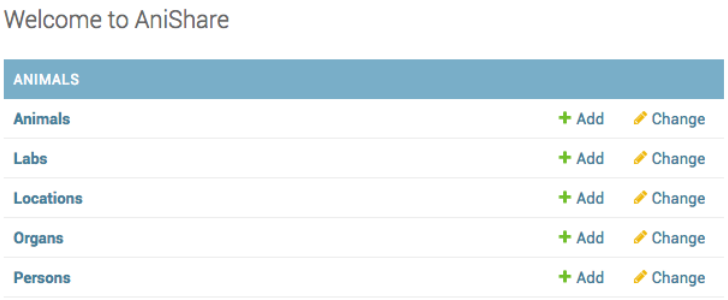
Click on `Organs` -> `Add` to add an organ.



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

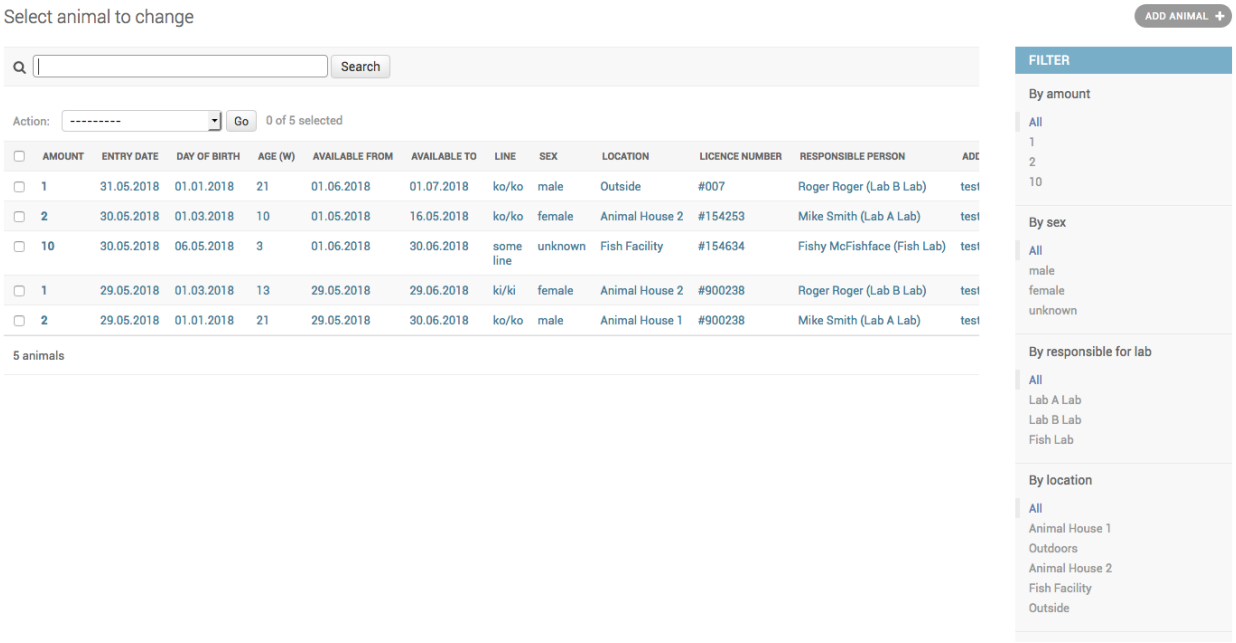
2.3 Main admin tasks

The admin interface allows to edit the following types of entries:



2.3.1 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.

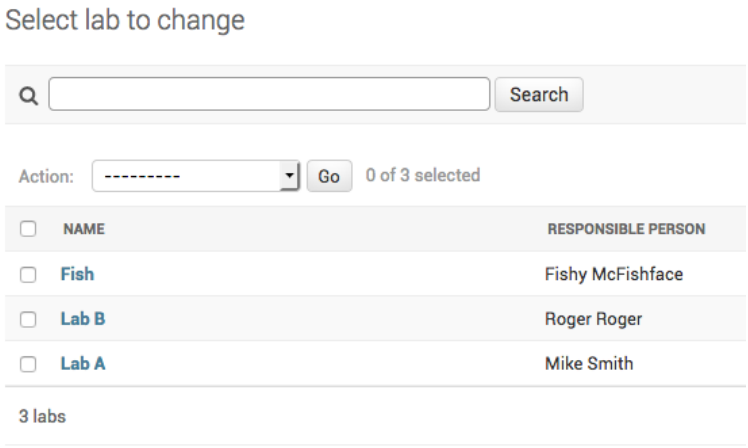


Note: in order to remove a claim (thus making the animal available again), either click on an animal and remove the email address from the field "new owner", or select one or multiple animals and select the "clear claim" *Action* and click "Go".

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

2.3.2 Labs

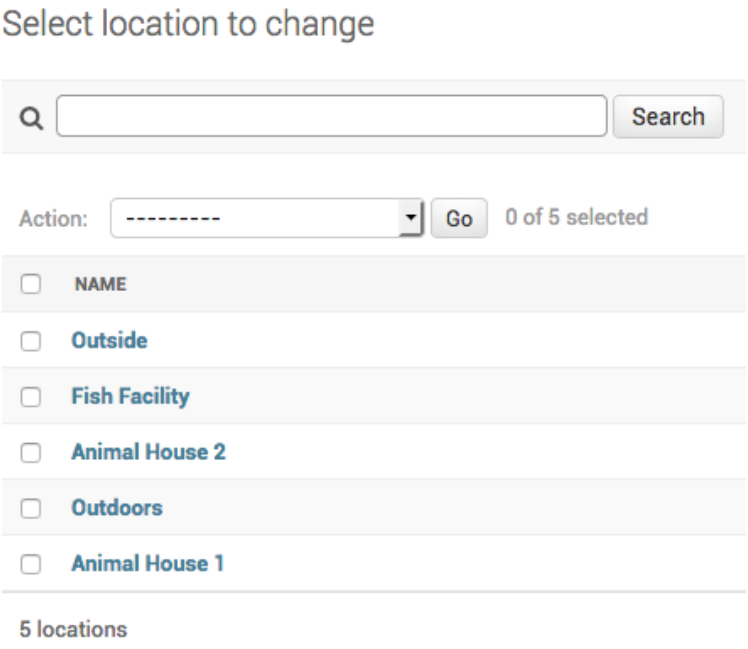
Labs are research labs/research groups and need to have at least one responsible/contact person each



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

2.3.3 Locations

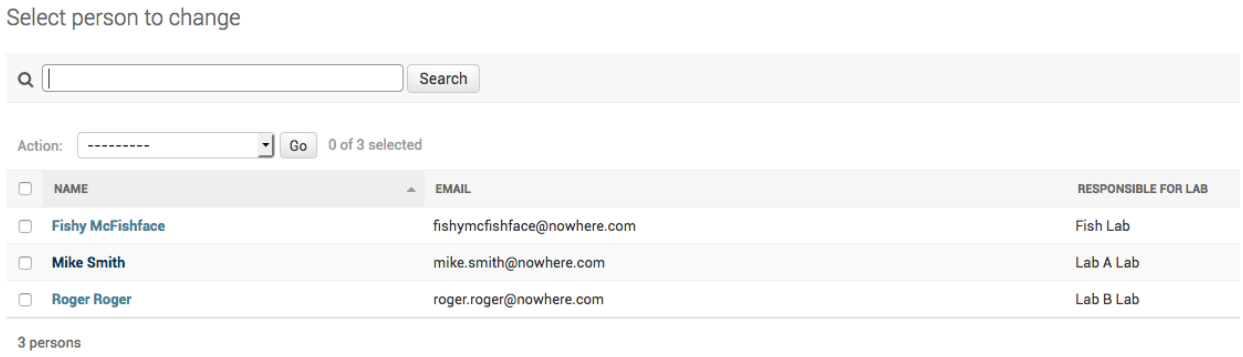
Locations are where animals are stored. Usually something like room numbers or "animal house" or "fish facility".



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

2.3.4 Persons

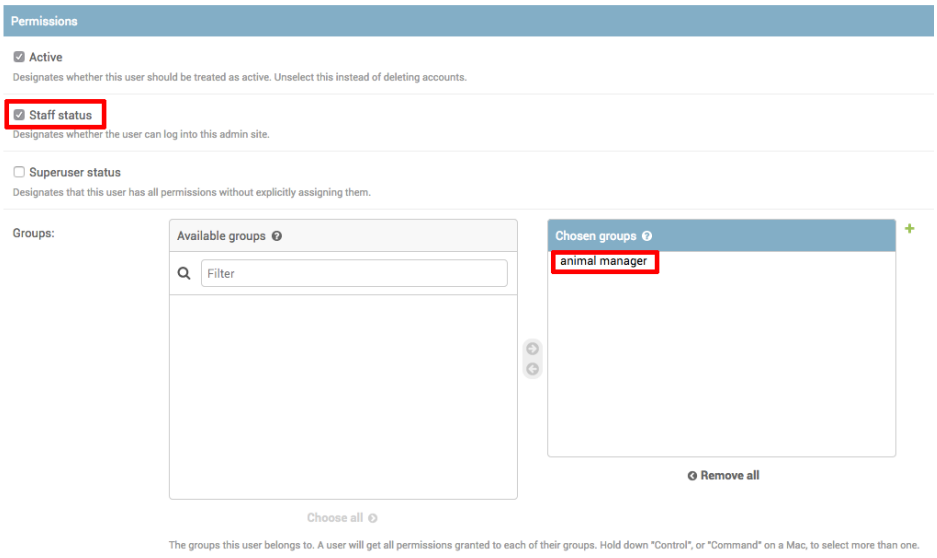
Persons responsible for the animals. Could be a vet or similar. Every animal needs to have a responsible person associated to them. This person then gets an email when the animal is being claimed.



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

2.3.5 Make a user an animal manager

In order to be able to add/edit animals, a user has to be in the group *animal manager* and have *staff status* in django admin. 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*):



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`.

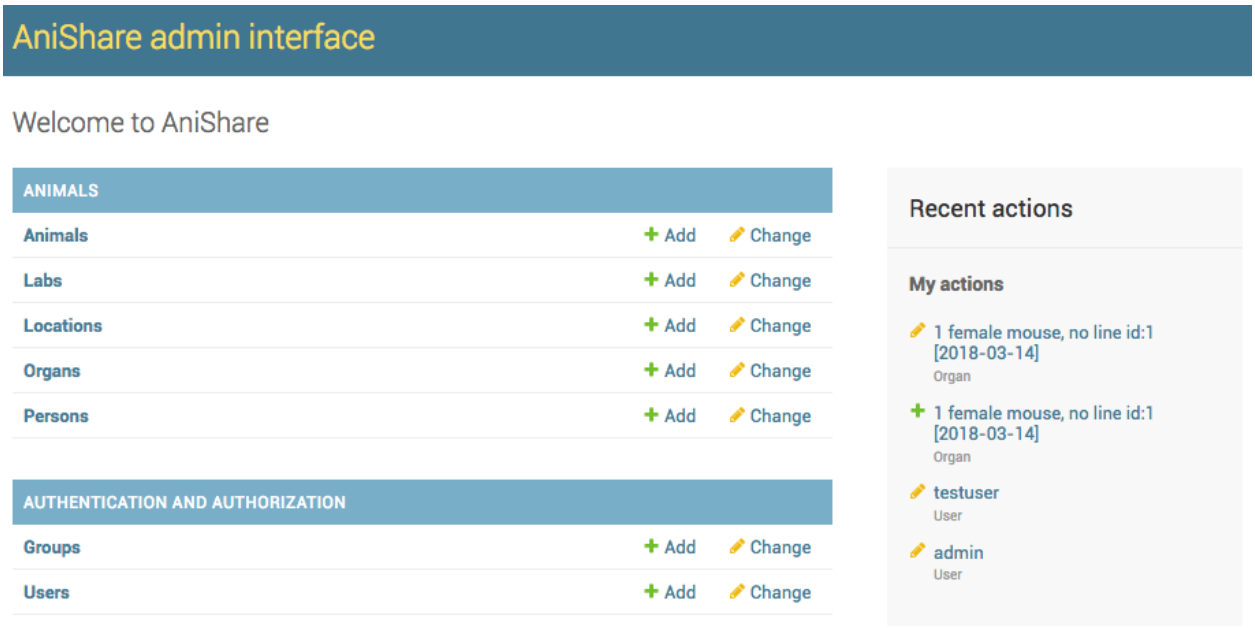
Now create a superuser:

```
python manage.py createsuperuser
```

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

```
python manage.py runserver
```

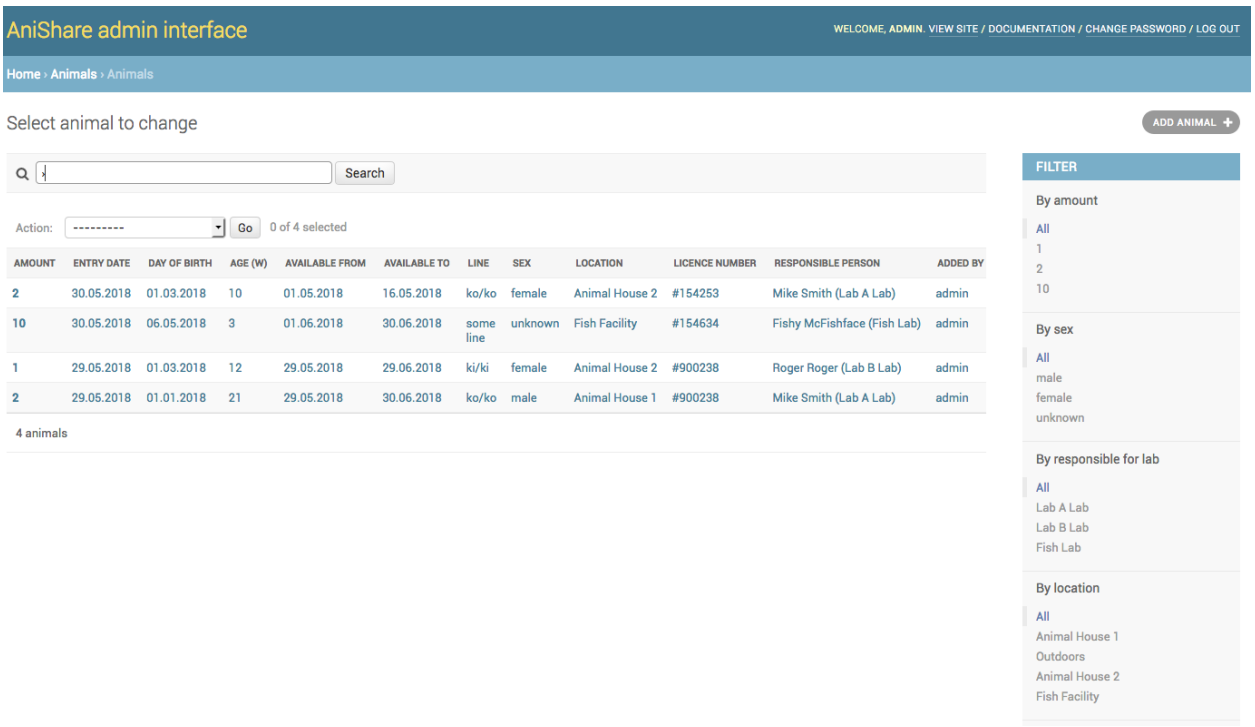
This will listen on `http://localhost:8000`, so browse to the admin page `http://localhost:8000/admin` and you should see this after login:



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.models.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 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)
        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')
```

(continues on next page)

(continued from previous page)

```
        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)
```

3.4.1 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!

3.4.2 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

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.**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'

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

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

```
"""
Admin module
"""
from datetime import datetime, timedelta
from django.contrib import admin
from django import forms
from django.conf import settings
from .models import Animal, Person, Lab, Location, Organ
from rangefilter.filter import DateRangeFilter, DateTimeRangeFilter


admin.site.site_header = 'AniShare admin interface'
admin.site.site_title = 'AniShare'
admin.site.index_title = 'Welcome to AniShare'

class AnimalForm(forms.ModelForm):
    """
    Form for animal editing in admin
    """
    class Meta:
        model = Animal
        fields = ('amount', 'animal_type', 'day_of_birth',
                  'available_from', 'available_to', 'sex', 'database_id',
                  'lab_id', 'line', 'location', 'responsible_person',
                  'licence_number', 'mutations', 'comment', 'new_owner', )

    def clean(self):
        available_from = self.cleaned_data.get('available_from')
        available_to = self.cleaned_data.get('available_to')
        day_of_birth = self.cleaned_data.get('day_of_birth')
#         self.author =  request.user
        if available_from > available_to:
            raise forms.ValidationError("Dates are incorrect")
        if day_of_birth and (
                (datetime.now().date() -  day_of_birth) <=
                timedelta(days=settings.MIN_SHARE_DURATION_PUPS)):
            if available_to - available_from <= timedelta(days=settings.MIN_SHARE_
↪DURATION_PUPS):
                raise forms.ValidationError(
                    "Minimum share duration for pups must be {} days!".format(
                        settings.MIN_SHARE_DURATION_PUPS))
        elif available_to - available_from <= timedelta(days=settings.MIN_SHARE_
↪DURATION):
            raise forms.ValidationError(
                "Minimum share duration must be {} days!".format(settings.MIN_SHARE_
↪DURATION))
        return self.cleaned_data

@admin.register(Person)
class PersonAdmin(admin.ModelAdmin):
    """
    ModelAdmin for Person model
```

(continues on next page)

(continued from previous page)

```
    """
    list_display = ('name', 'email', 'responsible_for_lab')
    search_fields = ('name', 'email', 'responsible_for_lab__name')
    ordering = ('name', )

@admin.register(Location)
class LocationAdmin(admin.ModelAdmin):
    """
    ModelAdmin for Location model
    """
    list_display = ('name',)
    search_fields = ('name',)

def clear_claim(modeladmin, request, queryset):
    """
    Convenience Function to delete a claim from several selected animals
    """
    queryset.update(new_owner='')
    clear_claim.short_description = "Clear 'new_owner' from selected animals"


@admin.register(Animal)
class AnimalAdmin(admin.ModelAdmin):
    """
    ModelAdmin for Animal model
    """
    list_display = ('amount', 'entry_date', 'day_of_birth', 'age', 'available_from',
                    'available_to', 'line', 'sex', 'location', 'licence_number',
                    'responsible_person', 'added_by', 'new_owner')
    list_display_links = ('amount', 'entry_date', 'day_of_birth', 'age',
                          'available_from', 'available_to', 'line', 'sex',
                          'location', 'licence_number', 'responsible_person',
                          'added_by', 'new_owner')
    search_fields = ('amount', 'database_id', 'lab_id', 'day_of_birth',
                     'line', 'sex', 'location__name', 'new_owner', 'licence_number',
                     'mutations', 'available_from', 'available_to', 'responsible_
↪person__name',
                     'responsible_person__email', 'added_by__email')
    autocomplete_fields = ['responsible_person']
    list_filter = ('amount', 'sex', 'responsible_person__responsible_for_lab',
                   ('day_of_birth', DateRangeFilter),
                   'location', 'licence_number', 'new_owner', 'added_by')
    radio_fields = {'sex':admin.HORIZONTAL}
    readonly_fields = ('creation_date', 'modification_date')
    form = AnimalForm
    actions = [clear_claim,]
    def age(self, obj):
        """Show the age in the admin as 'Age (w)' instead of 'age'"""
        return obj.age()
    age.short_description = 'Age (w)'
#    def formfield_for_foreignkey(self, db_field, request, **kwargs):
#        if db_field.name == 'author':
#            kwargs['queryset'] = get_user_model().objects.filter(username=request.
↪user.username)
#        return super(AnimalAdmin, self).formfield_for_foreignkey(db_field, request,␣
↪**kwargs)
    def save_model(self, request, obj, form, change):
```

(continues on next page)

(continued from previous page)

```
        if not obj.pk:
            # Only set added_by during the first save.
            obj.added_by = request.user
        super().save_model(request, obj, form, change)

@admin.register(Organ)
class OrganAdmin(admin.ModelAdmin):
    """
    ModelAdmin for Organ model
    """
    list_display = ('amount', 'animal_type', 'organ_type', 'entry_date', 'day_of_birth
↪',
                    'day_of_death', 'age', 'method_of_killing', 'killing_person',
↪'line',
                    'sex', 'location', 'licence_number', 'responsible_person', 'added_
↪by')
    list_display_links = ('amount', 'animal_type', 'organ_type', 'entry_date', 'day_
↪of_birth',
                          'day_of_death', 'age', 'method_of_killing', 'killing_person
↪', 'line',
                          'sex', 'location', 'licence_number', 'responsible_person',
↪'added_by')
    search_fields = ('amount', 'animal_type', 'organ_type', 'entry_date', 'day_of_
↪birth',
                     'day_of_death', 'age', 'method_of_killing', 'killing_person',
↪'line',
                     'sex', 'location', 'licence_number', 'responsible_person',
↪'added_by')
    autocomplete_fields = ['responsible_person']
    list_filter = ('amount', 'sex', 'responsible_person__responsible_for_lab',
                   ('day_of_birth', DateRangeFilter), ('day_of_death',␣
↪DateRangeFilter),
                   'location', 'licence_number', 'added_by')
    radio_fields = {'sex':admin.HORIZONTAL}
    readonly_fields = ('added_by', 'creation_date', 'modification_date')
#    form = OrganForm
    actions = [clear_claim,]

    def age(self, obj):
        """Show the age in the admin as 'Age (w)' instead of 'age'"""
        return obj.age()
    age.short_description = 'Age (w)'

    def save_model(self, request, obj, form, change):
        if not obj.pk:
            # Only set added_by during the first save.
            obj.added_by = request.user
        super().save_model(request, obj, form, change)

@admin.register(Lab)
class LabAdmin(admin.ModelAdmin):
    """
    ModelAdmin for Lab model
    """
    list_display = ('name', 'responsible_person')
    search_fields = ('name',)
```

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

description()
Return description of this model

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

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.

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

```
"""
This file describes all the models in the database.
"""
from datetime import datetime
from django.urls import reverse
from django.db import models
from django.contrib.auth.models import User

class Lab(models.Model):
    """
    Labs are only defined by a name and are referenced by
    Person(s) which are responsible (contact) person for this lab
    """
    name = models.CharField(max_length=200)
    def __str__(self):
        return self.name + ' Lab'
    def responsible_person(self):
        """
        Retrieve only the person(s) which are responsible for this lab.
        """
        persons = Person.objects.filter(responsible_for_lab=self)
        return ', '.join(i.name for i in persons)

class Person(models.Model):
    """
    The responsible (contact) person for each lab.
    This person gets an email when an animal is being claimed.
    """
    name = models.CharField(max_length=200)
    email = models.EmailField()
    responsible_for_lab = models.ForeignKey(Lab, on_delete=models.CASCADE, default=0)
    def __str__(self):
        return self.name + ' (' + str(self.responsible_for_lab) + ')'

class Location(models.Model):
    """
    Location of animals. Eg. animal house, fish facilities etc.
    """
    name = models.CharField(max_length=200)
    def __str__(self):
        return self.name

class Animal(models.Model):
    """
    Main model containing the animals.
    """
    amount = models.PositiveIntegerField(default=1,
                                         help_text="How many animals? (eg. fish in␣
↪tank)")
    animal_type = models.CharField(max_length=100, choices=(
        ('fish', 'fish'),
        ('mouse', 'mouse'),
    ),
                                   default='mouse')
    database_id = models.CharField(max_length=200, help_text="ID of animal in eg.␣
↪PYRAT")
```

(continues on next page)

(continued from previous page)

```
    lab_id = models.CharField(max_length=200, help_text="ID of lab in eg. PYRAT")
    creation_date = models.DateTimeField(null=False, auto_now_add=True)
    modification_date = models.DateTimeField(null=False, auto_now=True)
    entry_date = models.DateField(null=False, auto_now_add=True)
    day_of_birth = models.DateField()
    line = models.CharField(max_length=200, help_text="genetic trait of animal")
    sex = models.CharField(max_length=2, choices=(('m', 'male'), ('f', 'female'), ('u
↪', 'unknown')),
                           help_text='Select "unknown" if multiple animals.')
    location = models.ForeignKey(Location, on_delete=models.CASCADE,
                                 help_text='Where is the animal housed?')
    mutations = models.TextField(blank=True, null=True,
                                 help_text="Describe the mutations of this line in as␣
↪much detail as possible")
    licence_number = models.CharField(max_length=200)
    responsible_person = models.ForeignKey(Person, on_delete=models.CASCADE,␣
↪default=0,
                                           help_text='Person who is responsible in␣
↪the lab for dealing with the animals')
    available_from = models.DateField()
    available_to = models.DateField() # default=datetime.today() + timedelta(days=15))
    comment = models.TextField(blank=True, null=True,
                               help_text='Comments, such as individual organs to be␣
↪offered')
    new_owner = models.CharField(max_length=200, blank=True,
                                 help_text='Person claiming this animal for themselves
↪') # turn into foreignkey to auth_users?
    added_by = models.ForeignKey(User, unique=False, on_delete=models.CASCADE,␣
↪default=1)

    def age(self):
        """
        Return the age of the animal, calculated by the difference to either
        the current date or the available_to date
        """
#        return int((self.entry_date - self.day_of_birth).days / 7)
        now = datetime.today().date()
        if now < self.available_to:
            return int((now - self.day_of_birth).days / 7)
        return int((self.available_to - self.day_of_birth).days / 7)

    def available(self):
        """
        Returns True if the animal is still available
        """
        today = datetime.now().date()
        return (self.available_from <= today) and (today <= self.available_to)

    def get_absolute_url(self):
        """
        Get absolute url for this model. Important to link from the admin.
        """
        return reverse('animals:claim', kwargs={'primary_key': self.pk})

    def __str__(self):
        return "{} {} {}, {} id:{} [{}]".format(
            self.amount, self.get_sex_display(), self.animal_type, self.line, self.pk,
↪ self.day_of_birth)
```

(continues on next page)

(continued from previous page)

```
105
106     def description(self):
107         """
108         Return description of this model
109         """
110         return "id:{}, lab_id:{}, available:{}-{}, location:{}, mutations:{}".format(
111             self.database_id, self.lab_id, self.available_from,
112             self.available_to, self.location, "".join(self.mutations))
113
114
115
116 class Organ(models.Model):
117     """
118     Model containing the organs
119     """
120     amount = models.PositiveIntegerField(default=1,
121                                          help_text="How many organs?")
122     animal_type = models.CharField(max_length=100, choices=(
123         ('fish', 'fish'),
124         ('mouse', 'mouse'),
125         ('unknown', 'unknown'),
126         ),
127                                    default='mouse')
128     sex = models.CharField(max_length=2, choices=(('m', 'male'), ('f', 'female'), ('u
    ↪', 'unknown')),
129                            help_text='Select "unknown" if multiple animals.')
130     organ_type = models.CharField(max_length=100, choices=(
131         ('bladder', 'bladder'),
132         ('bone marrow', 'bone marrow'),
133         ('brain', 'brain'),
134         ('genitals', 'genitals'),
135         ('heart', 'heart'),
136         ('intestine', 'intestine'),
137         ('kidney', 'kidney'),
138         ('liver', 'liver'),
139         ('lungs', 'lungs'),
140         ('spleen', 'spleen'),
141         ('stomach', 'stomach'),
142         ('other', 'other'),
143         ),
144                                   )
145     day_of_birth = models.DateField()
146     day_of_death = models.DateField()
147     method_of_killing = models.CharField(max_length=100, choices=(
148         ('CO2', 'CO2'),
149         ('cervicale dislocation', 'cervicale dislocation'),
150         ('decapitation', 'decapitation'),
151         ('blood withdrawl', 'blood withdrawl'),
152         ('finale heart punction', 'finale heart punction'),
153         ('overdose anaesthetics', 'overdose anaesthetic'),
154         ('other', 'other'),
155         ),)
156     killing_person = models.EmailField(help_text='Email address of the person who is␣
    ↪responsible for killing the animal')
157     database_id = models.CharField(max_length=200, help_text="ID of animal in eg.␣
    ↪PYRAT")
158     lab_id = models.CharField(max_length=200, help_text="ID of lab in eg. PYRAT")
```

(continues on next page)

(continued from previous page)

```
159     entry_date = models.DateField(null=False, auto_now_add=True)
160     line = models.CharField(max_length=200, help_text="genetic trait of animal")
161     location = models.ForeignKey(Location, on_delete=models.CASCADE, help_text='Where␣
↪is the animal housed?')
162     licence_number = models.CharField(max_length=200)
163     responsible_person = models.ForeignKey(Person, on_delete=models.CASCADE,␣
↪default=0,
164                                            help_text='Person who is responsible in␣
↪the lab for dealing with the animals')
165     mutations = models.TextField(blank=True, null=True, help_text="Describe the␣
↪mutations of this line in as much detail as possible")
166     comment = models.TextField(blank=True, null=True,
167                                help_text='Comments, such as individual organs to be␣
↪offered')
168 #     new_owner = models.CharField(max_length=200, blank=True,
169 #                                  help_text='Person claiming this animal for␣
↪themselves') # turn into foreignkey to auth_users?
170     creation_date = models.DateTimeField(null=False, auto_now_add=True)
171     modification_date = models.DateTimeField(null=False, auto_now=True)
172     added_by = models.ForeignKey(User, unique=False, on_delete=models.CASCADE,␣
↪default=1)
173
174     def age(self):
175         """
176         Return the age of the animal, at the time of death
177         """
178         return int((self.day_of_death - self.day_of_birth).days / 7)
179
180     def available(self):
181         """
182         Returns True if the animal is still available
183         """
184         today = datetime.now().date()
185         return self.day_of_death >= today
186
187     def get_absolute_url(self):
188         """
189         Get absolute url for this model. Important to link from the admin.
190         """
191         return reverse('claim_organ', kwargs={'primary_key': self.pk})
192
193     def __str__(self):
194         return "{} {} {}, {} id:{} [{}]".format(
195             self.amount, self.get_sex_display(), self.animal_type, self.line, self.pk,
↪ self.day_of_birth)
196
197     def description(self):
198         """
199         Return description of this model
200         """
201         return "id:{}, lab_id:{}, available:{}-{}, location:{}, mutations:{}".format(
202             self.database_id, self.lab_id, self.day_of_birth,
203             self.day_of_death, self.location, "".join(self.mutations))
204
```

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.**AnimalIndexView**(***kwargs*)
Index / List view for all available animals. Generic ListView using the LoginRequiredMixin

Parameters

- **q** – query / search term to filter the results
- **show** – limit the results to 'current', 'archive', or all animals

get_queryset()
Return the latest additions to the Animals table

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.

class animals.views.**OrganIndexView**(***kwargs*)
Index / List view for all available Organs. Generic ListView using the LoginRequiredMixin

Parameters

- **q** – query / search term to filter the results
- **show** – limit the results to 'current', 'archive', or all Organs

get_queryset()
Return the latest additions to the Organs table

model
alias of animals.models.Organ

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

```
"""
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
"""
import operator
from functools import reduce
from datetime import datetime
from django import forms
from django.contrib import messages
from django.contrib.auth.decorators import login_required
from django.contrib.auth.mixins import LoginRequiredMixin
from django.contrib.syndication.views import Feed
from django.db.models import Q
from django.core.mail import EmailMessage
from django.http import HttpResponse
from django.http import HttpResponseRedirect
from django.shortcuts import get_object_or_404, render
from django.template.loader import render_to_string
#from django.urls import reverse
from django.views import generic

from .filters import AnimalFilter
from .models import Animal, Organ

@login_required
def claim(request, primary_key):
    """
    View to claim an animal.
```

(continues on next page)

(continued from previous page)

```
33
34      :param primary_key: the id/pk of the animal to retrieve
35
36      :returns: rendered page with the claim form
37                or 404 if animal not found
38      """
39      animal = get_object_or_404(Animal, pk=primary_key)
40      return render(request, 'animals/animal-claim.html', {'object': animal})
41
42  @login_required
43  def claim_organ(request, primary_key):
44      """
45      View to claim an organ.
46
47      :param primary_key: the id/pk of the organ to retrieve
48
49      :returns: rendered page with the claim form
50                or 404 if organ not found
51      """
52      organ = get_object_or_404(Organ, pk=primary_key)
53      return render(request, 'animals/organ-claim.html', {'object': organ})
54
55  class AnimalDetailView(LoginRequiredMixin, generic.DetailView):
56      """
57      Detail view for an animal.
58      This is rarely used, rather use the claim page.
59      """
60      model = Animal
61      template_name = 'animals/animal-detail.html'
62
63  class AnimalIndexView(LoginRequiredMixin, generic.ListView):
64      """
65      Index / List view for all available animals.
66      Generic ListView using the LoginRequiredMixin
67
68      :param q: query / search term to filter the results
69      :param show: limit the results to 'current', 'archive', or all animals
70      """
71      model = Animal
72      template_name = 'animals/index.html'
73      context_object_name = 'all_animals'
74      paginate_by = 100
75      def get_queryset(self):
76          """Return the latest additions to the Animals table"""
77          result = super(AnimalIndexView, self).get_queryset()
78          query = self.request.GET.get('q')
79          if query:
80              query_list = query.split()
81              result = result.filter(
82                  reduce(operator.and_, (Q(comment__icontains=q) for q in query_list)) |
83                  reduce(operator.and_, (Q(mutations__icontains=q) for q in query_
    ↪list)) |
84                  reduce(operator.and_, (Q(database_id__icontains=q) for q in query_
    ↪list)) |
85                  reduce(operator.and_, (Q(line__icontains=q) for q in query_list)) |
86                  reduce(operator.and_, (Q(lab_id__icontains=q) for q in query_list)) |
87                  reduce(operator.and_, (Q(location__name__icontains=q) for q in query_
    ↪list)) |
```

(continues on next page)

(continued from previous page)

```
                reduce(operator.and_, (Q(new_owner__icontains=q) for q in query_
↪list)) |
                reduce(operator.and_, (Q(responsible_person__name__icontains=q) for q␣
↪in query_list)) |
                reduce(operator.and_, (Q(licence_number__icontains=q) for q in query_
↪list))
            )
            return result
        try:
            show = self.kwargs['show']
        except KeyError:
            show = 'current'
        if show == 'archive':
            return Animal.objects.filter(available_to__lte=datetime.now().date()).
↪order_by('-entry_date')
        elif show == 'current':
            return Animal.objects.filter(available_to__gte=datetime.now().date()).
↪order_by('-entry_date')
        return Animal.objects.order_by('-entry_date')

class OrganIndexView(LoginRequiredMixin, generic.ListView):
    """
    Index / List view for all available Organs.
    Generic ListView using the LoginRequiredMixin

    :param q: query / search term to filter the results
    :param show: limit the results to 'current', 'archive', or all Organs
    """
    model = Organ
    template_name = 'animals/organs.html'
    context_object_name = 'all_organs'
    paginate_by = 100
    def get_queryset(self):
        """Return the latest additions to the Organs table"""
        result = super(OrganIndexView, self).get_queryset()
        query = self.request.GET.get('q')
        if query:
            query_list = query.split()
            result = result.filter(
                reduce(operator.and_, (Q(comment__icontains=q) for q in query_list)) |
                reduce(operator.and_, (Q(mutations__icontains=q) for q in query_
↪list)) |
                reduce(operator.and_, (Q(database_id__icontains=q) for q in query_
↪list)) |
                reduce(operator.and_, (Q(line__icontains=q) for q in query_list)) |
                reduce(operator.and_, (Q(lab_id__icontains=q) for q in query_list)) |
                reduce(operator.and_, (Q(location__name__icontains=q) for q in query_
↪list)) |
                reduce(operator.and_, (Q(killing_person__icontains=q) for q in query_
↪list)) |
                reduce(operator.and_, (Q(animal_type__icontains=q) for q in query_
↪list)) |
                reduce(operator.and_, (Q(method_of_killing__icontains=q) for q in␣
↪query_list)) |
                reduce(operator.and_, (Q(licence_number__icontains=q) for q in query_
↪list))
            )
```

(continues on next page)

(continued from previous page)

```
            return result
        return Organ.objects.order_by('-entry_date')

def send_email_animal(request):
    """
    Function to send an email about an animal being claimed.

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

    :param email: email address of the request user / new owner
    :param pk: primary_key of the animal(s) to be claimed
    :param count: how many animals are being claimed
    """
    email = request.POST['email']
    primary_key = request.POST['pk']
    count = request.POST['count']

    animal = Animal.objects.get(pk=primary_key)
    animal.new_owner = email
    amount_difference = int(animal.amount)-int(count)
    if amount_difference < 0:  # Save remainder of animals as new object
        messages.add_message(request, messages.ERROR, 'You cannot claim more animals then are available!')
        raise forms.ValidationError("You cannot claim more animals then are available!")
    animal.amount = count
    animal.save()  # Save the animal with the new owner
    messages.add_message(request, messages.SUCCESS,
                         'The entry {} has been claimed by {}.'.format(animal.pk, animal.new_owner))
    subject = "User {} claimed animal {} in AniShare".format(email, primary_key)
    message = render_to_string('email.html', {'email': email, 'object': animal, 'now': datetime.now()})

    msg = EmailMessage(subject, message, email, [animal.responsible_person.email, email])
    msg.content_subtype = "html"
    msg.send()
    if amount_difference > 0:  # If there were multiple animals, save the remainder of animals as a new object
        animal.pk = None
        animal.amount = amount_difference
        animal.new_owner = ''
        animal.save()
        messages.add_message(request, messages.SUCCESS, 'The amount of available animals in this entry has been reduced to {}.'.format(animal.amount))
    messages.add_message(request, messages.SUCCESS, 'An Email has been sent to <{}>.'.format(animal.responsible_person.email))

    return HttpResponseRedirect('/')

def send_email_organ(request):
    """
    Function to send an email about an animal being claimed.

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

```

(continues on next page)

(continued from previous page)

```
    :param email: email address of the request user / new owner
    :param pk: primary_key of the animal(s) to be claimed
    :param organs_wanted: organs wanted from the given animal
    """
    email = request.POST['email']
    primary_key = request.POST['pk']
    organs_wanted = request.POST['organs_wanted']

    organ = Organ.objects.get(pk=primary_key)
    subject = "AniShare User {} claimed organ(s) {}".format(email, organs_wanted)
    message = render_to_string('email.html', {'email': email, 'organs_wanted':organs_
↪wanted, 'object': organ, 'now': datetime.now()})

    msg = EmailMessage(subject, message, email, [organ.responsible_person.email,␣
↪email])
    msg.content_subtype = "html"
    msg.send()
    messages.add_message(request, messages.SUCCESS, 'An Email has been sent to <{}>.'.
↪format(organ.responsible_person.email))

    return HttpResponseRedirect('/organs/')


class LatestAnimalsFeed(Feed):
    """
    RSS Feed for new animals/organs.
    """
    title = 'Anishare animal/organ feed'
    link = '/animals/feed'
    description = 'Updates on animals/organs to share.'

    def __call__(self, request, *args, **kwargs):
        if not request.user.is_authenticated:
            return HttpResponse(status=401)
        return super().__call__(request, *args, **kwargs)

    def items(self):
        """
        Get latest animals as items.
        """
        from itertools import chain
        animals = Animal.objects.order_by('-entry_date')[:10]
        organs = Organ.objects.order_by('-entry_date')[:10]
        return chain(animals, organs)

    def item_title(self, item):
        """
        What to print as item title (use default __str__ of model).
        """
        return item

    def item_description(self, item):
        """
        What to print as item description (use default description from model).
        """
        return item.description()

```

(continues on next page)

(continued from previous page)

```
@login_required
def animal_list(request):
    f = AnimalFilter(request.GET, queryset=Animal.objects.order_by('-entry_date'))
    return render(request, 'animals/animal-index.html', {'filter': f})
```

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’))

```
"""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'))

"""

from django.urls import path
from . import views

app_name = 'animals'
urlpatterns = [
```

(continues on next page)

(continued from previous page)

```
#    path('', views.AnimalIndexView.as_view(), {'show':'current'}, name='animal-list
↪'),
    path('', views.animal_list, name='animal-list'),
#    path('archive', views.AnimalIndexView.as_view(), {'show':'archive'}, name=
↪'archive'),
#    path('all', views.AnimalIndexView.as_view(), {'show':'all'}, name='all'),
    path('claim/<int:primary_key>', views.claim, name='claim'),
    path('send_email_animal', views.send_email_animal, name='send_email_animal'),
    path('send_email_organ', views.send_email_organ, name='send_email_organ'),
    path('<int:pk>', views.AnimalDetailView.as_view(), name='animal-detail'),
    path('feed', views.LatestAnimalsFeed(), name='feed')
]
```

Index